Resource Optimization Algorithms for Virtual Private Networks Using the Hose Model

by

Monia Ghobadi
B.Sc., Sharif University of Technology, Tehran, Iran, 2005

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Resource Optimization Algorithms for Virtual Private Networks Using the Hose Model

by

Monia Ghobadi
B.Sc., Sharif University of Technology, Tehran, Iran, 2005

Supervisory Committee

Dr. Sudhakar Ganti (Department of Computer Science)
Co-Supervisor

Dr. Gholamali C. Shoja (Department of Computer Science)
Co-Supervisor

Dr. Ulrike Stege (Department of Computer Science)
Departmental Member

Dr. Lin Cai (Department of Electrical and Computer Engineering)
External Examiner

Supervisory Committee

Dr. Sudhakar Ganti (Department of Computer Science)
Co-Supervisor

Dr. Gholamali C. Shoja (Department of Computer Science)
Co-Supervisor

Dr. Ulrike Stege (Department of Computer Science)
Departmental Member

Dr. Lin Cai (Department of Electrical and Computer Engineering)
External Examiner

Abstract

Virtual Private Networks (VPNs) provide a secure and reliable communication between customer sites over a shared network. With increase in number and size of VPNs, providers need efficient provisioning techniques that adapt to customer demands. The recently proposed hose model for VPN alleviates the scalability problem of the pipe model by reserving for aggregate ingress and egress bandwidths instead of between every pair of VPN endpoints. Existing studies on quality of service guarantees in the hose model either deal only with bandwidth requirements or regard the delay limit as the main objective ignoring the bandwidth cost. In this work we propose a new approach to enhance the hose model to guarantee delay limits between endpoints while optimizing the bandwidth provisioning cost. We connect VPN endpoints using a tree structure and our algorithm attempts to optimize the total bandwidth reserved on edges of the VPN tree. Further, we introduce a fast and efficient algorithm in finding the shared VPN tree to reduce the total provisioning cost compared to the results proposed in previous works.

Our proposed approach takes into account the user preferences in meeting the delay limits and provisioning cost to find the optimal solution of resource allocation problem. Our simulation results indicate that the VPN trees constructed by our proposed algorithm meet maximum end-to-end delay limits while reducing the bandwidth requirements as compared to previously proposed algorithms.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

VPN	Virtual Private Network
QoS	Quality of Service
VoIP	Voice over IP
VCoIP	Video Conferencing over IP
IP-TV	IP Television
SLA	Service Level Agreement
MPLS	Multi-Protocol Label Switching
MDStT	Minimum Diameter Steiner Tree
LCLD	Least-Cost-Least-Delay
ISP	Internet Service Provider
OBDSTP	Optimal Bandwidth and Delay-constrained Shared Tree Problem
OBDST	Optimal Bandwidth and Delay-constrained Shared Tree
OBST	Optimal Bandwidth-constrained Shared Tree
ODST	Optimal Delay-constrained Shared Tree
HIST	Hierarchical Iterative Spanning Tree

Acknowledgments

I would like to express my deepest gratitude, first and foremost, to my supervisors, Dr. Ganti and Dr. Shoja for supervision, advice, and guidance from the very early stage of this research as well as giving me extraordinary experiences through out the work. Above all and the most needed, they provided me unflinching encouragement and support in various ways. I would have been lost without them.

I would also like to acknowledge Dr. U. Stege and Dr. L. Cai for taking time out from their busy schedules to serve as my committee members.

I wish to thank my family for their unwavering support and encouragement. One could not have a closer, more loving family. I appreciate you always being there for me.

Chapter 1. Introduction

1.1. Description

Globalization has revolutionized the business world in the last couple of decades. Instead of simply dealing with local or regional concerns, many businesses now have to think about global markets. Many companies have facilities spread out around the world, and hence they all need a way to maintain fast, secure and reliable communications wherever their offices are. Until fairly recently, this meant the use of leased lines to maintain a wide area network (WAN). Leased lines provided a company with a way to expand its private network beyond its immediate geographic area. A WAN had obvious advantages over a public network, like the Internet, when it came to reliability, performance and security. But maintaining a WAN, particularly when using leased lines, can be quite expensive and often the cost increases with distance between the offices.

As the popularity of the Internet grew, businesses turned to it as a means of extending their own private networks. First came intranets, which are password-protected sites designed for use only by the company employees. Now, many companies are creating their own Virtual Private Network (VPN) to accommodate the needs of remote employees and distant offices.

As the VPN market continues to expand, related paradigms are evolving to deliver new levels of capability and attracting wider interest from the academic research community. Current research in the area of VPNs revolves around many distinct sub-problems. In the next section we will present a few of the most relevant problems.

1.2. Current Research

VPNs allow service providers to host multiple private client networks over a common physical infrastructure. This is done using advanced hardware and software protocols, and these technologies have seen many evolutions so far.

Early VPN offerings utilized specialized layer 2 hardware and protocol technologies such as asynchronous transfer mode (ATM) [3] and frame relay [7] to implement tunnelled connectivity. The Layer 2 Virtual Private Network (L2VPN) is defined as a VPN that transports layer 2 frames across a shared IP network. These frames can be frame relay protocol data units, ATM cells, or even Ethernet frames as they are carried across the network using one of several different tunnel-encapsulation schemes [34].

Layer 3 VPNs (L3VPNs) based on RFC 2547 [5] have become increasingly popular over the last decade. They influenced the emergence of newer IP-based quality of service (QoS) and multi-protocol label switching (MPLS) [1] paradigms and inclusive security mechanisms such as IPSec [11]. These newer VPN architectures now offer key advantages in terms of converged data, voice, and video capabilities.

With the growth in size and number of VPNs and the uncertainties in the traffic patterns of customers, providers are faced with new challenges in efficient provisioning and capacity planning for VPN networks in order to satisfy the customer's Service Level

Agreements (SLA). To address these concerns, many works have been done in this context [2, 9, 18, 21, 22, 23, 27, 28, 32, 35, 37] where various provisioning problems are considered based on the routing structure, resource sharing capability, and any knowledge of VPN's traffic patterns between endpoints.

In this work, we focus on the resource provisioning and timeliness problems in VPNs.

1.3. Motivation

Due to progress in security and the overwhelming success of IP networking technologies, the number of VPNs that a service provider must support and the number of endpoints per VPN has grown tremendously. Moreover, communication patterns between endpoints are increasingly difficult to forecast and as such, users often are unable to predict and specify loads between pairs of VPN endpoints. Thus, it is typically difficult to specify QoS requirements on a point-to-point basis, which was the approach when deploying VPN services that used traditional solutions such as private line or frame relay services. Private lines isolate the traffic of a VPN from other flows and provide guaranteed bandwidth, loss, and delay characteristics. The emergence of IP technologies such as MPLS [1] and RSVP-TE [17] have made it possible to realize IP-based VPNs that can provide the end customers with QoS guarantees. Thus, an IP VPN service that replaces the traditional point-to-point connectivity between sites using legacy solutions must offer comparable performance, security and functionality.

There are two popular models for providing QoS in the context of VPNs—the *pipe* model [1] and the *hose* model [2]. The pipe model is a simple service model for an IP VPN which emulates the private line or frame relay service in which the VPN customer

specifies QoS requirements between every pair of VPN endpoints. Thus, the pipe model requires the customer to know its own complete traffic matrix. The hose model alleviates the shortcomings of the pipe model by specifying the QoS requirements per VPN endpoint and not every pair of endpoints.

Our goal is to address the resource management problem in VPNs and introduce algorithms that enable efficient resource provisioning with QoS guarantees. Our algorithms are based on the hose service model, which is a widely accepted service specification. As we will see in Chapter 2, existing studies on quality of service guarantees in the hose model either deal only with bandwidth requirements or regard the delay limit as the main objective ignoring the total provisioning cost. In this work we propose a new approach to enhance the hose model to guarantee end-to-end delay limits between endpoints while optimizing the provisioning cost. Further, we introduce a fast and efficient algorithm in finding a shared VPN tree with minimum total provisioning cost compared to the results proposed previously in [23]. We connect VPN endpoints using a tree structure and our algorithm attempts to optimize the total bandwidth reserved on edges of the VPN tree. Our proposed approach takes into account the user preferences in meeting the delay limits and provisioning cost in order to find the most optimal solution with respect to user specified parameters. Our simulation results indicate that the VPN trees constructed by our proposed algorithm meet the delay limits while reducing the bandwidth requirements as compared to previously proposed algorithms [8, 23].

1.4. Thesis Organization

This work describes a new solution to the resource provisioning problem and QoS guarantees in VPNs. The remainder of the thesis is organized as follows. Chapter 2 describes the VPN network model and some of the previous works in this area. Chapter 3 includes further details of the problem space, sets forth the assumptions that are made and provides detailed specification of our solution. This chapter also includes formal analysis of proposed algorithms. The simulations and performance evaluation results of our work are given in Chapter 4. Finally the thesis is concluded in Chapter 5 with a summary of major contributions and possible directions for future work.

Chapter 2. Background

A Virtual Private Network (VPN) is a group of computer systems connected as a private network that communicates over a public network. VPNs offer a cost-effective, scalable, and manageable way to create a private network over a public infrastructure such as a service provider's frame relay [7], ATM [3], or IP network [20]. For this reason, VPNs are deployed by businesses to meet their networking and communication needs and have rapidly emerged as leading solutions for multi-site enterprise communication demands. As illustrated in Figure 2.1, VPNs create a private means for communications between geographically distributed locations.

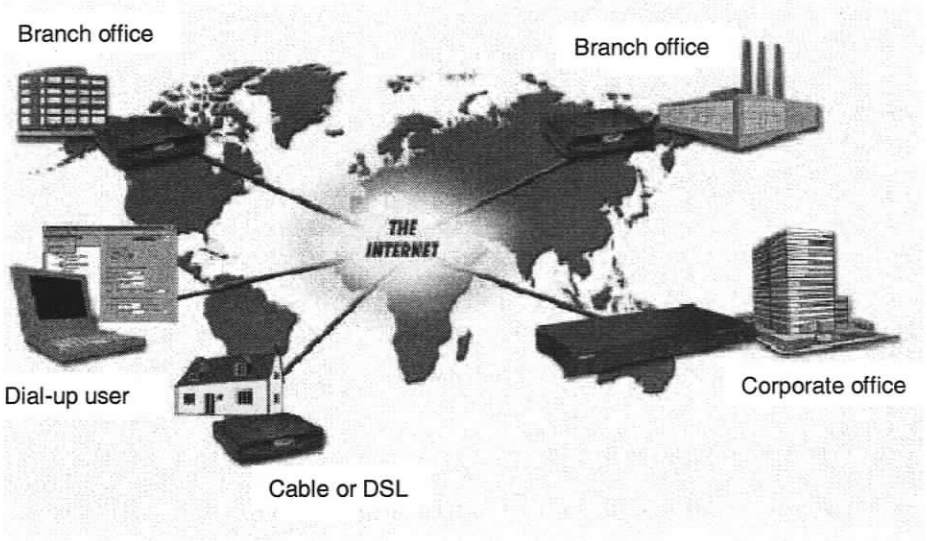


Figure 2.1: A Virtual Private Network [26]

An IP VPN is a private network constructed over an IP-based backbone such as the public Internet. There is a growing interest in the use of IP VPNs as a more cost effective way of building and deploying private communication networks for multi-site communications. Existing private networks can be generally categorized into two types [6]:

- Dedicated Wide Area Networks (WANs) that permanently connect together multiple sites.
- Dial-up networks that allow on-demand connections through the Public Switched Telephone Network (PSTN) to one or more sites in the private network.

VPNs need to meet certain customer requirements as follows [6]:

1. *Data Security*: In general, customers using VPNs require some form of data security. Depending on the deployment scenario, different levels of security may be needed within the provider's backbone network such that:
    a. All the traffic on the VPN is encrypted and authenticated,
    b. No one outside the VPN should be able to affect the security properties of the VPN. Also it must be impossible for an attacker to change the security properties of any part of the VPN, such as to weaken the encryption or to affect which encryption keys are used.
2. *Tunneling Mechanism*: The above requirement implies that VPNs must be implemented through some form of tunneling mechanism. Tunnelling mechanisms allow the traffic to be encrypted at the sender, moved over the backbone network like any other data, and then decrypted when it reaches the receiver. This encrypted

traffic acts like it is in a tunnel between the two hosts: even if attackers see the traffic, they cannot read it, and they cannot change the traffic without the changes being seen by the receiving party and therefore rejected.

3. *Quality of Service Guarantees*: Effective quality of service (QoS) capabilities will be needed to ensure that the traffic is delivered efficiently in the face of network congestion so that applications can deliver the quality of experience that users have come to expect from traditional leased line networks.

Our focus in this work is on the quality of service aspects of VPNs. Quality of service guarantees are important if the network capacity is limited, especially for real-time streaming multimedia applications such as Voice-over-IP (VoIP) [15], Video Conferencing over IP (VCoIP) [16, 29], and IP television (IP-TV) [30]. These applications often require fixed bit rate and are delay sensitive. In the following sections, we will explain the VPN service models and some existing solutions to provide QoS in VPNs.

2.1. VPN Service Models

Two popular models have been proposed for providing QoS in the context of VPNs: the "pipe" model [1] and the "hose" model [2].

The pipe model is a simple service model for an IP VPN which emulates the private line or frame relay service. As depicted in Figure 2.2, in the pipe model, a VPN customer purchases a set of customer-pipes, i.e., allocations of specific bandwidth on paths between every source-destination pair of the VPN endpoints. The network provider would need to provision adequate bandwidth along the path of each pipe to ensure that

the Service Level Agreement (SLA) is satisfied. The primary disadvantage of this approach is that it requires the customer to have precise knowledge of its own traffic matrix between all the VPN sites. Moreover, resources made available to a customer pipe cannot be allocated to other traffic.

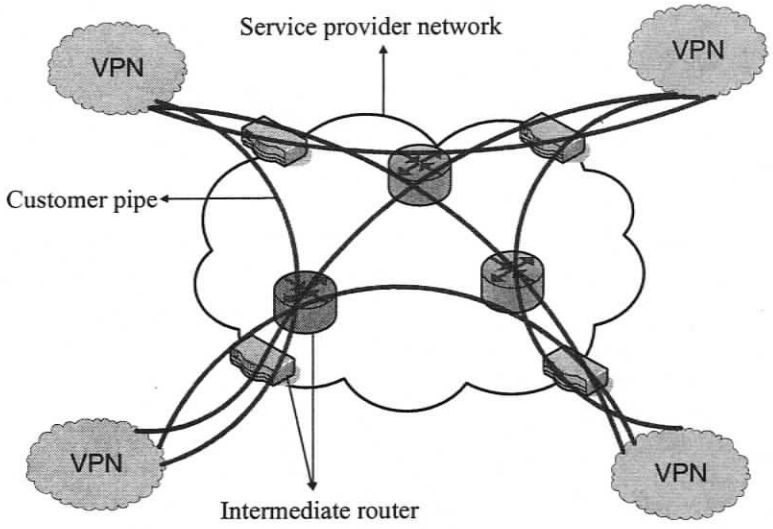


Figure 2.2: VPN pipe model

Due to the progress in security and the success of IP networking technologies, the number of endpoints per VPN is growing, and the communication patterns between endpoints are becoming increasingly difficult to predict. It is expected that users will be unwilling to, or simply unable to predict loads between pairs of endpoints. Similarly, it will become increasingly difficult to specify QoS requirements on a point-to-point basis, as is the conventional approach.

The hose model, introduced by Duffield et al. in [2], serves as both a VPN service interface as well as a performance abstraction. A hose offers performance guarantees at a given endpoint for the traffic to and from the set of all other endpoints in the VPN. Thus,

the hose service interface allows the customer to send traffic into the network without the need to predict point-to-point loads.

Figure 2.3 illustrates an example of the use of the hose model. Each VPN endpoint i is connected to the network by a hose, which is specified by its aggregate ingress and egress bandwidth ($B^{in}{}_i$ and $B^{out}{}_i$ respectively). $B^{in}{}_i$ is the amount of aggregate traffic from all endpoints to endpoint i and $B^{out}{}_i$ is the amount of aggregate traffic from endpoint i to all other endpoints of the same VPN. 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 other endpoints. As a result, in contrast to the pipe model, the hose model does not require a customer to know its own complete traffic matrix.

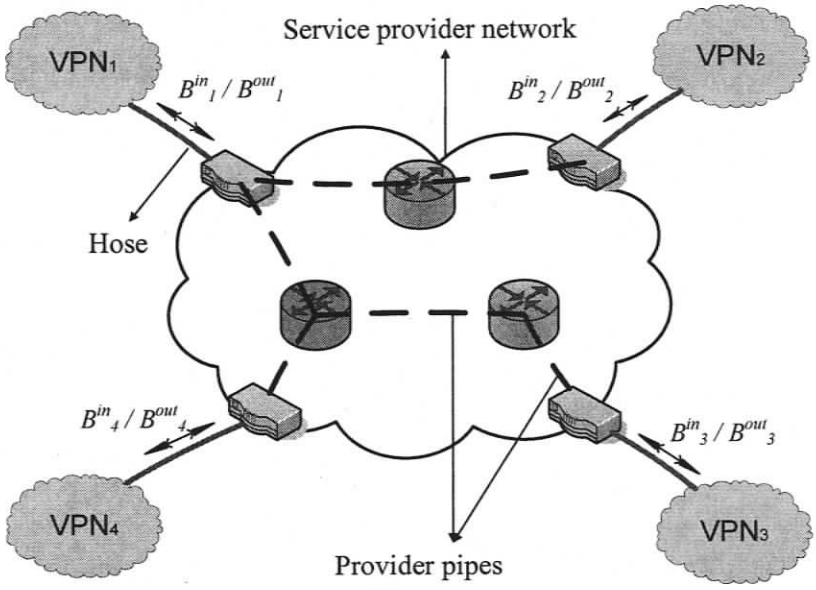


Figure 2.3: VPN hose model

In summary, the hose model provides customers with the following advantages over the pipe model [2]:

- *Ease of specification*: Only one ingress and egress bandwidth per VPN endpoint needs to be specified, compared to the pipe model in which the bandwidth for every pipe between pairs of VPN endpoints should be specified.
- *Flexibility*: Traffic to and from a VPN endpoint can be distributed arbitrarily over other endpoints as long as the ingress and egress bandwidths of each VPN endpoint are not violated.
- *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.
- *Characterization*: The hose model's requirements are easier to describe as the variability in the individual source-destination traffic is smoothed by aggregation into hoses.

From the above discussion, it follows that the hose model provides the VPN customers with a simple mechanism for specifying bandwidth requirements and enables the VPN service providers to utilize network bandwidth more efficiently. However, in order to exploit 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 minimized. Intuitively, in order to conserve bandwidth and utilize the multiplexing benefits of the hose model, paths entering into and originating from each hose endpoint need to share as many links as possible [9]. Thus, appropriate hose provisioning

algorithms need to be developed to ensure that the amount of bandwidth reserved in order to meet the hose traffic requirements is minimized.

As an example of resource allocation using pipe and hose models, consider the graph depicted in Figure 2.4. It contains a network with seven nodes and three VPN endpoints. Nodes 1, 2, and 3 are the VPN endpoints with ingress / egress bandwidths equal to 1/1, 2/2, and 2/2 units, respectively.

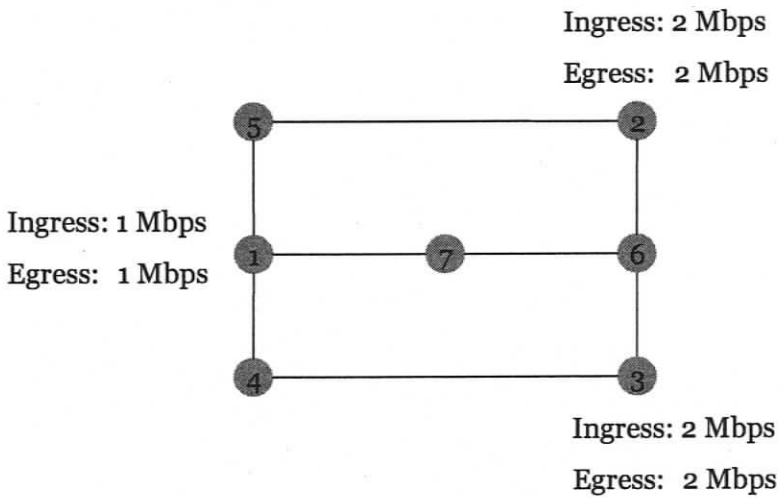
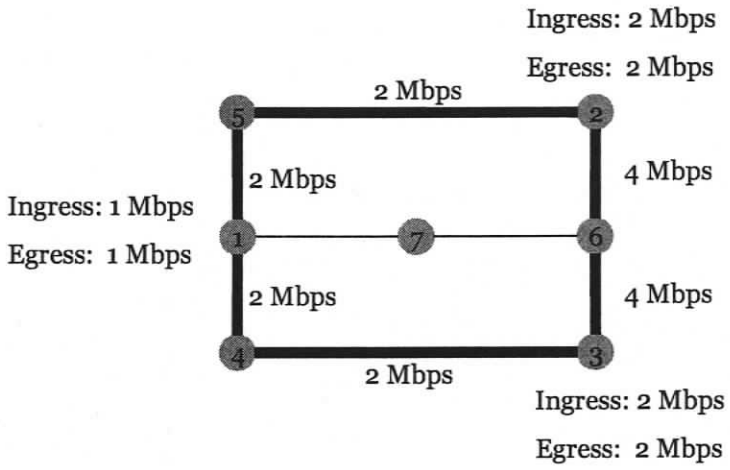


Figure 2.4: A sample network

The bold links in Figure 2.5 show the bandwidth reserved on relevant links of the network when an independent shortest path approach is used to connect VPN endpoints. One may see this as the classical approach used in the pipe model. Here, the shortest path between node 1 and node 2 passes through node 5, while the shortest path between node 1 and node 3 passes through node 4. This causes an overall network reservation of 16 units.



Total provisioning cost : 16 Mbps

Figure 2.5: The pipe model

In Figure 2.6, the hose model lets all paths pass through node 6, resulting in a multiplexing gain and an overall reservation of only 12 units. Note that the path from node 1 to node 2 passing through node 6 in Figure 2.6 traverses more hops than the path through node 5 in Figure 2.5. However, since VPN endpoint 1 may not receive or send more than one unit of traffic, the bandwidth reserved on each of the links from node 1 to node 6 is one unit (in each direction) and is shared between two paths. This example shows how the hose model is able to save bandwidth by sharing links in a given network.

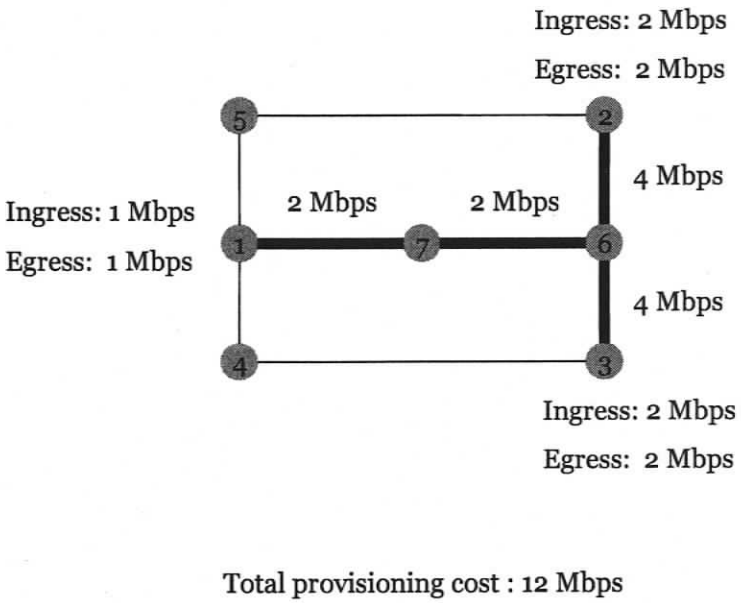


Figure 2.6: The hose model

2.1.1. VPN Network Model

A VPN network is modeled as a connected graph $G = (V, E)$ where V is the set of nodes and E is the set of bidirectional links connecting the nodes. Each link (u, v) is associated with two QoS metrics: maximum bandwidth (capacity) over the link and the delay between link endpoints. The delay value of a path is defined as the sum of the delay values of all the links along the path.

The VPN specification in the hose model includes:

1. A subset of endpoints $P \subseteq V$ corresponding to the VPN endpoints, and

2. for each VPN endpoint $p \in P$, the associated ingress and egress bandwidths B^{in}_p and B^{out}_p, respectively.

The basic problem of finding a reservation of minimum cost in the hose model may be subject to variable conditions. First, ingress and egress bandwidth $B^{in}{}_p$ and $B^{out}{}_p$, $p \in P$ may be defined in three different ways:

- The symmetric case: $B^{in}{}_p = B^{out}{}_p, \forall\, p \in P.$
- The sum-symmetric case: $\sum_{p \in P} B_p^{in} = \sum_{p \in P} B_p^{out}, \forall p \in P.$
- The asymmetric or general case: $B^{in}{}_p$ and $B^{out}{}_p$ are arbitrary values.

Additional variations arise from different approaches for implementing the resource provisioning in the hose model:

- *Pipe mesh* approach: This approach was first suggested in [2] to implement the hoses with a mesh of pipes between the VPN endpoints. This can be viewed as the traditional pipe model in which a hose is implemented by a mesh of customer-pipes between VPN endpoints. For a given customer-pipe from the ingress VPN endpoint i to the egress VPN endpoint j, $\min(B^{out}{}_i\,,\, B^{in}{}_j)$ units of bandwidth is reserved on each link along the path.
- *Multiple source-based trees* approach: This approach builds a source-based tree to implement each hose. The provider needs to build one tree per each hose resulting in a total of $|P|$ source-based trees. For a given source based tree T rooted at VPN endpoint i, assume that T_v denotes the connected component of T containing node v when link (u, v) is deleted from T. In this case, the traffic passing through link (u, v) can only originate from the VPN endpoint i to the other VPN endpoints in T_v. The traffic that VPN endpoint i can send is bounded by $B^{out}{}_i$, and the traffic that T_v can receive cannot exceed $\sum_{j \in P \cap T_v} B_j^{in}$. Thus the

bandwidth reserved for link (u, v) is given by $C_T(u,v) = \min\{B_i^{out}, \sum_{j \in P \cap T_v} B_j^{in}\}$ [8].

Therefore, the total bandwidth reserved for tree T is given by $C_T = \sum_{(u,v) \in T} C_T(u,v)$.

In the above formula, only links directing away from endpoint i need to be considered. This is because the source-based tree is only used to send traffic from i to the other VPN endpoints. Therefore, if link (u, v) is a link in tree T directing away from i, no bandwidth needs to be reserved on link (v, u) going in the other direction.

- *Shared tree* approach: This approach uses a single shared tree to connect all the VPN endpoints. The traffic between VPN endpoints u and v (from u to v or from v to u) is routed along the unique path P_{uv} in T. The resource provisioning objective is to find a shared tree with minimum total provisioning cost.
- *General subgraph* approach: In this approach the structure of the VPN may form a tree or a general subgraph in which there is a path P_{ij} for each (ordered) VPN endpoints (i, j). The traffic from each VPN endpoint pair (i, j) will be routed on the path P_{ij}. The resource provisioning objective is to find a feasible solution minimizing the total required bandwidth to be reserved on set of edges.

The above variation of the resource provisioning problem in VPNs has been studied in [22, 28]. However, for many key applications of VPNs, we need to impose the requirement that the union of the edges in the path set P_{ij} should form a tree. This requirement is motivated by applications that require guarantees on delay and throughput [23] to preserve that the traffic from endpoint i to j is traversed over the same path from j to i. Important applications of this type are VoIP and IP-TV, where strong real-time

requirements prevail. Moreover, the underlying virtual private network should be structurally simple enough to facilitate routing. In summary, a VPN shared tree has several benefits, as listed below [9, 23, 32]:

1. *Sharing of Bandwidth Reservation*: A 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*: A tree structure scales better in terms of adding new endpoints to the VPN especially for networks with large number of VPN endpoints. This is because only one path from the new endpoint to one of the nodes in the tree is required rather than a path to every endpoint in the VPN.

3. *Simplicity of Routing*: The structural simplicity of trees ensures that Multi-Protocol Label Switching (MPLS) [1], the predominant standard for setting up paths between pair of VPN endpoints, is considerably simplified since 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.

In order to take advantage of the above benefits, in this study we will connect VPN endpoints using a tree structure.

The following sections provide the research objectives and background for provisioning algorithms in VPNs.

2.2. Research Objectives

With increasing popularity of IP VPNs for enterprise networking solutions, providers are faced with new challenges in provisioning and operating a complex and growing VPN infrastructure. In the presence of accurate information about customer traffic profile and available network resources, a provider can make provisioning decisions while ensuring SLAs are met. However, with the growth in size and number of VPNs and the uncertainties in the traffic patterns of customers, providers are faced with new challenges in efficient provisioning, QoS guarantees, and capacity planning for their networks.

The nature of the SLA between a customer and a service provider is driven by the traffic characteristics and QoS requirements of the customer applications that make use of the VPN. For example, a VoIP VPN service might require tight bounds on the packet loss rate, delay, and possibly jitter. On the other hand, a data-only VPN service might have relatively less stringent or no delay limits.

Although the hose model provides customers with simpler and more flexible SLAs, the model presents the provider with a more challenging problem of resource management. On the other hand, VPNs are being used by customers as a replacement for networks constructed using private lines and should, at the very least, provide a comparable quality of service. However, it is difficult to provide QoS guarantee in the hose model since VPN customers specify QoS requirements per VPN endpoint and not for every pair of endpoints.

This thesis presents ways to provision VPNs in order to guarantee quality of service while saving cost. We apply the concept of the hose model presented in [2] and present

new ideas and methods to improve on the previous research in this area. In the following section we will describe current VPN provisioning algorithms in the hose model.

2.3. Previous Work

A number of provisioning algorithms for VPNs in the hose model have been proposed. In [2], Duffield et al. introduced the hose model for provisioning a VPN. In their work, a hose is implemented with a source-based tree or a Steiner tree [14] and a factor of two to three in capacity savings over the pipe model is achieved. The authors suggest that using a Steiner tree to connect VPN endpoints would optimize the total provisioning cost.

Further, in [9] and [32] [1], the optimal bandwidth allocation problem was formulated as follows:

"Given a set of VPN endpoints P and their ingress B_p^{in} and egress B_p^{out} bandwidths for each VPN endpoint $p \in P$, compute a shared VPN tree T, connecting VPN endpoints for which the total bandwidth reserved on edges of T is minimum".

Their work gives algorithms and results for the above problem summarized as:

- With assumption of infinite bandwidth capacity on the links and symmetric ingress and egress bandwidths for all VPN endpoints $p \in P$, a Breadth First Search (BFS) based polynomial algorithm will compute the optimal provisioning tree.

[1] [9] is the journal version of [32].

- With assumption of infinite bandwidth capacity on the links and asymmetric ingress and egress bandwidths for VPN endpoints, the authors proved that computing the optimal reservation is an NP-hard problem.

The total bandwidth cost of tree T, is calculated in [9] as follows:

For a given shared tree T, and a link (u, v), let $P_u^{(u,v)}$ (or $P_v^{(u,v)}$) denote the set of VPN endpoints in the connected component of T containing node u (or v) when link (u, v) is deleted from T. Since all traffic from VPN endpoint u to VPN endpoint v traverses the unique path in the VPN tree T, the traffic from node u to v cannot exceed $\min\{\sum_{l\in P_u^{(u,v)}} B_l^{out}, \sum_{l\in P_v^{(u,v)}} B_l^{in}\}$, that is the minimum of the cumulative egress bandwidths of endpoints in $P_u^{(u,v)}$ and the sum of ingress bandwidths of endpoints in $P_v^{(u,v)}$. This is because the only traffic that traverses link (u, v) from u to v is the traffic originating from endpoints in $P_u^{(u,v)}$ and directed toward endpoints in $P_v^{(u,v)}$. The bound on the former is $\sum_{l\in P_u^{(u,v)}} B_l^{out}$, while the latter is bounded by $\sum_{l\in P_v^{(u,v)}} B_l^{in}$. Thus, the bandwidth to be reserved on link (u, v) of T in the direction from u to v is given by $C_T(u,v) = \min\{\sum_{l\in P_u^{(u,v)}} B_l^{out}, \sum_{l\in P_v^{(u,v)}} B_l^{in}\}$. Similarly, the bandwidth that must be reserved on link (v, u) in the direction from v to u can be shown to be $C_T(v,u) = \min\{\sum_{l\in P_u^{(u,v)}} B_l^{in}, \sum_{l\in P_v^{(u,v)}} B_l^{out}\}$. Note that in case of asymmetry bandwidths, $C_T(u, v)$ may not be equal to $C_T(v, u)$. Therefore, the total bandwidth reserved for tree T is given by:

$$C_T = \sum_{(u,v)\in T} C_T^{(u,v)}. \qquad (2.1)$$

The authors in [9] formulated the bandwidth allocation problem as an integer linear program and a 10-approximation algorithm is introduced by solving the linear program

relaxation and rounding the fractional solution. The simulation results show that their algorithms perform better than BFS-based and Steiner tree algorithms. It was proved that the proposed approximation algorithm will find a solution with cost at most a factor 10 times the optimum solution [9].

In [23], Gupta et al. studied the VPN provisioning problem under different scenarios: symmetric versus asymmetric ingress and egress bandwidths, as well as using a tree versus using a graph to connect VPN endpoints and the following results are given:

- For asymmetric bandwidths with an assumption that links have infinite bandwidth capacity, the approximation ratio of the approach to build a shared tree (named as *AsymT* algorithm) is improved to 9.002 from 10.
- For symmetric bandwidths with an assumption that links have infinite bandwidth capacity, it is shown that the cost of the optimal tree is at most twice as large as the cost of the optimal reservation, which may not form a tree.
- For symmetric bandwidths with an assumption that links have finite bandwidth capacity, it is NP-hard to check whether there is a feasible shared tree. A polynomial algorithm is given to compute a shared tree whose cost is within a constant factor of the optimum and that violates edge capacities at most by a constant factor.

The bandwidth efficiency of the hose model is studied in [25] where the over provisioning factor of the model is evaluated in networks with various sizes and node densities. The authors conclude that hose model performs better in reducing blocking probability, decreasing traffic loss, and ease of implementation over the pipe model. In [22], a randomized 5.55-approximation algorithm for the general VPN design problem is

given that finds a set of paths $\{P_{ij}\}$ between each ordered pair (i, j) of VPN endpoints such that all valid traffic matrices can be routed using these paths. In their approach, the union of the paths may not necessarily form a tree and thus their solution is not suitable for VPNs that carry delay sensitive applications. Another shortcoming of this solution is that it may not perform well for real network topologies as they are not completely random.

In [28], a multi-path routing provisioning approach is proposed for the hose model. The authors ran 6200 series of experiments with small connected random graphs with 3 to 5 nodes. Their results indicate that multi-path routing had reduced reservation cost compared to shared tree routing for roughly 20% of the instances with 3 nodes, 25% of the instances with 4 nodes, and 17% of the instances with 5 nodes. In the cases where the multi-path routing had reduced reservation cost compared to tree routing, the cost reduction was 8.6% on the average. The authors viewed the results as an indication that multi-path routing has the potential of offering bandwidth savings for VPN reservations in the hose model. However, as discussed earlier, the target of our work is to find an optimal shared tree and thus multi-path routing is not suitable for our work.

In [24] it is shown that for sum-symmetric tree routing, the optimal solution may be computed in polynomial time and its cost is within a factor of three of the optimal solution's cost. Further, in [37] the authors enhanced the algorithm in [9] and [32] to consider the case where the links have finite capacities under the assumption that ingress and egress bandwidths are symmetric. In our work, however, the ingress and egress bandwidths can also be asymmetric.

The above studies on resource provisioning, and QoS guarantees in the hose model deal only with bandwidth requirements and do not consider providing end-to-end delay bound guarantee between VPN endpoints, which is an important metric in VPNs that carry delay-sensitive applications such as VoIP, IP-TV, and VCoIP.

In [8], the authors enhanced the original hose model to allow for specification of delay limits between VPN endpoints. They proposed three provisioning approaches for the enhanced hose model: the pipe mesh approach, the multiple source-based trees approach, and the shared tree approach. Using theoretical analysis and simulation results the authors concluded that the shared tree approach is appealing because of its low provisioning cost and ease of routing and restoration.

Enhancing the hose model to include the delay bound requirement is done by grouping applications that use the VPN into different delay classes characterized by their end-to-end delay limit requirements. This delay limit must hold between every pair of endpoints. Thus, the network may identify a set of delay classes and each delay class is characterized by its maximum allowable end-to-end delay. To construct a shared tree supporting the delay limit the authors in [8] proposed a solution based on Minimum Diameter Steiner Tree (MDStT) algorithm. The diameter of a Steiner tree is defined as the maximum delay between any two VPN endpoints. Thus, the maximum allowable end-to-end delay limit that can be supported by the tree can be obtained by finding the MDStT. To find the solution, the authors proved that MDStT problem is equivalent to the absolute subset 1-center problem of a general graph.

An absolute subset 1-center of a graph $G = (V, E)$ with respect to a subset $P \subseteq V$ is a point x (on a link or at one of the nodes) which represents the position at which the

greatest distance from x to any destination in P is minimized. The distance from x to a given destination in P is defined as the length of the shortest path (with respect to link weights) connecting them.

In [8] the MDStT algorithm is developed based on the algorithms for the absolute center problem [10] [19]. The main idea of the algorithm is to identify a local absolute subset 1-center for each link in the graph. The global absolute center can be found by selecting the optimal one from the $|E|$ local centers.

The MDStT algorithm supports the lowest delay limit using a tree structure. However to build a low provisioning cost tree, the authors suggested a Least-Cost-Least-Delay (LCLD) approach which tries to reduce the provisioning cost based on minimum hop counts while maintaining the delay limit. The LCLD algorithm satisfies the delay limit, but the approach to reduce the provisioning cost can be improved.

2.4. Our New Contributions

Our work aims to address the shortcomings of previous works in the following ways:

- Construct a shared tree that provides maximum allowable end-to-end delay guarantee between VPN endpoints.
- Introduce a more efficient algorithm to reduce the provisioning cost of such tree while satisfying the delay limit.
- Take into account the user preferences in meeting the delay limit and reducing bandwidth cost.

- Introduce a hierarchical algorithm for VPN provisioning problem in the hose model.

2.5. Summary

The preceding sections have given a broad background to the problems of resource provisioning in the virtual private networks. The basics of calculating the cost of a VPN provisioning were explained. We then covered the issue of meeting the maximum allowable end-to-end delay limit between VPN endpoints. In the following chapter we will give a more detailed description of our work with our basic assumptions.

Chapter 3. Problem Statement, Proposed Solution and Methodology

The general context and scope of this project were given in the preceding chapters where resource provisioning algorithms for Virtual Private Networks (VPNs) in the hose model were introduced and previous works were described. The deficiencies of prior solutions as they relate to our problem space were also described. At the end of Chapter 2, an overview of the latest ideas on how to improve upon the resource optimization in Virtual Private Networks was also presented. In particular, the issues of quality of service (QoS) guarantees and minimizing the cost of provisioning algorithms are addressed.

In this chapter we define our study in greater detail and explain our approach to improve upon the issues addressed by previous efforts. In Section 3.1, specification of the problem will be provided through scope clarification, a set of assumptions and a concise problem statement. Sections 3.2 and 3.3 present the process by which we show improvement over the previous results in this area. The correctness properties of proposed algorithms are provided in Section 3.4. Finally, in Section 3.5 time complexity analysis of the proposed algorithms will be presented.

3.1. Problem Definition

3.1.1. Scope and Assumptions

The work detailed in this thesis focuses on resource provisioning algorithms and QoS guarantee for Virtual Private Networks in the hose model. The set of assumptions in our work is as follows:

- The ingress and egress bandwidths of VPN endpoints can be asymmetric.
- In the context of QoS guarantee in VPNs, only delay and aggregate bandwidth guarantees have been considered.
- Links of the network have enough capacity to satisfy the required bandwidth provisioning. In Chapter 5, we will explain the basic idea to extend our approach to handle link capacity constraints.
- For clarity of presentation, we discuss the provisioning of one specific delay class with its given delay limit and its associated ingress and egress bandwidths for each VPN endpoint.
- The delay limit is identical for all the VPN endpoints of a given class and provisioning a VPN network can be viewed as provisioning each of the delay classes separately.

3.1.2. Statement of the Problem

Virtual Private Networks (VPNs) are becoming an increasingly important source of revenue for Internet Service Providers (ISPs). The aim is to provide the VPN endpoints with a service comparable to a dedicated private network established with leased lines.

In this work, we address the problem of resource allocation in VPN hose model with QoS guarantees while minimizing the total provisioning cost. Our simulation results indicate that our proposed algorithms reduce bandwidth requirements as compared to previously proposed algorithms [8, 23]. Our main objective is to find a near optimal tree supporting the maximum allowable end-to-end delay limit while trying to minimize the total provisioning cost. The problem can be formulated as follows:

Optimal Bandwidth and Delay-constrained Shared Tree Problem (*OBDSTP*): Given a set of VPN endpoints P with their associated ingress and egress bandwidths and the maximum allowable end-to-end delay, compute a shared tree T connecting all the VPN endpoints that satisfies the delay limit in which the total provisioning cost is the minimum.

THEOREM 1: OBDSTP is NP-hard.

Proof: We prove the above theorem by showing that the OBDSTP is at least as difficult as the problem of computing a Steiner tree connecting the VPN endpoints. This is because by assuming there is no delay limit, OBDSTP becomes the problem of computing a shared VPN tree with optimal provisioning cost without the delay limit, which is proved in [9] to be as difficult as computing a Steiner tree by assuming $B^{in}{}_p$ to be very small compared to $B^{out}{}_p$ for every VPN endpoint p. In this case the bandwidth reserved on each edge is determined by ingress bandwidths alone, and according to

Formula 2.1 is the constant $\sum_{l \in P} B^{in}_{l}$. Thus, the cost of the VPN tree is proportional to the number of edges in the tree, and as a consequence, the Steiner tree connecting the VPN endpoints has the smallest cost. However, since the Steiner tree computation problem for a set of VPN endpoints is NP-hard [36], it follows that the problem of computing the optimal VPN tree is also NP-hard and hence OBDSTP is NP-hard.

3.2. Solution Overview

We propose Optimal Bandwidth and Delay-constrained Shared Tree (OBDST) algorithm as a new heuristic approach to enhance the hose model to guarantee delay limits between endpoints while reducing the provisioning cost and execution time compared to previous works. Our OBDST algorithm takes into account the user preferences in meeting the most stringent delay limits versus decreasing the provisioning cost to find a near optimal solution for the OBDSTP.

Further, we introduce Hierarchical Iterative Spanning Tree (HIST) algorithm as a more efficient provisioning algorithm in finding the shared VPN tree compared with previous provisioning algorithms. Our HIST algorithm considers essentially the provisioning problem in hose model, i.e., how much capacity is needed in network links to provision the hose model. Our simulation results indicate that the HIST algorithm performs the best over a wide range of parameter values, and in most cases, reserves less bandwidth than previous works [9, 23, 32].

In the following sections, the detailed description of these algorithms is provided.

3.3. Methodology

3.3.1. OBDST Algorithm

Optimal Bandwidth and Delay-constrained Shared Tree Algorithm (OBDST) is our proposed heuristic solution to solve OBDSTP. This algorithm finds a shared tree connecting all the VPN endpoints satisfying the delay limit while trying to reduce the provisioning cost compared to the previous works.

Our methodology is to find two sets of shared trees: one optimizing the delay limit and the other reducing the bandwidth cost. From these sets, we select the best solution regarding both delay and bandwidth requirements using user specified preference parameters. Our approach consists of four phases:

In phase 1, we use a modified version of Minimum Diameter Steiner Tree (MDStT) based algorithm introduced in [8] to construct shared tree(s) connecting all VPN endpoints, with the objective of satisfying the given *maximum_allowable_delay* limit. In this step, similar to Least-Cost-Least-Delay (LCLD) approach in [8], we develop our MDStT algorithm based on the absolute center problem [19]. In our approach, if more than one local 1-center points satisfy the delay limit, in contrast to LCLD approach, all of them would be considered as candidates for center of the graph. For each center candidate, a Steiner tree [14] connecting it to all VPN endpoints using the minimum delay path would be constructed. The set of constructed trees is called Optimal Delay-constrained Shared Tree (ODST) set. All the shared trees in this set satisfy the delay limit but might not minimize the provisioning cost.

In phase 2, the total provisioning cost of each tree in ODST set, using Formula 2.1 from Chapter 2, is computed. The maximum provisioning cost for trees in ODST set is chosen as the bandwidth threshold to be used in the next phase.

In phase 3, we use our Hierarchical Iterative Spanning Tree (HIST) algorithm, explained in detail in Section 3.3.2, to construct shared tree(s) connecting all the VPN endpoints with the cost less than the bandwidth threshold. The set of constructed trees in this phase is called Optimal Bandwidth-constrained Shared Tree set (OBST set). This means that the shared trees in OBST set have total provisioning cost less than the bandwidth threshold.

We originally used a modified version of *AsymT* algorithm [23] in this step. The modification involved saving all the trees satisfying the bandwidth threshold requirement instead of finding only one tree with the smallest cost. Recalling from Chapter 2, *AsymT* is the best known approximation algorithm to find a tree T while the ingress and egress bandwidths are asymmetric. However as we were looking for an efficient solution, we further replaced it by our HIST algorithm which provides a better solution in terms of time complexity and provisioning cost. The simulation results studying the performance of HIST algorithm and comparisons with the *AsymT* algorithm are provided in detail in Chapter 4.

In phase 4, a ranking scheme is introduced to rank the trees in ODST and OBST sets. The tree(s) with smallest rank would be the best candidate for OBDSTP. Ranking the trees is done according to user specified bandwidth/delay preference and maximum allowable end-to-end delay of the particular service class. We formulate the following scheme for choosing from the above sets of trees the ones that will be closest to satisfy

user's bandwidth/delay preferences. For each shared tree T that belongs to OBST or ODST sets, the following value will be calculated:

$$\forall T \in ODST \vee OBST,$$

$$Rank(T) = delay\_preference \times \frac{(delay\_diameter)_T}{maximum\_allowable\_delay} + bandwidth\_preference \times \frac{(bandwidth\_cost)_T}{bandwidth\_threshold} \quad (3.1)$$

In the above formula, the *maximum_allowable_delay* is defined as the maximum allowable end-to-end delay dependent on the class-of-service. The $(delay\_diameter)_T$ is defined as maximum end-to-end delay between VPN endpoints of each tree T. The $(bandwidth\_cost)_T$ is defined as sum of provisioning costs over the links of T using Formula 2.1. As explained earlier, the bandwidth threshold is the maximum bandwidth cost for trees in ODST set. The delay preference and bandwidth preference parameters are set by the user and are dependent on the traffic characteristics. The lowest ranked trees are candidates for providing near optimal solutions.

This approach provides needed flexibility with respect to user's preferences in choosing delay versus bandwidth requirement. For example, VoIP applications may use larger delay preference while a guaranteed data service application may use larger bandwidth preference.

3.3.1.1. Example

As an example consider the network N_1 depicted in Figure 3.1. It contains a network with 6 nodes. Nodes 0, 1, 2, 3 are VPN endpoints with ingress/egress bandwidth equals to 5/5, 6/6, 7/7, 8/8 Mbps, respectively. The numbers on each edge indicate the link's delay

in milliseconds. Assume that the maximum allowable end-to-end delay for a particular application is 58 ms.

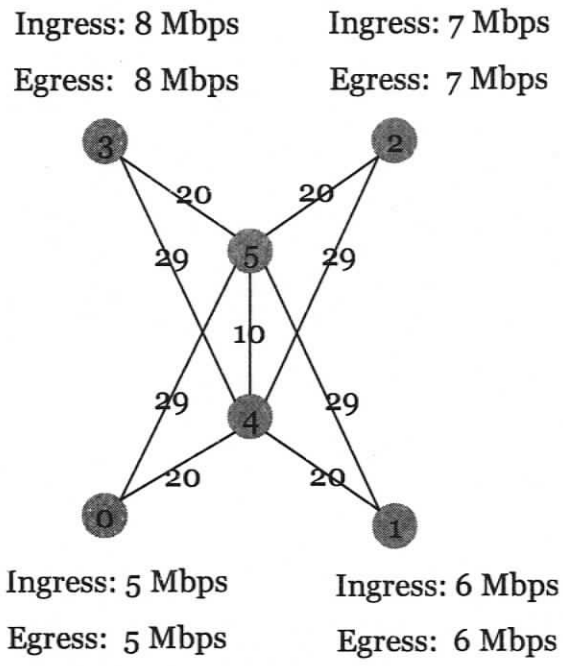


Figure 3.1: Sample network N_1

The result of phase 1 of OBDST algorithm is illustrated in Figure 3.2 in which the two shared trees (a) and (b) belong to the ODST set and satisfy the maximum allowable end-to-end delay limit. Then in phase 2, we compute the bandwidth cost of each tree and the bandwidth threshold is set as the maximum bandwidth cost over the trees in ODST. In this case the maximum bandwidth cost is the cost of the tree in Figure 3.2 (a) which is 74 Mbps.

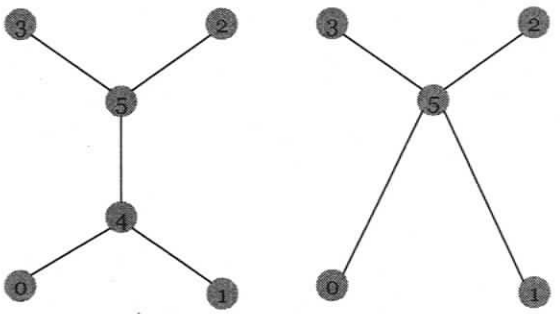


Figure 3.2: ODST set, the result of performing phase 1

Now, in phase 3, we execute our HIST algorithm, explained in detail in Section 3.3.2, on the original network N_1 to find all the shared trees with total bandwidth cost less than the bandwidth threshold computed in phase 2. The HIST algorithm finds trees depicted in Figure 3.3 and they will be added as members of OBST set.

Note that for the sake of clarity we have not shown the homogeneous trees in Figure 3.2 and Figure 3.3.

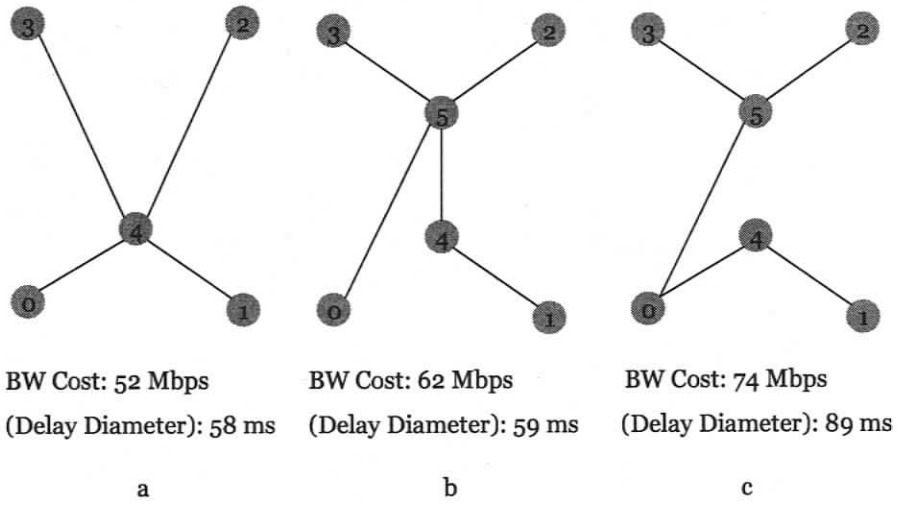


Figure 3.3: OBST set, the result of performing phase 3

In phase 4, the ranking is performed for each tree in ODST and OBST sets using Formula 3.1. Assume that the bandwidth preference and delay preference are set equal to one. The ranking value for each tree is as following:

Rank of tree in Figure 3.2 (a) = 74/74 + 50/58 = 1.86

Rank of tree in Figure 3.2 (b) = 52/74 + 58/58 = 1.70

Rank of tree in Figure 3.3 (a) = 52/74 + 58/58 = 1.70

Rank of tree in Figure 3.3 (b) = 62/74 + 59/58 = 1.85

Rank of tree in Figure 3.3 (c) = 74/74 + 89/58 = 2.53

For this example, both trees in Figure 3.2 (b) and Figure 3.3 (a) have the same minimum rank. Therefore, one can choose either one of them. However, executing the LCLD algorithm on this network would result in selection of tree depicted in Figure 3.2 (a) that has a considerably larger bandwidth cost.

3.3.2. Hierarchical Iterative Spanning Tree Algorithm

In this section, we describe our proposed Hierarchical Iterative Spanning Tree (HIST) heuristic algorithm to compute a near-optimal VPN tree; that is, the tree for which the amount of total bandwidth reserved on its edges is near-optimal. The HIST algorithm is a novel hierarchical approach to construct shared trees for the general VPN tree computation problem where ingress and egress bandwidths of VPN endpoints are arbitrary. The problem can be formulated as follows:

Optimal Bandwidth-constrained Shared Tree Problem (OBSTP): Given a set of VPN endpoints P and ingress egress bandwidths for each VPN endpoint, find a shared tree T

connecting VPN endpoints for which the total bandwidth reserved on edges of T is minimum.

As stated in Chapter 2, it is proved in [9] that OBSTP is NP-hard. In this section we explain our HIST algorithm as a heuristic approach to find a near-optimal solution for the OBSTP. Our simulation results with synthetic network graphs as well as real Tier-1 ISPs indicate that the VPN trees constructed by our proposed algorithm require less bandwidth reservation compared to *AsymT* algorithm [23]. Furthermore, we implemented and executed these algorithms on the same hardware platform and the HIST algorithm's execution time is measured to be far less than that of *AsymT* algorithm. The simulation results will be discussed in more detail in Chapter 4. In the following section, the basic idea behind our hierarchical approach will be explained.

3.3.2.1. Network Hierarchy Overview

In our approach, we considered a network with two levels of hierarchy: the core of the network and the edge of the network. VPN endpoints are located in the edge network and are connected to the routers in the core network. The edge network, representing VPN endpoints for a particular customer is essentially different branches of that VPN.

Our algorithm consists of two steps: step one is executed on the edge network to find a possible minimum cost tree connecting all the VPN endpoints without considering any intermediate routers in between. The result of this step is independent of the underlying network topology and is only dependent on the VPN endpoints' ingress and egress bandwidths. In step two we extend the result of step one to the core network and connect

the VPN endpoints by intermediate routers in a way to reduce the total provisioning cost. In the following sections, these two steps are explained in detail.

3.3.2.2. Step 1: ITERATIVE_SPANNING_TREE Procedure

As described in Chapter 2, a VPN network is modeled as a graph $G = (V, E)$ where V is the set of nodes and E is the set of bidirectional links connecting the nodes. The VPN specification in the hose model includes a subset of endpoints $P \subseteq V$ corresponding to the VPN endpoints and for each VPN endpoint $i \in P$, its associated ingress and egress bandwidths B^{in}_i and B^{out}_i, respectively.

The idea of step one is to assume that all VPN endpoints are connected to each other as vertices of a graph G'. The graph G', which is constructed iteratively in this step, can be considered as a virtual topology in which VPN endpoints are connected by virtual links. Thus in this step, we try to find minimum cost shared tree $T_{G'}$ connecting the vertices in graph G' (the VPN endpoints). Later, in the second step, we will replace each virtual edge (u, v) in $T_{G'}$ by the appropriate physical path between VPN endpoints u and v trying to keep the provisioning cost minimum.

Let us assume that the virtual topology $G' = (V', E')$ is a $K_{|P|}$ complete graph where $V' = P$ (set of VPN endpoints) and $E' = \{(u, v) \mid u, v \in P \text{ and } u \neq v\}$ (each pair of vertices is connected by an edge). We will relax this assumption in the next paragraph. The aim is to find a spanning tree $T_{G'}$ connecting all the vertices in G' with minimum cost. As the number of VPN endpoints in a network is mostly less than 10 percent of total number of nodes, one may suggest that $T_{G'}$ can be found by constructing all the spanning trees of

graph G' and finding the one with minimum cost. However, since the number of spanning trees for a complete graph K_n with n nodes is n^{n-2}, this approach is not scalable in terms of increasing the number of VPN endpoints.

Thus, we introduced an iterative approach to build the graph G' to overcome this problem. Figure 3.4 contains the ITERATIVE_SPANNING_TREE procedure which builds graph G' and outputs $T_{G'}$.

```
ITERATIVE_SPANNING_TREE (P)
1. T_G' = ø, G' = ø
2. for each vertex v ∈ P
3.     G' ← T_G'
4.     add new vertex v to G'
5.     add an edge from v to all other nodes in G'
6.     T_G' ← MODIFIED_SHIOURA(G')
7. return (T_G')
```

Figure 3.4: ITERATIVE_SPANNING_TREE procedure

The input of this procedure is the set of VPN endpoints P and the output is $T_{G'}$ that is a tree connecting VPN endpoints by virtual links. Since only the ingress and egress bandwidths of the VPN endpoints contribute to the shared tree's cost, the ITERATIVE_SPANNING_TREE procedure only iterates on the VPN endpoints while the previous works, *AsymT* and primal-dual algorithms introduced in [9, 23, 32], iterate over all the nodes of the graph. As the number of VPN endpoints is normally 10 percent of the

total number of nodes, this will reduce the execution time of our algorithm compared to previous works.

Without loss of generality, assume that the VPN endpoints are indexed as $p_1, p_2, \ldots, p_{|P|}$. The procedure starts with empty G' and $T_{G'}$ topologies. At iteration k, there is a tree $T_{G'}$ with k vertices connecting k VPN endpoints. At iteration $k + 1$, the $(k + 1)^{th}$ VPN node will eventually join the tree by adding node p_{k+1} to G' and also k edges from p_{k+1} to nodes $p_1, p_2, \ldots, p_k$ in G'. To find the spanning tree $T_{G'}$ in G', we use a modification of the algorithm proposed by Shioura et al. in [12], recognized as the best algorithm in terms of the time complexity and memory requirements to compute all the spanning trees of a given graph. The Shioura et al.'s algorithm is explained in detail in the Appendix. In the following section we provide a short review of their work along with our modification to the algorithm.

3.3.2.2.1. MODIFIED_SHIOURA Procedure

As explained in the Appendix, Shioura et al.'s algorithm enumerates all the spanning trees of a graph. This is done by first building a depth-first spanning tree T^0 and replacing some of its edges with appropriate substitute edges to build a new spanning tree. The former tree is called the parent tree, T^p, and the latter tree is called the child tree, T^c. For every newly built spanning tree the same procedure will apply to find all of its children. In this section, we provide details about our modifications to this algorithm.

Figures 3.5 and 3.6 contain the MODIFIED_SHIOURA and FIND_CHILDREN algorithms based on Shioura et al.'s all-spanning-trees and find-children procedures provided in Figure 7.3 in the Appendix. The input of MODIFIED_SHIOURA procedure is graph G' and

the output is *minTree* which is a spanning tree in G' with minimum provisioning cost over the enumerated spanning trees. In MODIFIED_SHIOURA procedure, we first find a depth-first spanning tree T^0 in G' and set the *minTree* equal to it. Further, we call the FIND_CHILDREN procedure to enumerate the spanning trees in G' and return the *minTree* which is the tree with minimum cost over the enumerated spanning trees.

MODIFIED_SHIOURA(G')

1. $n \leftarrow$ number of nodes in G'
2. **if** $n \leq 2$ **then return** G'
3. $minTree \leftarrow \emptyset$
4. $T^0 \leftarrow$ A depth-first spanning tree in G'
5. $minTree \leftarrow T^0$
6. $minTree \leftarrow$ FIND_CHILDREN(T^0, $n-1$, $minTree$)
7. **return** ($minTree$)

Figure 3.5: MODIFIED_SHIOURA procedure

Similar to find-children procedure in [12], calling FIND_CHILDREN procedure with arguments T^p, k, and *minTree* results in finding children of tree T^p not containing an edge e_k and saving the child with minimum cost in *minTree*. Whenever a child T^c is found, FIND_CHILDREN procedure recursively calls itself to find children of T^c. Further, it recursively calls itself again to find all children of T^p not containing edge e_{k-1}.

Our modifications to Shioura et al.'s algorithm includes adding lines 4-6 to Figure 3.6 in order to find *minTree* as the tree with minimum cost over the enumerated spanning trees in graph G'. Lines 4 and 5 keep track of the tree with minimum cost and line 6 is a pruning scheme in which we find the children of a tree provided that the tree diameter, which is the longest shortest path between tree endpoints based on the number of hops, is less than its parent's diameter. As shown in Table 4.2 in Chapter 4, our pruning scheme helps in decreasing the number of enumerated spanning trees and hence the execution time, while it keeps the results close to the case without using the pruning scheme. Note that this pruning scheme can be omitted for graphs with small number of VPN endpoints since the number of spanning trees of graph G' will not be very large.

```
FIND_CHILDREN(T^p, k, minTree)
1. if k ≤ 0 then return minTree
2. for each edge g ∈ Entr(T^p, e_k) as defined in [12]
3.         new tree T^c = Replace g with e_k in T^p
4.         if cost of T^c < cost of minTree
5.                 minTree ← T^c
6.         if diameter of T^c < diameter of T^p
7.                 FIND_CHILDREN(T^c, k − 1, minTree)
8. FIND_CHILDREN(T^p, k − 1, minTree)
```

Figure 3.6: FIND_CHILDREN procedure

3.3.2.3. Step 2: HIERARCHICAL_EXTENSION Procedure

In previous sections we explained the first step of HIST algorithm in which we assumed that VPN endpoints are vertices of a virtual topology G' and we found a spanning tree $T_{G'}$ connecting VPN endpoints. In the second step, we will expand each virtual edge (u, v) in $T_{G'}$ to the physical path between VPN endpoints u and v in the original network G.

Figure 3.7 contains the HIERARCHICAL_EXTENSION procedure. The input to this procedure is graph G and tree $T_{G'}$ which is the output of the last iteration of ITERATIVE_SPANNING_TREE procedure. The main goal of this procedure is to extend $T_{G'}$ to the core of the network to contain the intermediate routers.

At the beginning of the procedure, the final shared tree connecting all the VPN endpoints in the network, *finalTree*, is empty and all edges in the network have weights equal to one. These weights will be used by the Dijkstra's algorithm [13]. For each edge (u, v) in $T_{G'}$, if there is no path between u and v in the *finalTree* already, we use Dijkstra's algorithm to find the shortest path between u and v in the graph G. The new edges will be added to the *finalTree*. Moreover, to increase the link sharing probability, we set the weights of all edges in G that were added to *finalTree* to zero. Thus, the edges that are already in *finalTree* have less weight and hence higher probability of being selected in Dijkstra's algorithm over other edges. This is done to increase the probability of using the current edges in *finalTree* which increases the probability of having fewer edges in the *finalTree* and reducing the total provisioning cost. Finally, when all the VPN endpoints

are connected to each other, the resulting *finalTree* is the shared tree connecting all the VPN endpoints.

```
HIERARCHICAL_EXTENSION(T_G', G)
1. finalTree = ø
2. for each edge e ∈ G
3.     weight(e) ←1
4. for each edge (u, v) ∈ T_G'
5.     if (there is no path between u and v in finalTree)
6.         path_uv ← shortest path between u and v in G based on edge weights
7.         for each edge g ∈ path_uv and g ∉ finalTree
8.             add g to finalTree
9.             weight(g) ← 0
10. return (finalTree)
```

Figure 3.7: HIERARCHICAL_EXTENSION procedure

3.3.2.4. Example

As an example, consider the network N_2 in Figure 3.8. The four VPN endpoints 1, 2, 3, and 4 have ingress/egress bandwidth requirements of 3/12, 12/15, 5/8, 9/4 units respectively. This network has 48 spanning trees in total, some of which are shown in Figure 3.9. Each spanning tree is a possible candidate for the hose model. However, as

the number of nodes in the network grows, the number of spanning trees grows exponentially and investigating all the spanning trees to find the hose tree would not be feasible.

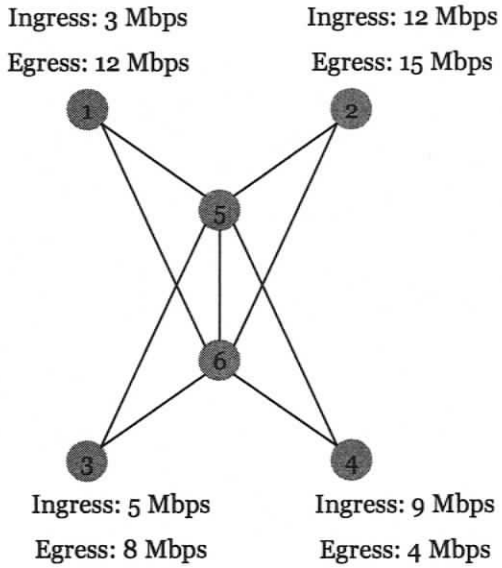


Figure 3.8: Sample network N_2

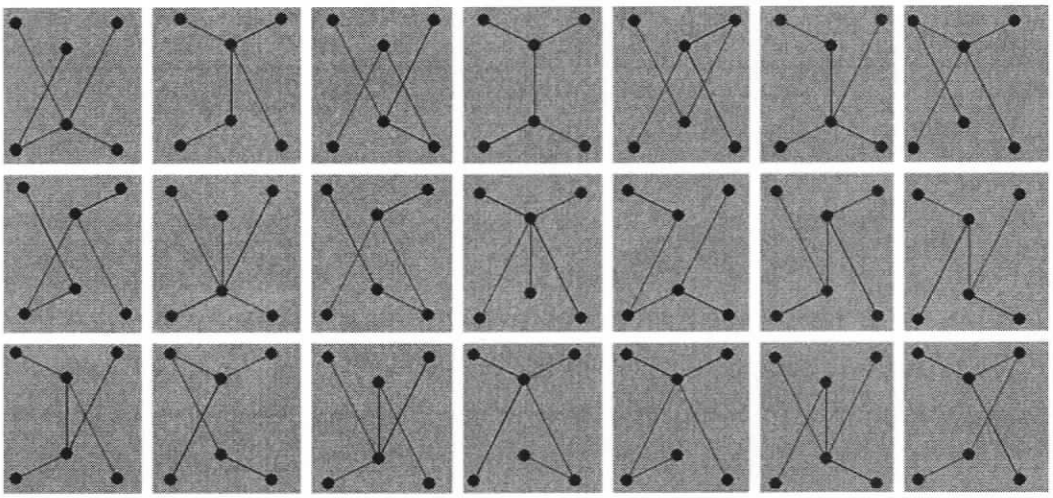


Figure 3.9: Some spanning trees of N_2

Figure 3.10 depicts the steps of performing the ITERATIVE_SPANNING_TREE procedure. Figures 3.10(a), 3.10(c), 3.10(e), and 3.10(g) illustrate the virtual topologies G' and Figures 3.10(b), 3.10(d), 3.10(f), and 3.10(h) depict the trees $T_{G'}$ in iterations 1, 2, 3, and 4, respectively.

Figures 3.10(a) and 3.10(b) show the first iteration of ITERATIVE_SPANNING_TREE procedure while there is only one node in the virtual topology. In Figure 3.10(c) node 2 is added to G' during the second iteration of the procedure. The only spanning tree connecting nodes 1 and 2 is shown in Figure 3.10(d). Furthermore, Figures 3.10(e) and 3.10(g) show the result of adding nodes 3 and 4 to G', respectively. Figures 3.10(f) and 3.10(h) depict the result of MODIFIED_SHIOURA procedure to find a spanning tree in Figures 3.10(e) and 3.10(g). The final result of ITERATIVE_SPANNING_TREE procedure is the tree $T_{G'}$ depicted in Figure 3.10(h) which is a virtual tree to connect the VPN endpoints in network N_2 according to their ingress and egress bandwidths. Further, $T_{G'}$ will be the input of HIERARCHICAL_EXTENSION procedure in which each edge in $T_{G'}$ will be replaced by a path in the network graph N_2.

Note that the total number of spanning trees for the graph in Figure 3.10(e) is three and total number of spanning trees for the graph in Figure 3.10(g) is eight. Thus, a total of 11 spanning trees have to be constructed in the first step of HIST algorithm, compared to 48 trees which is the total number of spanning trees of graph N_2.

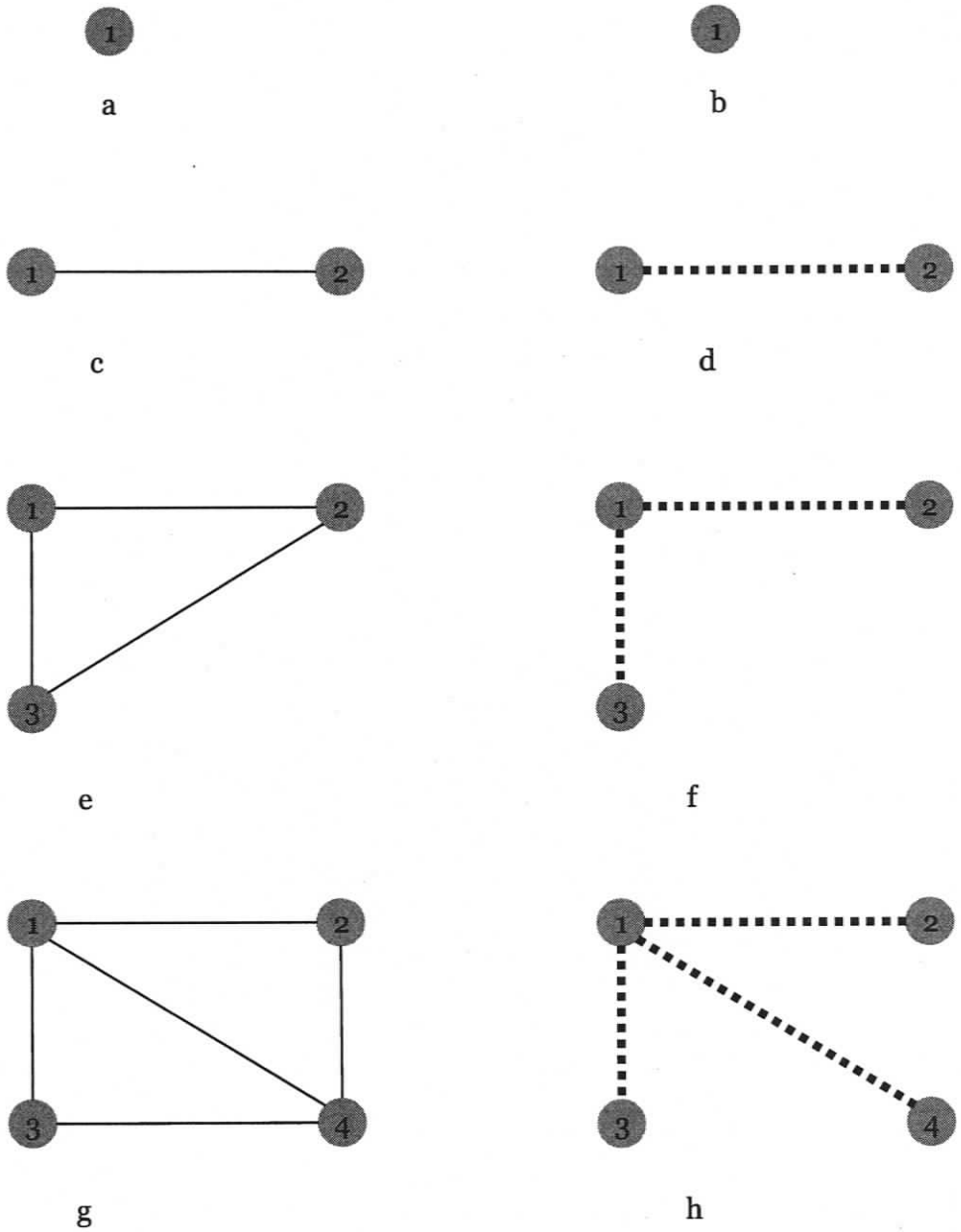


Figure 3.10: Steps of performing ITERATIVE_SPANNING_TREE procedure on N_2

The result of performing the HIERARCHICAL_EXTENSION procedure is depicted in Figure 3.11. Figure 3.11(a) shows the original network graph with each link having weight equal to 1. The procedure starts by selecting an edge from its input $T_{G'}$. Let us

assume that the edge (1, 3) is the first edge selected from $T_{G'}$. In Figure 3.11(b) the bold links show a shortest path between nodes (1, 3) considering the link weights. These edges will be added to the *finalTree* and their weights will be set to 0. Figures 3.11(c) and 3.11(d) show the result of finding the shortest path between nodes (1, 2) and nodes (1, 4), respectively. The *finalTree* is shown in Figure 3.11(e) with required bandwidth on each link and total provisioning cost of 68 bandwidth units.

To compare our result with the optimum tree, we calculated the cost of all the spanning trees of N_2 (some of which are depicted in Figure 3.9) and observed that the result of HIST algorithm is the optimum solution in our example. As mentioned earlier, although the total number of spanning trees of the network is 48, by using HIST algorithm our hierarchical approach builds only 11 spanning trees to find the tree with minimum provisioning cost.

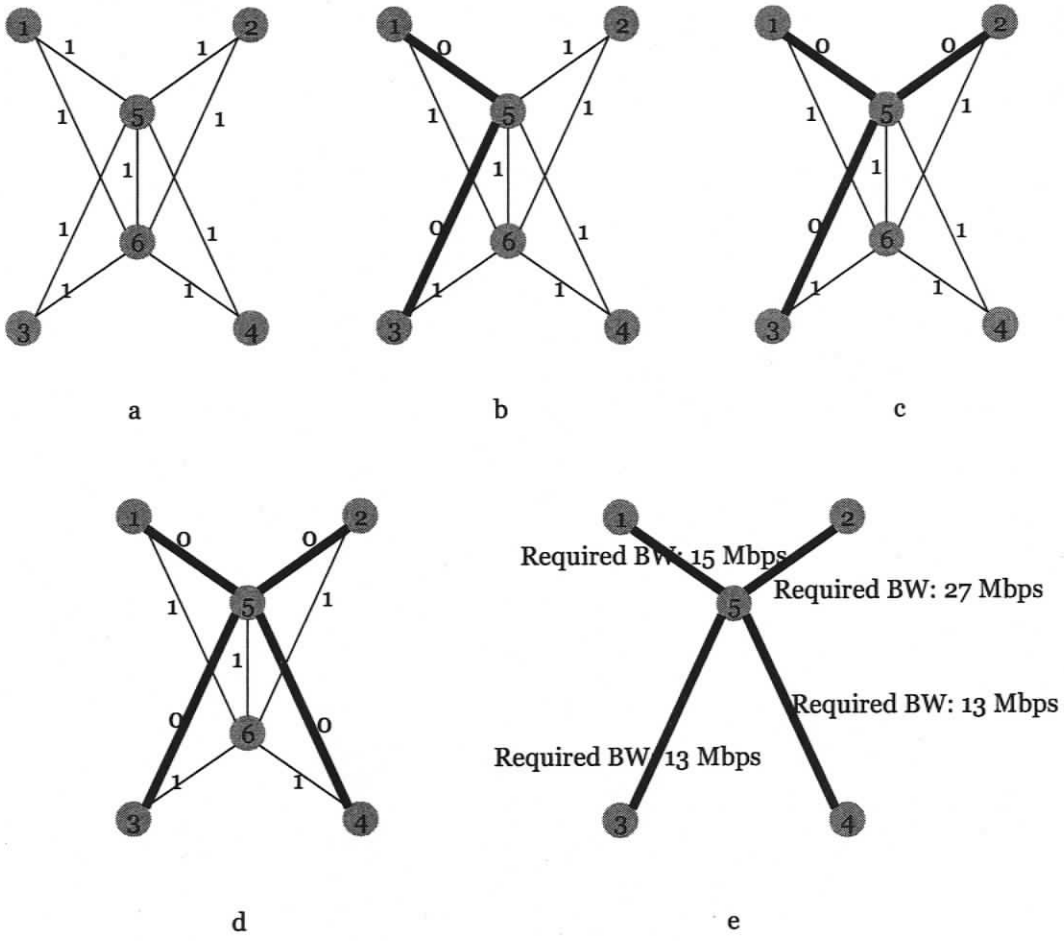


Figure 3.11: Steps of performing HIERARCHICAL_EXTENSION procedure on N_2

3.3.2.5. Embedding the HIST Algorithm in OBDST Algorithm

To be able to perform the HIST algorithm on phase 3 of our OBDST algorithm, explained in Section 3.3.1, we added another step to the final iteration of procedure ITERATIVE_SPANNING_TREE in Figure 3.4. The aim is to select all trees $T_{G'}$ with cost less

than the bandwidth threshold obtained in phase 2 of OBDST algorithm. As described earlier in Section 3.3.1, this set of trees is denoted as OBST set.

This is done by changing the *minTree* variable to a linked list of trees (OBST set) and by modifying MODIFIED_SHIOURA and FIND_CHILDREN procedures to add trees with cost less than the bandwidth threshold to this linked list. Note that these modifications are only required for the final iteration of ITERATIVE_SPANNING_TREE procedure, i.e., when number of nodes in G' is equal to number of VPN endpoints.

In the next section we will prove the correctness of OBDST and HIST algorithms.

3.4. Correctness Properties

3.4.1.1. Correctness Properties of OBDST Algorithm

Assumption 3.1: *The Optimal Delay-constrained Shared Tree (ODST) set is non-empty.*

ODST set is the set of Steiner trees in G with delay diameter less than the *maximum_allowable_delay*. Thus, ODST set is empty if and only if there is no Steiner tree with diameter less than the *maximum_allowable_delay*. To prevent this scenario, in our implementations, we set this value to be greater than the delay diameter of G, calculated based on the algorithms for the absolute center problem [10] explained in Section 2.3

Another alternative solution is to add tree(s) with diameter equal to delay diameter to ODST set. Although the delay diameter of these trees is greater than the *maximum_allowable_delay*, the difference might be tolerable by the user application.

This tolerance is specified by *delay_preference* parameter and will be in effect in our ranking scheme.

LEMMA 3.1: *Optimal Bandwidth-constrained Shared Tree (OBST) set can be empty.*

Proof: OBST set is the set of trees with provisioning cost less than *bandwidth_threshold*. Since the *bandwidth_threshold* is defined as the maximum provisioning cost of trees in ODST set, such tree exists. However, it is possible that the heuristic algorithms, such as our HIST algorithm or *AsymT* algorithm [23], used to find ODST set do not find any of these trees. Although this situation did not happen in our simulations, this case does not affect the correctness of our algorithm as the ranking scheme is not dependant on the size of OBST set.

LEMMA 3.2: *OBDST algorithm will return a non-empty tree.*

Proof: According to Assumption 3.1, the ODST set will have at least one element and thus the OBDST algorithm will always return a non empty tree.

The above lemmas prove the correctness of OBDST algorithm.

3.4.1.2. Correctness Properties of HIST Algorithm

LEMMA 3.3: *The "minTree" returned by MODIFIED_SHIOURA procedure in Figure 3.5 is a non-empty spanning tree of graph G'.*

Proof: Recall from Section 3.3.2.2 that graph $G' = (V', E')$ is a virtual topology in which $V' = P = \{p_1, \ldots, p_{|P|}\}$ (set of VPN endpoints) and E' is the set of virtual links. To prove the above lemma, we consider two cases:

1. The number of VPN endpoints in the network is equal or less than two: In this case since $|V'| = |P|$, the number of nodes of graph G' is one or two:

a. If $|V'| = 1$ then $V' = \{p_1\}$ and $E' = \{\emptyset\}$,

b. and if $|V'| = 2$ then $V' = \{p_1, p_2\}$ and $E' = \{(p_1, p_2)\}$,

In either of the above cases, MODIFIED_SHIOURA procedure will return graph G' since it is obvious that the only spanning tree in G' is G' itself.

2. The number of VPN endpoints is greater than two: In this case, in MODIFIED_SHIOURA procedure, the depth-first spanning tree T^0 will be set as *minTree* and the in FIND_CHILDREN procedure is called with *minTree* as its input. Since our modification to Shioura et al.'s algorithm did not change the method of building the spanning trees, it is guaranteed that the trees constructed by this procedure connects all the vertices of G'. Thus, the MODIFIED_SHIOURA procedure returns a non-empty spanning tree in G'.

LEMMA 3.4: *$T_{G'}$ returned by ITERATIVE_SPANNING_TREE procedure in Figure 3.4 is a spanning tree connecting all the VPN endpoints.*

Proof: Assume $P = \{p_1, \ldots, p_m\}$ is the set of VPN endpoints. To prove this lemma we use mathematical induction on m, number of VPN endpoints:

1. The basis, $m = 1$: This is the trivial case as there is only one node in G' and hence in $T_{G'}$.

2. The basis, $m = 2$: For this case, in the first iteration of ITERATIVE_SPANNING_TREE procedure, node p_1 is added to G'. In the second iteration, node p_2 is added to set of vertices of graph G' and the edge (p_1, p_2) is added to set of edges of G'. Further, calling the MODIFIED_SHIOURA procedure on G' will result in $T_{G'}$ having the same topology as G', since the number of nodes of G' is two. Thus in this case $T_{G'}$ connects all the VPN endpoints.

3. The inductive step: In this step we show that if the above lemma holds for $m = k$, then it holds for $m = k + 1$: Our induction hypothesis implies that with k VPN endpoints, $T_{G'}$ is a spanning tree connecting VPN endpoints $p_1, \ldots, p_k$. By adding node p_{k+1} to set of VPN endpoints and assuming that ingress and egress bandwidths of nodes $p_1, \ldots, p_k$ is kept the same, $T_{G'}$ will be the same in the first k iterations of ITERATIVE_SPANNING_TREE procedure. In the last iteration, node p_{k+1} will be added to graph G' in which there is a path between every VPN endpoints p_i and p_j, $1 \leq i, j \leq k$. By adding edges (p_{k+1}, p_i), $1 \leq i \leq k$ to G', there will also be a path between p_{k+1} and nodes p_i, $1 \leq i \leq k$ and hence there is a path between all vertices of G'. Further, we use the MODIFIED_SHIOURA procedure to find spanning tree $T_{G'}$ in G'. According to Lemma 3.3 the *minTree* returned by MODIFIED_SHIOURA procedure is a non-empty spanning tree in graph G' and since there is a path between every VPN endpoint in G', it is guaranteed that G' is a connected graph and the spanning tree returned by this procedure, connects all the VPN endpoints.

This proves the induction hypothesis and hence proves the Lemma.

LEMMA 3.5: *The "finalTree" returned by HIERARCHICAL_EXTENSION procedure in Figure 3.7 is loop free.*

Proof: Recall from Section 3.3.2.3 that *finalTree* is built using $T_{G'}$ by finding the shortest paths between VPN endpoints that are connected by an edge in $T_{G'}$. Assume that (p_i, p_j) is the first edge selected from $T_{G'}$. As there is no path between p_i and p_j, the set of edges in the shortest path between p_i and p_j in G will be added to *finalTree*. Lets call this set of edges as $Path(p_i, p_j)$. Further, the weight of all edges in $Path(p_i, p_j)$ will be set to zero. Without loss of generality, assume that edge (p_m, p_n) is the second edge selected

from $T_{G'}$. Moreover, assume that there is no path between p_m and p_n, and the shortest path between p_m and p_n is called $Path(p_m, p_n)$. As illustrated in Figures 3.12 (a) to 3.12 (c) these two paths can have three cases of relative relations:

Case a) $Path(p_i, p_j) \cap Path(p_m, p_n) = \emptyset$ as illustrated in Figure 3.12 (a).

Case b) $Path(p_i, p_j) \cap Path(p_m, p_n) = u$ where u is a vertex in the graph, as illustrated in Figure 3.12 (b).

Case c) $Path(p_i, p_j) \cap Path(p_m, p_n) = U$, where U is the set of common vertices and $|U| > 1$. The case where $|U| = 2$ is illustrated in Figure 3.12 (c).

Both cases *a*) and *b*) are possible and as depicted in Figures 3.12 (a) and (b), no loop will be generated by adding edges in $Path(p_m, p_n)$ to the *finalTree*.

Case *c*) is only possible if the set of edges between vertices in $Path(p_i, p_j) \cap Path(p_m, p_n)$ coincide. This is because the weight of edges between vertices in U that belong to $Path(p_i, p_j)$ is zero but the weight of edges between vertices in U that belong to $Path(p_m, p_n)$ is one. For example in Figure 3.12 (c) the weight of edges between u and v that belong to $Path(p_i, p_j)$ is zero but the weight of edges between u and v in $Path(p_m, p_n)$ (depicted by dashed lines) is one. This implies that the only possible situation is when the set of edges between u and v coincide, illustrated in Figure 3.12 (d).

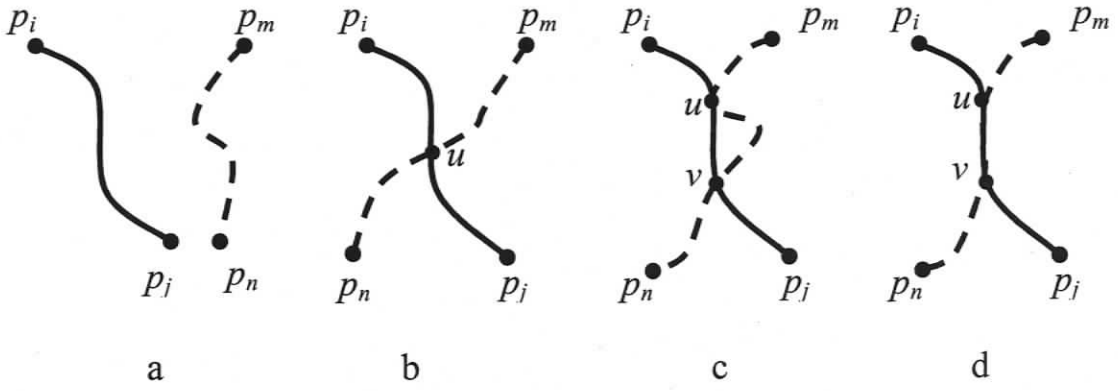


Figure 3.12: Relative relation between paths

This proves that no loop will be added to *finalTree* while replacing each edge of $T_{G'}$ with a path in G. Thus the *finalTree* will be loop-free.

LEMMA 3.6: *All VPN endpoints are connected in finalTree.*

Proof: According to Lemma 3.4, $T_{G'}$ connects all VPN endpoints. Thus there is a path between every VPN endpoint in $T_{G'}$. Let's call $Path^T(p_m, p_n)$ to be the path between p_m and p_n in $T_{G'}$ denoted by set of edges $e_j, e_{j+1}, \ldots, e_t \in T_{G'}$. In HIERARCHICAL_EXTENSION procedure every edge in $T_{G'}$ including edges $e_i = (p_k, p_l) \in Path^T(p_m, p_n)$ will be replaced by a path between p_k and p_l, thus there will be a path between p_m and p_n in *finalTree* by replacing each e_i with the path connecting e_i endpoints.

3.5. Time Complexity Analysis

In this section the time complexities of HIST and OBDST algorithms are analyzed. Since the OBDST algorithm uses the HIST algorithm, we provide the time complexity analysis of HIST algorithm first.

3.5.1. Time Complexity Analysis of HIST Algorithm

In this section we will show that the time complexity of HIST algorithm is $O(p^3 + pm + n\log n)$ where m is the number of edges, n is the number of nodes and p is the number of VPN endpoints in the network:

- Time complexity of step 1: The loop in line 2 of ITERATIVE_SPANNING_TREE procedure in Figure 3.4 performs p iterations, one for each VPN node. At line 3 of k^{th} iteration, $T_{G'}$ which is the result of the previous iteration, is copied to graph G'. Thus G' will have k-1 vertices and k-2 edges. Further, in line 4 the number of vertices of G' is increased by one and in line 5 the number of edges of G' is increased by k-1. Thus in line 6, the MODIFIED_SHIOURA procedure in Figure 3.5 is executed on a graph G' with k vertices and $2k$-3 edges. According to [12], the time complexity of MODIFIED_SHIOURA procedure is $O(N_k + k + 2k - 3)$, where N_k is the number of spanning trees that the algorithm iterates for a graph with k vertices. Thus, the total time complexity of ITERATIVE_SPANNING_TREE procedure is $O(\sum_{k=1}^{p}(N_k + 3k - 3)) = O(\sum_{k=1}^{p} N_k + p^2)$.

  As shown in Table 4.2, it is observed in our simulations that using our pruning scheme will keep N_k to be $O(k^2)$ and hence the time complexity of ITERATIVE_SPANNING_TREE procedure is $O(\sum_{k=1}^{p} k^2 + p^2) = O(p^3)$.

- Time complexity of step 2: It is clear that the loop in Line 2 of HIERARCHICAL_EXTENSION procedure (Figure 3.7) will be executed m times. Further, the loop in line 4 will be executed p-1 times since tree $T_{G'}$ has p-1 edges. As the worst case time complexity of Dijkstra's algorithm is O(m + $n\log n$) [13], the total time complexity of Figure 3.7 is $O(m + (p - 1)(m + n\log n))$ = $O(pm + pn\log n)$.

Thus our HIST algorithm has an overall time complexity of $O(p^3 + pm + pn\log n)$. Since the number of VPN endpoints (p) is usually less than 10 percent of the total number of nodes (n), the time complexity of HIST algorithm is far less than that of *AsymT*'s algorithm given as $O(nm^2p+mn^2p+n^3\log n))$ in [10].

3.5.2. Time Complexity Analysis of OBDST Algorithm

Our OBDST algorithm uses a modified version on LCLD algorithm in [8] based on MDStT algorithm to find trees with delay diameter less than the maximum allowable end-to-end delay. The LCLD algorithm has a time complexity equal to $O(mp+nplogp)$ where m is the number of edges, n is the number of nodes and p is the number of VPN endpoints in the network. As explained in Section 3.3.1, we implemented the MDStT algorithm based on the algorithms for the absolute center problem in [19]. The main idea is to identify a local 1-center for each edge in the graph and the global absolute center can be found by selecting the optimal one from the m local centers. In the worst case in OBDST algorithm, during phase 1, all trees constructed by setting each local center as root of the tree might have delays less than the maximum allowable end-to-end delay.

This will not change the time complexity of finding the global center in the MDStT algorithm but the complexity of tree construction will be affected. Thus the worst case time complexity will be $O(m(mp+nplogp))$ for phase 1.

The time complexity of constructing the trees in ODST set and finding the bandwidth threshold (phase 2) is $O(c)$ where c is the size of ODST set and $1 \leq c \leq m$.

In phase 3, as explained in Section 3.3.2.5 we use HIST algorithm to find trees with provisioning cost less than the bandwidth threshold. As explained in the previous section, time complexity of HIST algorithm is $O(p^3 + pm + pn\log n)$. To be able to use HIST algorithm in phase 3 of OBDST algorithm, we keep the trees with bandwidth costs less than the bandwidth threshold in a linked list for further ranking which does not change the order of time complexity of phase 3.

In phase 4, we rank the trees in ODST and OBDT sets. In the worst case, the maximum size of ODST set is the number of edges in the network (m) since the candidate center point can be on any edge of the graph. Recalling from Section 3.3.2.5, we add trees with costs less than the bandwidth threshold in the last iteration of Figure 3.4 to OBST set. Thus, the maximum size of OBST set is the maximum number of enumerated spanning trees (N_p) in the last iteration of Figure 3.4. As mentioned earlier, in our simulations, N_k has been observed to be $O(k^2)$, thus the maximum size of OBST set is $O(p^2)$. Since the ranking is done over all trees in OBST and ODST sets, the time complexity of ranking phase is $O(p^2 + m)$.

Chapter 4. Simulations

We have designed a number of simulation experiments to measure the performance of our proposed OBDST algorithm, described in Section 3.3.1, and our proposed HIST algorithm, described in Section 3.3.2.

The simulations are implemented in C++ and all simulations were performed on a dual processor Intel Pentium D CPU 3 GHz machine with 2 GB of RAM, running Microsoft Windows XP Professional.

4.1. Network Topologies

In our simulations, we used two sets of network topologies. The first set of topologies was selected from real Tier-1 ISP topologies available from Rocketfuel project [4]. For the second set we implemented a random network generator based on the work by Waxman [31].

4.1.1. Real Tier-1 ISP Topologies

Rocketfuel is an ISP topology mapping engine developed at University of Washington. In Rocketfuel project, the routing information is used to understand an ISP's topology

using "traceroutes" sourced from 800 vantage points hosted by nearly 300 traceroute web servers.

From the available data in Rocketfuel's project website [4], we used the "Backbone topologies annotated with inferred weights and link latencies" file which contains the topologies for six ISPs along with link weights and link latencies. The provided latency of a link, as used in our simulations, is estimated based on the geographic distance of link endpoints. Among all six provided topologies, we selected two dominant tier-1 ISP topologies as listed in Table 4.1.

Table 4.1: Rocketfuel ISP topologies used in our simulations

AS number	Name	Tier	No. of links	No. nodes
1239	Sprint(US)	1	168	52
7018	ATT(US)	1	296	115

4.1.2. Random Networks

We used Waxman model [31] to generate random networks. 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 simulations we placed the nodes on a 3000×2400 KM^2 plane, roughly the size of the USA. The probability function for two nodes to be connected by a link is: $P_e^{(u,v)} = \alpha \exp(-l(u, v) / L\beta)$ where L is the maximum distance between any two nodes in the network and $l(u, v)$ is the distance between nodes u and v. The parameter β controls the ratio of short links to long links,

while the parameter α controls the average node degree of the network. Large value of β increases the number of long links, and a large value of α results in a large average node degree. In the simulations, α and β were set at 2.2 and 0.15 respectively. These values were selected to obtain random networks with close resemblance to real networks. The same parameters are also used in [8].

Since we can easily control the size of the topologies, we use this model to study the effect of the network size. Like the Rocketfuel topologies and topologies used in LCLD algorithm's implementation in [8], the link delay values of the random networks were calculated according to their geographical distances.

4.2. Parameters

A subset of the nodes in each network is chosen randomly and uniformly as the VPN endpoints. The number of VPN endpoints was set to 10% of the total number of network nodes in the network unless explicitly specified. To model asymmetric endpoint bandwidths, an "asymmetry parameter" r is associated with each endpoint, representing the ratio between the ingress and egress bandwidths at that endpoint. This ratio was selected randomly from 1 to 256 for each VPN endpoint. The ingress bandwidth of an endpoint was uniformly chosen between 2 and 100 Mbps and the egress bandwidth was set to ingress bandwidth multiplied by r.

In the following section we provide the simulation results. Each simulation result given below is the average of 5 rounds of simulation run. We calculated 95 percent confidence intervals as $\hat{\theta} - t_{\alpha/2,f}\hat{\sigma}(\hat{\theta}) \le \theta \le \hat{\theta} + t_{\alpha/2,f}\hat{\sigma}(\hat{\theta})$ where $\hat{\theta}$ is the average value of

simulations runs, $\hat{\sigma}^2(\hat{\theta})$ is the standard deviation and $t_{\alpha/2,f}$ is the quantile of the t distribution where $f+1$ is the degree of freedom and is equal to the number of simulation runs[1]. The results show that the confidence intervals for provisioning costs are less than 0.5 Gbps and confidence intervals for delay diameters are less than 5 ms.

4.3. Results

4.3.1. OBDST Algorithm

As mentioned in Chapter 3, we implemented two versions of OBDST algorithm: One using *AsymT* algorithm [23] and one using HIST algorithm in phase 3.

Let γ be the ratio of *bandwidth_preference* to *delay_preference* parameters used in Formula 3.1. We define three different scenarios for our OBDST algorithm:

Scenario 1) bandwidth preference is equal to delay preference ($\gamma = 1$);

Scenario 2) bandwidth preference is greater than delay preference ($\gamma > 1$);

Scenario 3) bandwidth preference is smaller than delay preference ($\gamma < 1$).

The performance of different versions of OBDST algorithm and LCLD algorithm in different scenarios are compared in Figures 4.1 to 4.7. As mentioned in Chapter 2, LCLD algorithm is the proposed approach in [8] to enhance the hose model to support the delay limit. Figures 4.1, 4.2 and 4.3 compare provisioning cost of LCLD algorithm with OBDST using *AsymT* algorithm or OBDST using HIST algorithm for scenarios 1, 2, and 3, respectively. The results show that OBSDT using HIST algorithm requires less

[1] For 95% confidence interval, α=0.05. Therefore, $t_{0.025,4}$ is 2.78 according to Table A.5 in [26].

provisioning cost in most cases. Moreover, it can be observed from these figures that for each topology, the total provisioning cost of OBDST algorithm in scenario 2 is less than the total provisioning cost in scenarios 1 and 3 as the bandwidth preference is higher than delay preference in scenario 2.

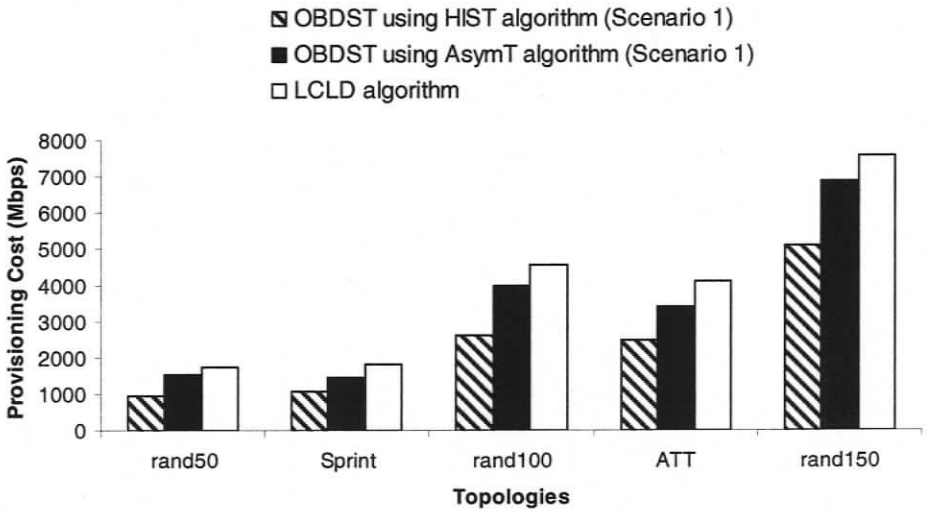


Figure 4.1: The provisioning cost comparison: Scenario 1 ($\gamma = 1$)

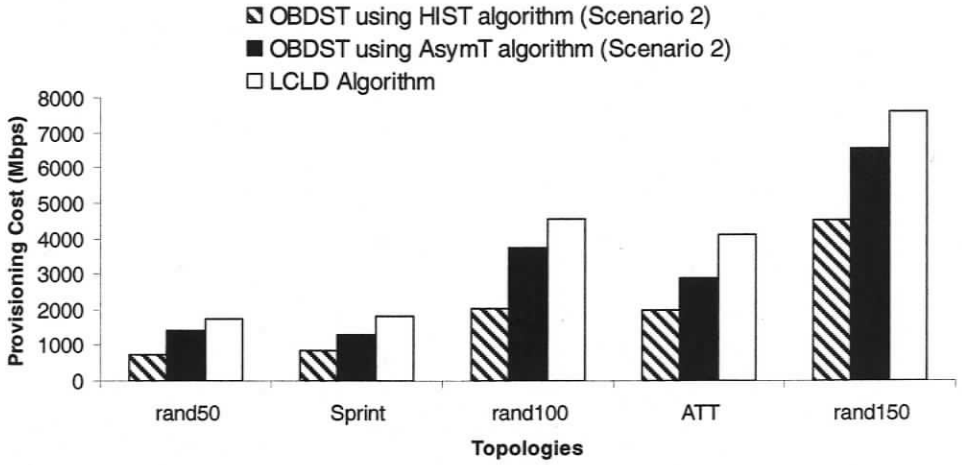


Figure 4.2: The provisioning cost comparison: Scenario 2 ($\gamma > 1$)

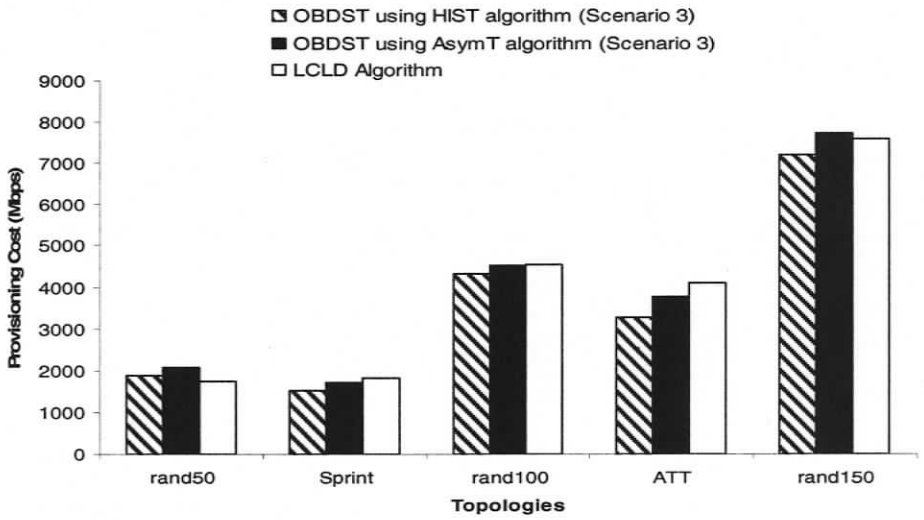


Figure 4.3: The provisioning cost comparison: Scenario 3 ($\gamma < 1$)

Figures 4.4, 4.5, and 4.6 compare the delay diameter of constructed shared trees connecting VPN endpoints. This value can be interpreted as the maximum allowable end-to-end delay that can be '*supported*' by each tree. The results show that using OBDST algorithm with delay preference greater than bandwidth preference (as in scenario 3) would result in smaller delay diameter than LCLD algorithm. Note that since the Sprint

network is more a linear network than the random networks generated by the Waxman model, there is an increase in the delay diameter for Sprint network in Figures 4.4 to 4.6.

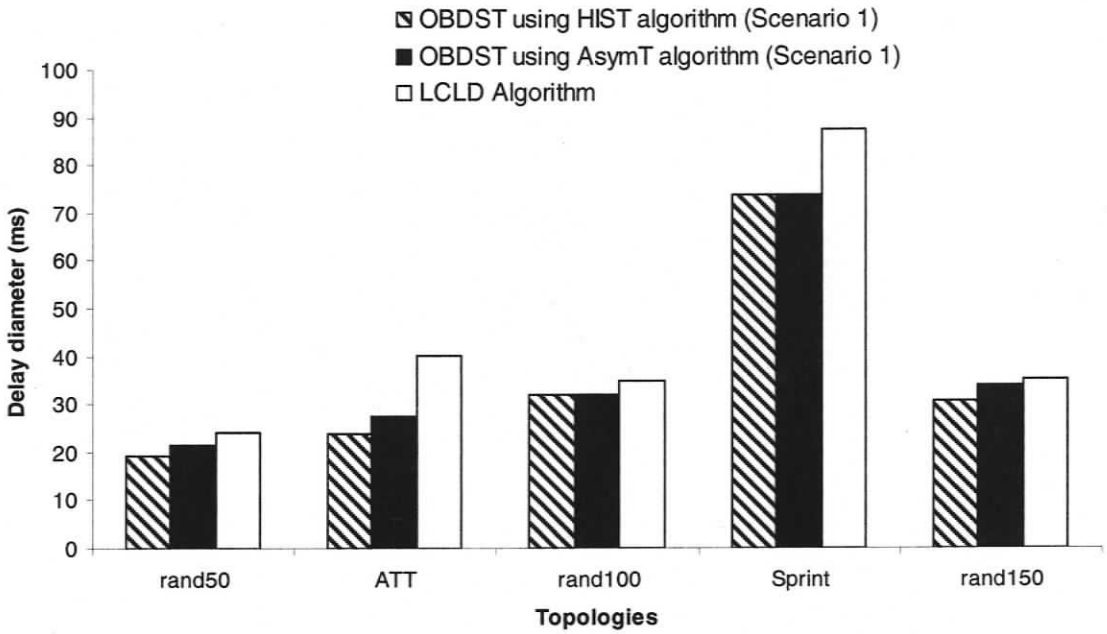


Figure 4.4: The delay diameter comparison: Scenario 1 ($\gamma = 1$)

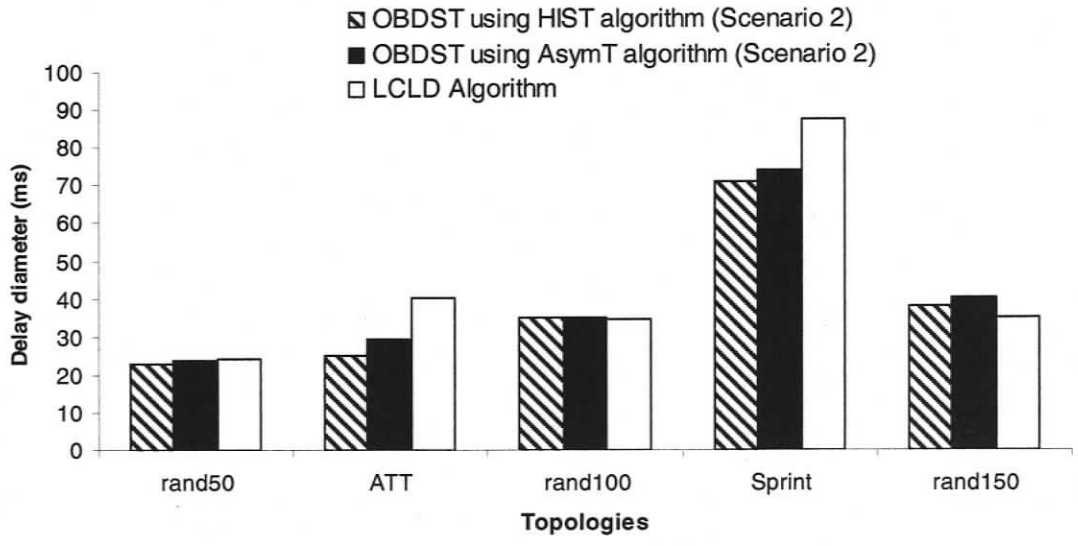


Figure 4.5: The delay diameter comparison: Scenario 2 ($\gamma > 1$)

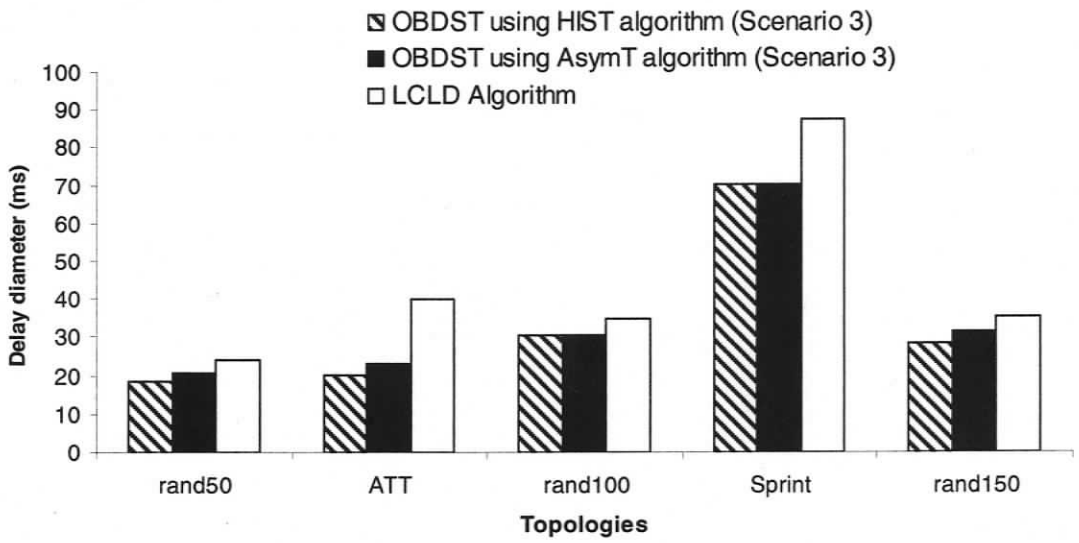


Figure 4.6: The delay diameter comparison: Scenario 3 ($\gamma < 1$)

Fig. 4.7 illustrates the effect of increasing the number of VPN endpoints from 15 to 45 on execution time for a 150 nodes network. The results show that OBDST using HIST algorithm has an execution time comparable to LCLD algorithm while OBDST using *AsymT* algorithm has an exceedingly large execution time.

Figure 4.8 shows the effect of increasing the network size on the execution time. As the results show, OBDST using *AsymT* algorithm has a relatively high execution time for large networks.

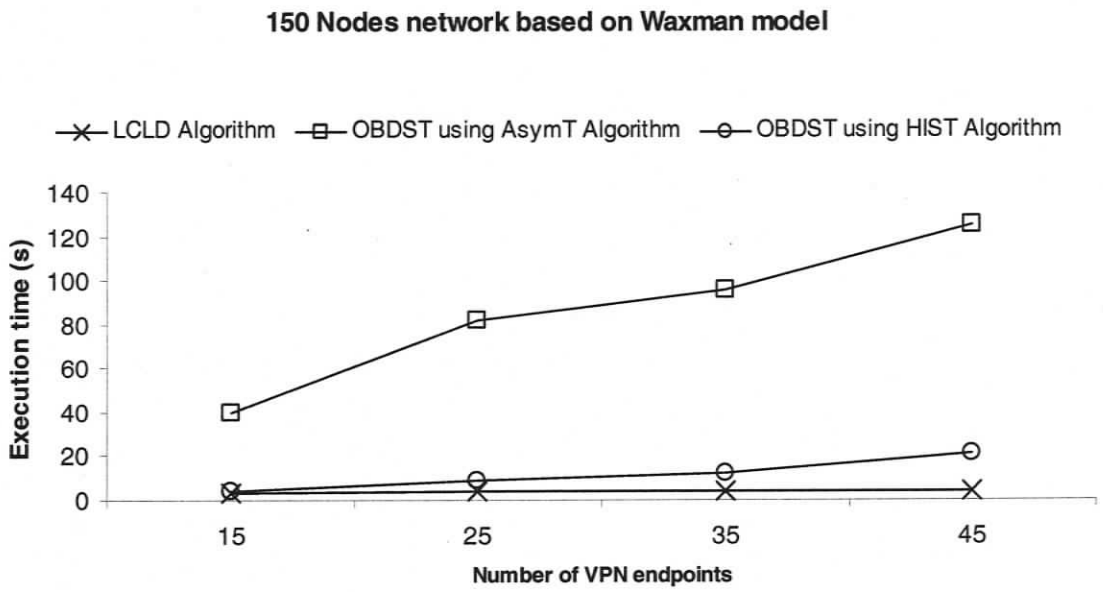


Figure 4.7: Effect of number of VPN endpoints on execution time

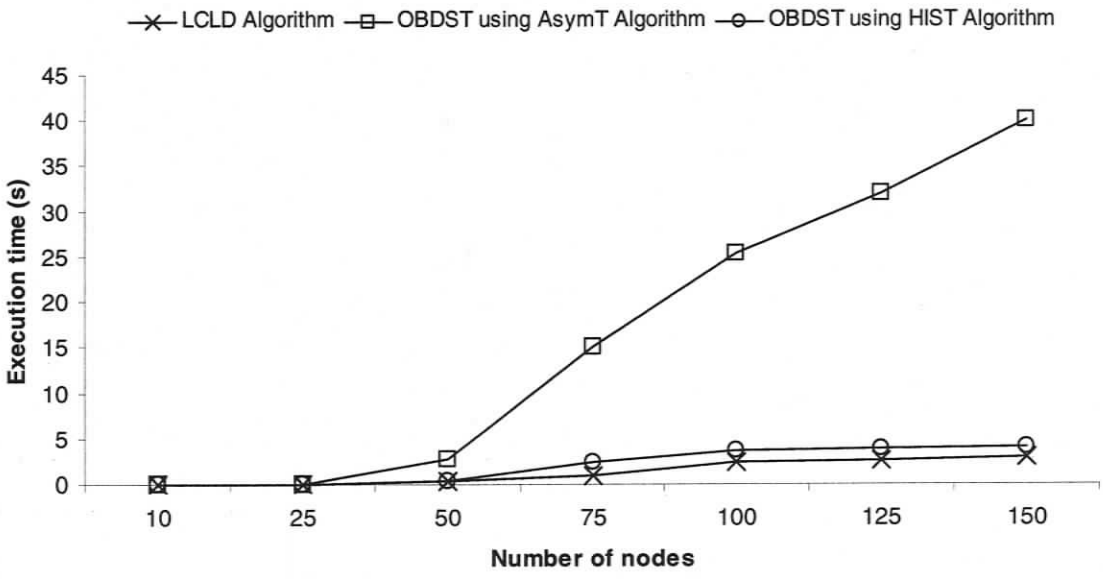


Figure 4.8: Effect of number of nodes on execution time

Figure 4.9 studies the effect of changing the bandwidth preference over delay preference in different scenarios for a 150 nodes network based on Waxman model. In scenario 1, γ (the ratio of bandwidth preference to delay preference) is 1, γ is 10 in scenario 2 and γ is 0.1 in scenario 3. As expected, based on the proposed ranking scheme, scenario 2 with γ equal to 10 finds the shared tree with smallest cost. Also Figure 4.10 illustrates the effect of changing γ on delay diameter of constructed shared trees. The results show that scenario 3 with γ equal to 10 supports the smallest delay limit since the delay preference is greater than bandwidth preference in this case.

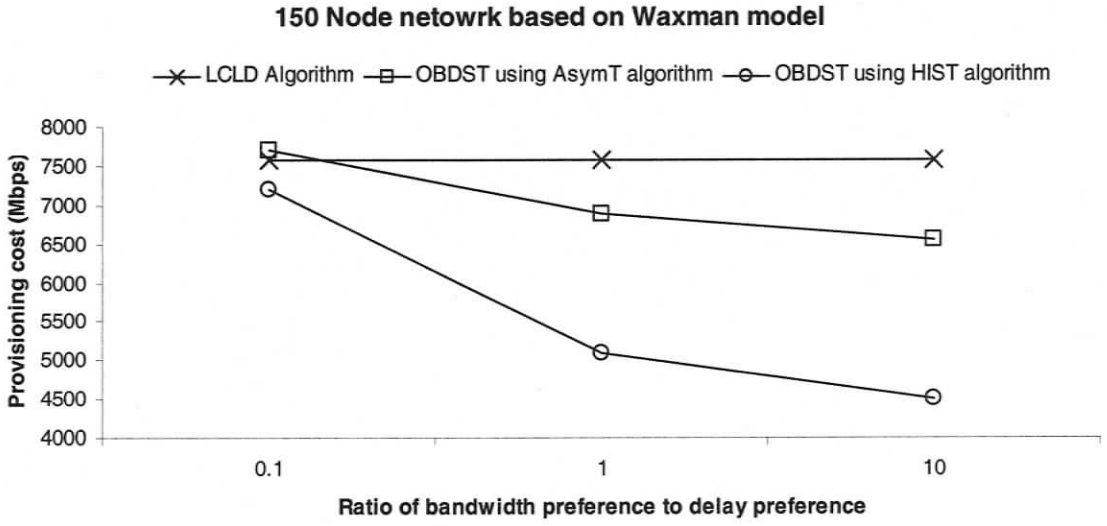


Figure 4.9: Effect of γ on provisioning cost

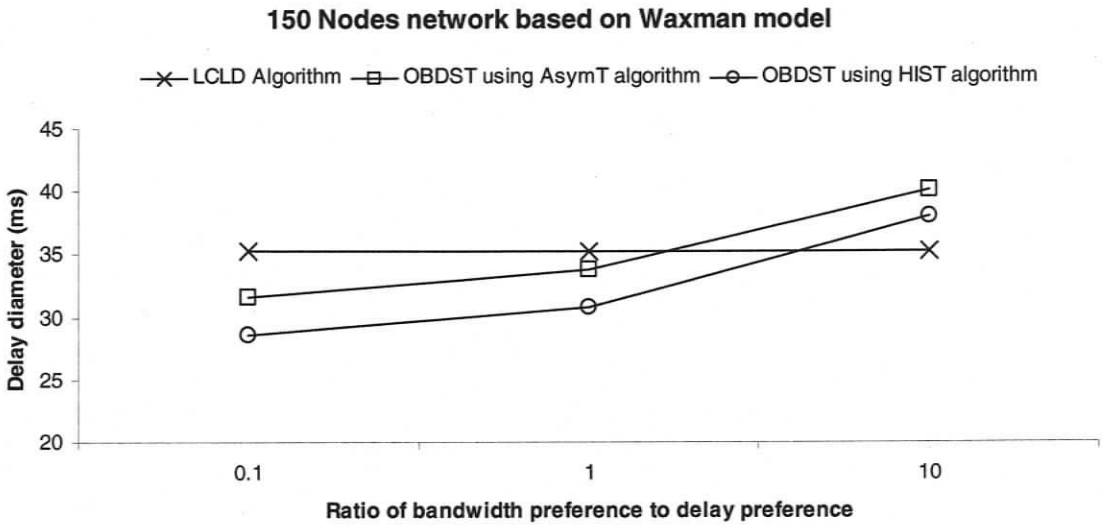


Figure 4.10: Effect of γ on delay diameter

4.3.2. HIST Algorithm

In this section, we compare the performance of HIST algorithm and *AsymT* algorithm [23] with the optimal solution. The optimal solution is found by constructing all the spanning trees of each network and finding the tree with minimum cost. Moreover, the effect of varying the network size and the number of VPN endpoints on performance is also investigated. The provisioning cost (the total bandwidth reserved on edges of the tree) is used as a performance metric for the HIST algorithm.

Figure 4.11 shows the required provisioning cost of HIST algorithm with and without pruning scheme as well as *AsymT* algorithm and optimal solution for some small random networks. The name of the random topologies indicates the number of nodes in the network, e.g., "rand15" is a random topology with 15 nodes. In this case, the number of

VPN endpoints in each network is fixed at 50 % of the total number of nodes. The dashed line shows the provisioning cost of the optimal reservation.

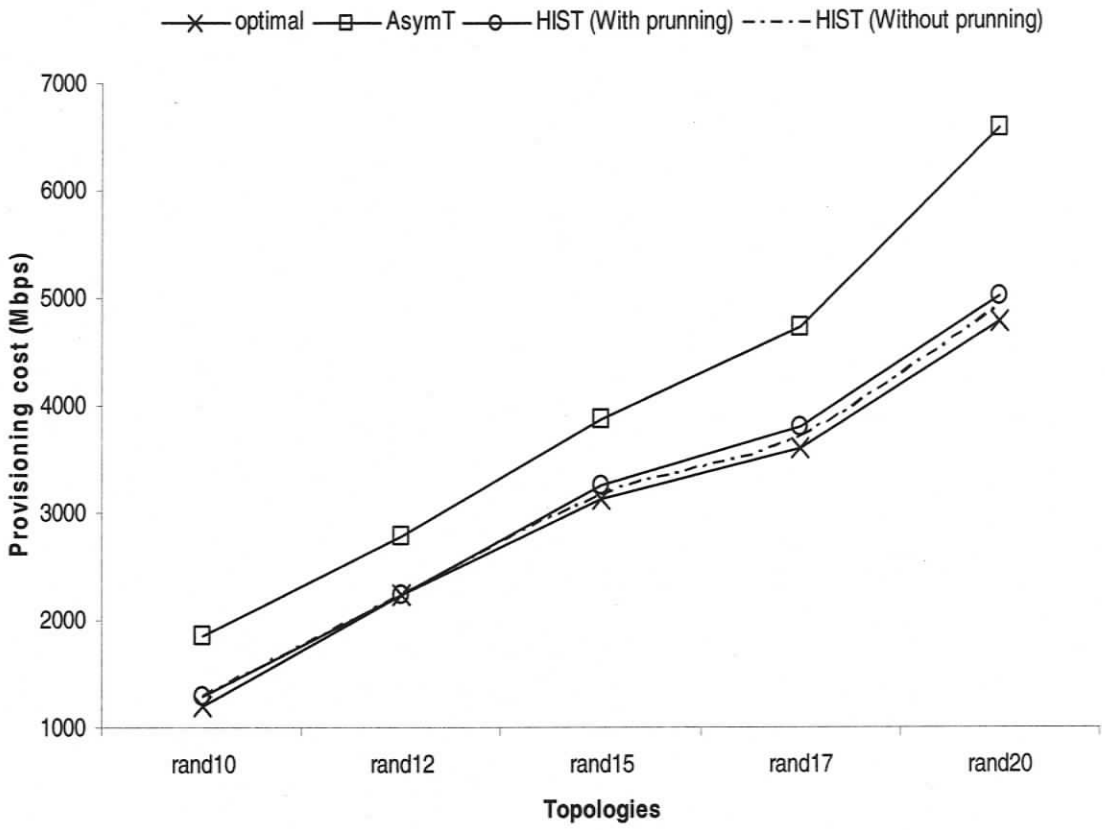


Figure 4.11: Provisioning cost comparison between HIST, *AsymT* and Optimal solution

Table 4.2 summarizes the average number of spanning trees that has been constructed in HIST algorithm and the optimal solution. As shown in this Table, since the total number of spanning trees to find the optimal solution increases dramatically, we calculated the optimal solution for networks up to 20 nodes.

Table 4.2: Number of constructed spanning trees

Algorithm / Topology	HIST (Without pruning)	HIST (With pruning)	Optimal
rand10	32	19	63
rand12	87	36	759
rand15	231	64	10501
rand17	577	100	553067
rand20	4179	269	7625186

From Figure 4.11 and Table 4.2, the following observations can be made:

- The provisioning cost of HIST algorithm is very close to the optimal solution while it requires fewer spanning trees to be constructed.
- There is not much difference between the provisioning costs of HIST algorithm with the pruning scheme compared to HIST without pruning scheme. However using the pruning scheme decreases the total number of constructed spanning trees by nearly 50 percent. Thus we implement the pruning scheme and refer to HIST algorithm with pruning scheme as HIST algorithm for the rest of this Chapter.
- The *AsymT* algorithm requires higher provisioning cost compared to HIST and optimal solutions.

The performance of *AsymT* algorithm and HIST algorithm for larger networks are also compared in Figures 4.12 to 4.16. Figure 4.12 compares the provisioning cost of *AsymT* algorithm with HIST algorithm. The results show that our HIST algorithm requires less bandwidth provisioning over all considered topologies.

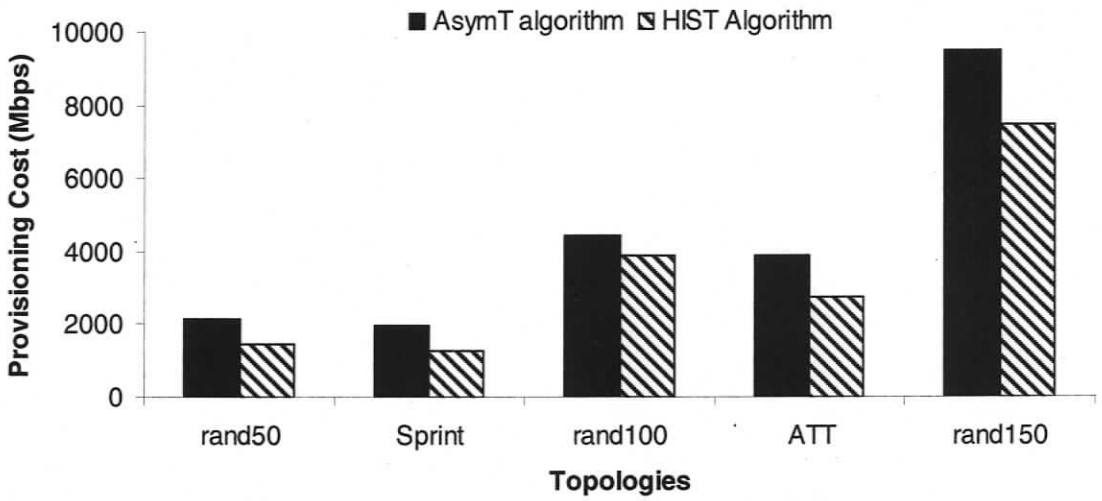


Figure 4.12: Provisioning cost comparison between *AsymT* and HIST algorithms

Figure 4.13 shows that the execution time of HIST algorithm is far less than that of the *AsymT* algorithm since the former iterates over VPN endpoints while the latter iterates over all network nodes.

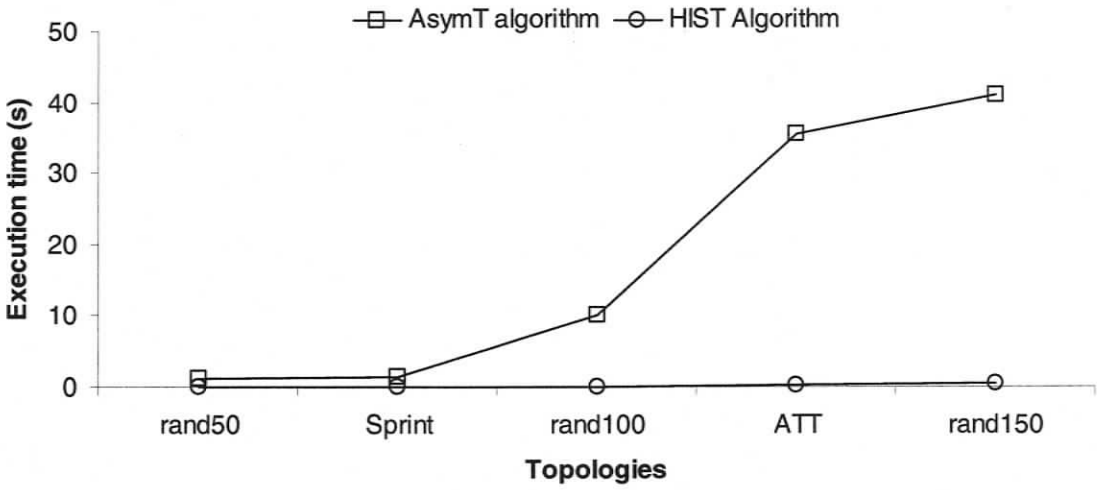


Figure 4.13: Execution time comparison between *AsymT* and HIST algorithms

Figures 4.14 and 4.15 illustrate the effect of increasing the number of VPN endpoints on total provisioning cost and execution time for a 100 node network based on Waxman model, respectively. The results show that the HIST algorithm finds a tree with smaller cost with low execution time than *AsymT* algorithm.

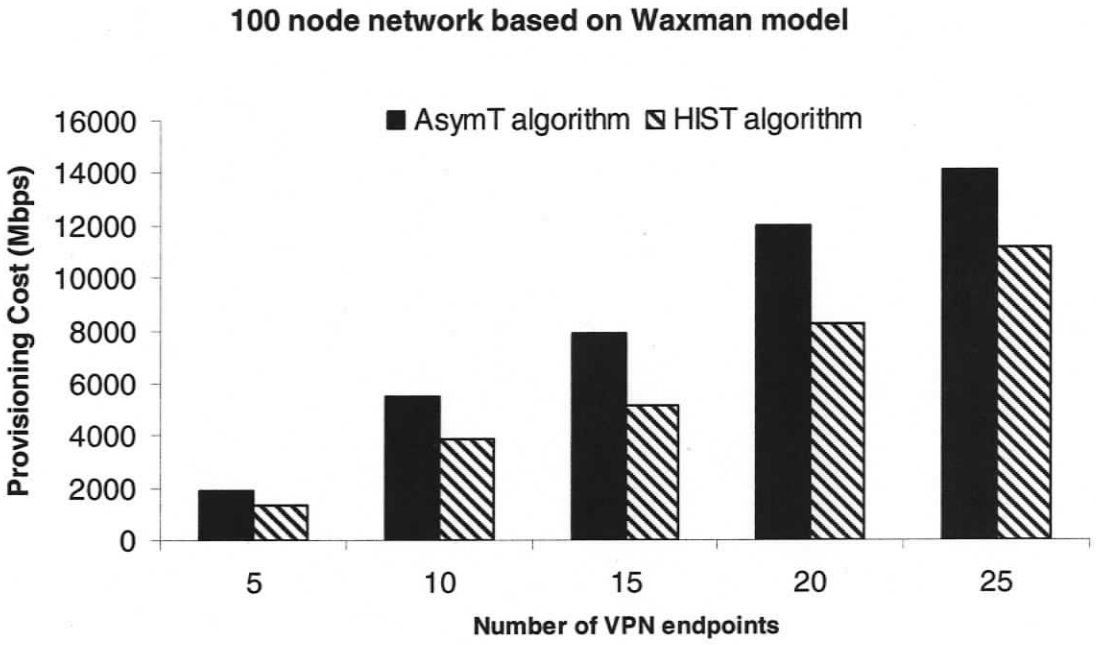


Figure 4.14: Effect of number of VPN endpoints on provisioning cost

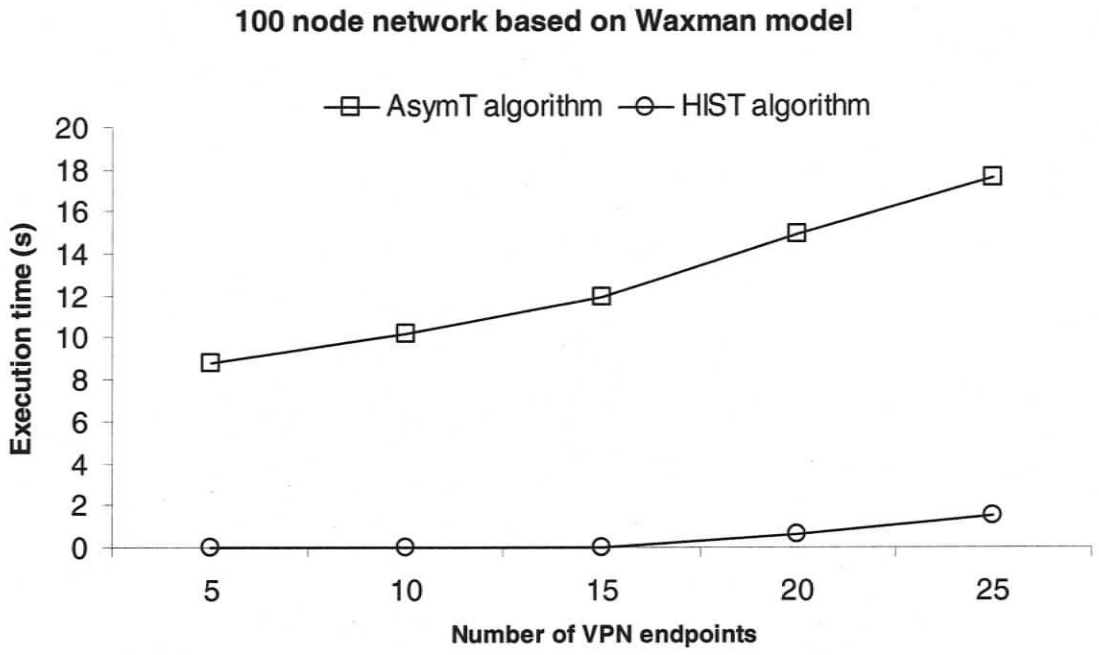


Figure 4.15: Effect of number of VPN endpoints on execution time

Figure 4.16 studies the effect of changing the bandwidth asymmetry ratio on provisioning cost for a 100 nodes network with 20 VPN endpoints. The ratio between ingress and egress bandwidth of all VPN endpoints has been increased from 1 to 256. The results show that our HIST algorithm would still perform better than the *AsymT*.

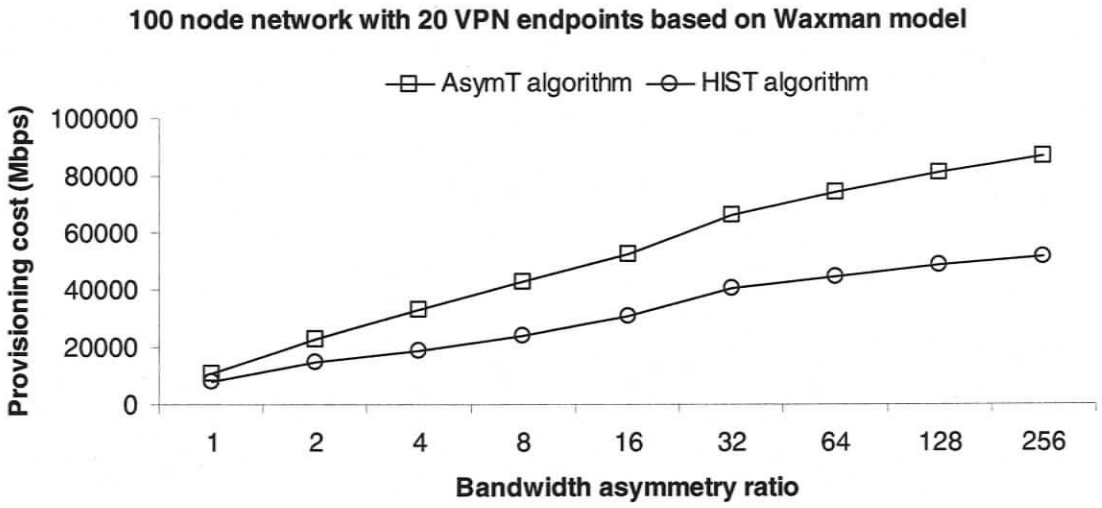


Figure 4.16: Effect of asymmetry ratio on provisioning cost

In summary, our simulation results with synthetic network graphs as well as real Tier-1 ISPs indicate that:

- In most cases, the OBDST algorithm using HIST performs better than OBDST algorithm using *AsymT* and LCLD algorithm in terms of the provisioning cost, the delay diameter, and the execution time.
- The ranking scheme in OBDST algorithm is an effective way to reflect the user's preference in meeting the end-to-end delay limit or lowering the provisioning cost.
- The execution time of OBDST algorithm using HIST is very close to the execution time of LCLD algorithm.
- The VPN trees constructed by HIST require lower bandwidth reservation when compared to *AsymT* algorithm.

- The provisioning cost of VPN trees constructed by HIST is very close to that of the optimal solution for small networks.
- The HIST algorithm's execution time is measured to be far less than that of the *AsymT*'s algorithm.

Chapter 5. Conclusions

5.1. Summary

In this work, we introduced a new ranking scheme based on user preferences to reduce the total provisioning cost while supporting the maximum end-to-end delay limit in the VPN hose model. We connect VPN endpoints using a tree structure and our Optimal Bandwidth and Delay-constrained Shared Tree (OBDST) algorithm attempts to optimize the total bandwidth reserved on edges of the VPN tree as well as supporting the delay limit. Our proposed approach takes into account the user preferences in meeting the delay limits and provisioning cost to find the near optimal solution of resource allocation problem.

Our OBDST algorithm combines our proposed HIST algorithm and Least-Cost-Least-Delay (LCLD) algorithm [8] to find a tree that satisfies the maximum allowable end-to-end delay and provides less provisioning cost compared to LCLD algorithm. Our extensive simulation results show that OBDST algorithm is capable of finding trees with smaller provisioning cost while meeting the end-to-end delay constraints. Moreover, it is observed that with large bandwidth preference, our scheme results in lowering the

provisioning costs and with large delay preference, our scheme results in lowering the maximum end-to-end delay.

We have also proposed a new Hierarchical Iterative Spanning Tree (HIST) algorithm as a solution to the provisioning problem in the VPN hose model without considering delay limit. This algorithm is then used in our OBDST algorithm. Our simulation results with synthetic network graphs as well as real Tier-1 ISPs indicate that the VPN trees constructed by HIST algorithm require less bandwidth reservation when compared to *AsymT* algorithm [23]. Furthermore, our HIST algorithm's execution time is measured to be far less than that of the *AsymT*'s algorithm.

In summary, the major contributions of this work are:

- Introducing OBDST algorithm that uses user preferences to rank trees and finds a VPN shared tree with efficient cost that satisfies maximum end-to-end delay limit.
- Introducing HIST algorithm as a fast and efficient algorithm to reduce the provisioning cost of shared trees when only considering the bandwidth cost.

5.2. Future Work

For future work, the ranking scheme will be extended to include link capacity constraints and other link capacity enhancement features (e.g. increasing a link's bandwidth capacity) into our ranking scheme. This can be done by introducing more parameters such as the enhancement preference and enhancement cost in our ranking scheme. The latter can be interpreted as the preference of network administrator to

increase the bandwidth capacity of a particular link while the former is the monetary cost to perform such operation.

Also, in future research our work can hopefully evolve to what be called the HOPE model (concatenation of "HO" in hose and "PE" in pipe) as a new hybrid VPN service model that combines hose and pipe models. It will provide the customer and the service provider with the same flexibilities of the hose model such as ease of specification and multiplexing gain, but unlike the pipe model, will not require extensive traffic matrix estimation. The HOPE model will guarantee minimum end-to-end bandwidth and maximum end-to-end delay bounds, i.e., the model will offer maximum end-to-end delay limit guarantee from a given endpoint to the set of all other endpoints in the VPN, and minimum bandwidth requirement guarantee for certain VPN endpoints for which the minimum rate is specified.

Chapter 6. Bibliography

[1] B. Davie, Y. Rekhter, "MPLS technology and applications', San Mateo, CA, Morgan Kaufmann, 2000.

[2] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, J. E. van der Merwe, "A flexible model for resource management in virtual private networks", ACM SIGCOMM, vol. 29 (4), August 1999, pp. 95–108.

[3] S. Fotedar, M. Gerla, P. Crocetti, L. Fratta, "ATM virtual private networks", Commun. ACM, vol. 38, 1995, pp. 101–109.

[4] Rocketfuel project, Computer Science and Engineering, Univ. of Washington. [Online]. Available: http://www.cs.washington.edu/research/networking/rocketfuel

[5] E. Rosen and Y. Rekhter, "RFC2547: BGP/MPLS VPNs", IETF. [Online]. Available: http://www.ietf.org/rfc/rfc2547.txt.

[6] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, "RFC 2764: A Framework for IP Based Virtual Private Networks", IETF. [Online] Available: http://www.ietf.org/rfc/rfc2764.txt.

[7] J. T. Buckwalter, "Frame Relay: Technology and Practice", Addison-Wesley Professional, 1999.

[8] L. Zhang, J. Muppala, S. Chanson, "Provisioning virtual private networks in the hose model with delay limits", Hong Kong University, International Conference on Parallel Processing, 2005, pp.211-218.

[9] A. Kumar, R. Rastogi, A. Silberschatz, B. Yener, "Algorithms for provisioning virtual private networks in the hose model", IEEE/ACM Transaction on Networking, vol. 10(4), 2002, pp. 565-578.

[10] S. Hakimi. "Optimal locations of switching centers and medians of a graph", Operations Research, vol. 12, 1964, pp. 450-459.

[11] S. Kent, R. Atkinson, "RFC2401: Security Architecture of the Internet Protocol", IETF. [Online]. Available: http://www.ietf.org/rfc/rfc2401.txt.

[12] A. Shioura, A. Tamura, T. Uno, "An optimal algorithm for scanning all spanning trees of undirected graph", SIAM J .Comput. 26(3), 1997, pp. 678- 692.

[13] E. W. Dijkstra, "A note on two problems in connection with graphs", Numerische Math. 1, 1995, pp. 269-271.

[14] F.K. Hwang, D.S. Richards, P. Winter, "The Steiner tree problem", Elsevier, North-Holland, 1992.

[15] P. P. Mishra, H. Saran, "Capacity management and routing policies for voice over IP traffic", IEEE Network, vol. 14 (2), 2000, pp. 20-27.

[16] S. Firestone, T. Ramalingam, S. Fry, "Voice and Video conferencing fundamentals", Cisco Press, 2007.

[17] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, "RSVP-TE: extensions to RSVP for LSP tunnels", RFC 3209, 2001.

[18] D. Benhaddou, W. Alanqar, "Layer 1 Virtual Private Networks in Multidomain Next-Generation Networks", IEEE communications magazine, vol. 45 (4), pp. 52-58.

[19] S. Hakimi, A. F. Schmeichel, J. G. Pierce, "On p-centers in Networks", Transportation Science, vol. 12, 1978, pp. 1-15..

[20] R. Yuan, W. T. Strayer, "Virtual private networks: technologies and solutions", Addison-Wesley, 2001.

[21] S. Raghunath, K. K. Ramakrishnan, "Resource management for virtual private networks", IEEE Communications Magazine, vol. 45 (4), April 2007, pp. 38-44.

[22] A. Gupta, A. Kumar, T. Roughgarden, "Simpler and better algorithms for network design", ACM Symposium on Theory of Computing, 2003.

[23] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, B. Yener, "Provisioning a virtual private network: a network design problem for multicommodity flow", Proc. of the 33rd ACM Symposium on Theory of Computing (STOC), 2001, pp. 389-398.

[24] G. F. Italiano, S. Leonardi, G. Oriolo, "Design of networks in the hose model", Proc. of the 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE), 2002, pp. 65–76.

[25] A. Juttner, I. Szabo, A. Szentesi, "On bandwidth efficiency of the Hose resource management model in Virtual Private Networks", In Proc. INFOCOM, vol. 1, 2003, pp. 386-395.

[26] http://www.vpn-technology.com/

[27] G. F. Italiano, R. Rastogi, B. Yener, "Restoration algorithms for virtual private networks in the hose model", Proc. of IEEE INFOCOM, 2002, pp. 131 - 139.

[28] T. Erlebach, M. Ruegg, "Optimal bandwidth reservation in hose model VPNs with multi-path routing", In Proc. INFOCOM , vol. 4, 2004, pp. 2275-2282.

[29] ITU-T Recommendation H.323: Infrastructure of Audio-Visual Services – Systems and Terminal Equipment for Audio-Visual Services: Packet-based Multimedia Communications Systems. Draft Version 4, 2000.

[30] L. Harte, "Introduction to IP Television", Althos Publishing, 2005.

[31] B. M. Waxman, "Routing of multipoint connections", IEEE Journal on Selected Areas in Communications, vol. 6(9), 1988, pp. 1617-1622.

[32] A. Kumar, R. Rastogi, A. Silberschatz, B. Yener, "Algorithms for provisioning virtual private networks in the hose model", Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications SIGCOMM '01, vol. 31 (4), pp. 135 – 146.

[33] V. Sharma, N. Ghani, L. Fang, "Virtual Private Networks {Guest Editorial}", IEEE Communications Magazine, vol. 45 (4), April 2007, pp. 24-25

[34] C. Metz, "The latest in VPNs: part II", IEEE Internet Computing, vol. 8(3), May 2004, pp. 60 – 65.

[35] C. Cavdar, A. Gencata, B. Mukherjee, "CATZ: Time-Zone-Aware Bandwidth Allocation in Layer 1 VPNs", IEEE communications magazine, vol. 45 (4), pp. 60-66.

[36] D. S. Hochbaum, "Approximation Algorithms for NP-Hard Problems", Boston, MA, 1997.

[37] W. C. Tat, L. King-Shan , K. L. Yeung, P. W. Chi, "Routing algorithm for provisioning symmetric virtual private networks in the hose model", IEEE Global Telecommunications Conference, vol. 2, 2005, pp. 1-5.

Chapter 7. Appendix

7.1. Enumerating Spanning Trees

This section provides details about the algorithm proposed by Shioura et al. in [12] to enumerate all the spanning trees of a graph. We used a modified version of this algorithm, called MODIFIED_SHIOURA procedure, in Section 3.3.2.2.

The main idea used in [12] is using the so-called fundamental cuts and fundamental cycles: Let T be a spanning tree of graph $G = (V, E)$. For any edge $f \in T$, the deletion of f from T yields two connected components. The *fundamental cut* associated with T and f is defined as the set of edges connecting these components and is denoted by $Cut(T \setminus f)$. Likewise, the *fundamental cycle* associated with T and $g \notin T$ is defined as the set of edges contained in the unique cycle of $T \cup g$ and is denoted as $Cyc(T \cup g)$.

Consider the graph G_1 and the tree T_1 in Figure 7.1 In this example $Cyc(T_1 \cup e_7) = \{e_7, e_2, e_3\}$ and $Cut(T_1 \setminus e_1) = \{e_1, e_5, e_6\}$.

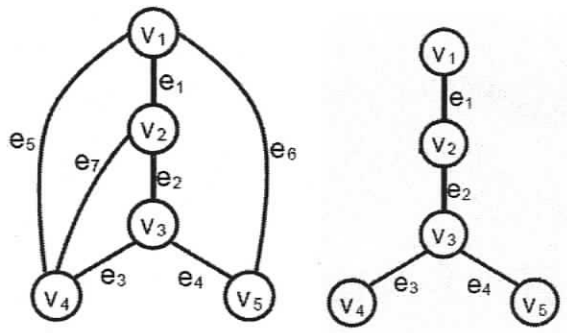


Figure 7.1: Graph G_1 (left) and tree T_1 (right) [12]

Further, the authors showed that for any edge $f \in T$ and any edge $g \in Cut(T \setminus f)$ replacing edge f in T with edge g (denoted by $T \setminus f \cup g$) is a spanning tree. In our example in Figure 7.1, replacing edge $e_1 \in T_1$ with edge $e_5 \in Cut(T_1 \setminus e_1)$ results in another spanning tree of G_1 with edge set $\{e_2, e_3, e_4, e_5\}$. Similarly, for any $g \notin T$ and any edge $f \in Cyc(T \cup g)$, replacing edge f with edge g in T (denoted by $T \cup g \setminus f$) is also a spanning tree. In our sample graph G_1, assuming $g = e_7$, replacing edge $e_2 \in Cyc(T_1 \cup e_7)$ with edge e_7 will result in another spanning tree of G_1 with edge set $\{e_1, e_3, e_4, e_7\}$. These properties are useful for enumerating spanning trees because by using fundamental cuts or cycles, different spanning tree can be constructed from a given one by exchanging exactly one edge.

It is assumed in [12] that the set of edges ($e_i \in E$) and vertices ($v_i \in V$) in G are ordered such that $v_1 < v_2 < \ldots < v_V$ and $e_1 < e_2 < \ldots < e_E$. Moreover, the authors introduced five assumptions regarding the vertex set and the edge set of G. Specifically, the first two assumptions are that tree T^0 is a depth-first spanning tree of G and the set of edges in T^0 is $\{e_1, \ldots, e_{V-1}\}$. The following lemma proved in [12] is the main idea to enumerate all the spanning trees:

LEMMA 7.1: Given a spanning tree $T^c \neq T^0$ and the edge f to be the smallest edge in T^0 that is not in T^c, there will be a unique edge g in $\text{Cyc}(T^c \cup f) \cap \text{Cut}(T^0 \setminus f) \setminus f$ where replacing edge g with edge f in T^c is also a spanning tree T^p. T^p is called the parent of T^c and T^c a child of T^p.

Back to graph G_1 and tree T^0 in Figure 7.1, let T^c be the graph depicted in Figure 7.2 (a). The following steps based on the above lemma would result in finding parent T^p of tree T^c:

- $f = e_1$
- $\text{Cyc}(T^c \cup f) = \{e_1, e_5, e_7\}$
- $\text{Cut}(T^0 \setminus f) = \{e_1, e_5, e_6\}$
- $\text{Cyc}(T^c \cup f) \cap \text{Cut}(T^0 \setminus f) = \{e_1, e_5\}$
- $\text{Cyc}(T^c \cup f) \cap \text{Cut}(T^0 \setminus f) \setminus f = \{e_5\}$
- $g = e_5$

Replacing edge e_5 in T^c with e_1 results in tree depicted in Figure 7.2 (b) as new spanning tree T^p.

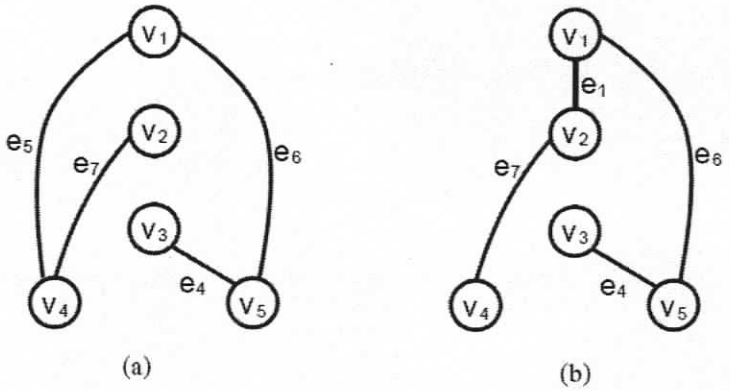


Figure 7.2: Spanning trees T^c (left) and T^p (right)

Further, in [12] a similar lemma is given to build child T^c of an arbitrary spanning tree T^p of G using edges belong to $Cut(T^p \setminus f) \cap \text{Cut}(T^0 \setminus f) \setminus f$. Figure 7.3 shows the algorithm all-spanning-trees in [12] in which $Cut(T^p \setminus e_j) \cap \text{Cut}(T^0 \setminus e_j) \setminus e_j$ is abbreviated as $Entr(T^p, e_j)$ on the grounds that any edge in $Cut(T^p \setminus e_j) \cap \text{Cut}(T^0 \setminus e_j) \setminus e_j$ can be 'entered' into T^p in place of e_j.

```
ALGORITHM all-spanning-trees(G);
  input: a graph G with a vertex set {v_1, ..., v_V} and an edge set {e_1, ..., e_E};
begin
    by using a depth-first search,
        • find a depth-first spanning tree T^0 of G,
        • sort vertices and edges to satisfy Assumptions 2, 3, 4, and 5;
    output("e_1, e_2, ..., e_{V-1}, tree,") ;{output T^0}
    find-children(T^0,V-1);
end.

PROCEDURE find-children(T^p,k);
  input: a spanning tree T^p and an integer k with e_k < Min(T^0 \ T^p);
begin
    if k ≤ 0 then return;
    for each g ∈ Entr(T^p, e_k) do begin
                                  {output all children of T^p not containing e_k}
        T^c := T^p\e_k∪g;
        output("-e_k, +g, tree,");
        find-children(T^c,k-1);   {find the children of T^c}
        output("-g, +e_k,");
    end;
    find-children(T^p,k-1);       {find the children of T^p not containing e_{k-1}}
end.
```

Figure 7.3: All-spanning-trees algorithm [12]

Based on the above properties, procedure find-children in Figure 7.3 finds all children of each spanning tree. When it is called with two arguments T^p and k, it finds all children of T^p not containing an edge e_k. Whenever it finds such a child T^c, it recursively calls itself again to find all children of T^c. In this stage, arguments are set to T^c and k–1. If all children of T^p not containing e_k have been found, it recursively calls itself again to find all

children of T^p not containing e_{k-1}. In this case, arguments are T^p and $k-1$. Initially, algorithm all-spanning-trees(G) calls find-children( ) with arguments T^0 and $V-1$, and all spanning trees of G are found. Figure 7.4 shows a graph G_2 and the enumeration tree of spanning trees in G_2.

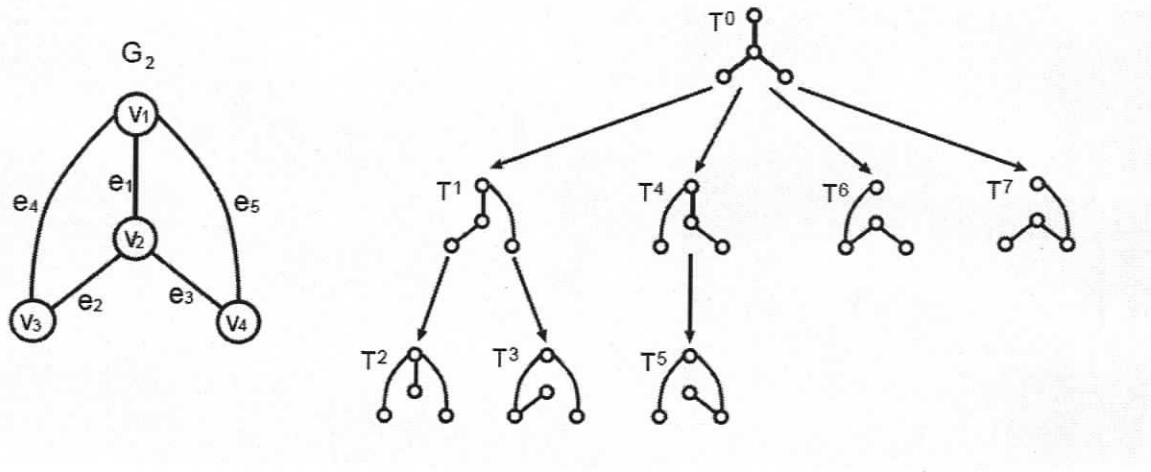


Figure 7.4: Graph G_2 and enumeration of its spanning trees [12]

It is proved in [12] that all-spanning-trees() algorithm outputs each spanning tree exactly once. Further, the time and space complexities of their algorithm is measured to be $O(N+V+E)$ and $O(V+E)$, respectively, where N is the number of spanning trees, V is the number of vertices, and E is the number of edges of a graph. The authors concluded that their algorithm is optimal in sense of time and space complexities.