Computer Networks 52 (2008) 3130–3147

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Resource optimization algorithms for virtual private networks using the hose model

Monia Ghobadi *, Sudhakar Ganti, Gholamali C. Shoja

Department of Computer Science, University of Victoria, BC, Canada V8W 3P6

ARTICLE INFO

Article history:
Received 23 January 2008
Received in revised form 28 July 2008
Accepted 5 August 2008
Available online 14 August 2008

Responsible editor: J. Domingo-Pascual

Keywords:
Virtual private networks
Hose model
Quality of service
Provisioning cost
Spanning tree

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, service 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 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.

1. Introduction

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)[6]. 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.

A 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

* Corresponding author. Tel.: +1 647 899 0370.
E-mail addresses: monia@cs.uvic.ca (M. Ghobadi), sganti@cs.uvic.ca (S. Ganti), gshoja@cs.uvic.ca (G. C. Shoja).

1389-1286/$ - see front matter Crown Copyright © 2008 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.08.010

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.

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. As depicted in Fig. 1, 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.

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. Fig. 2 illustrates an example of the use of the hose model. Each

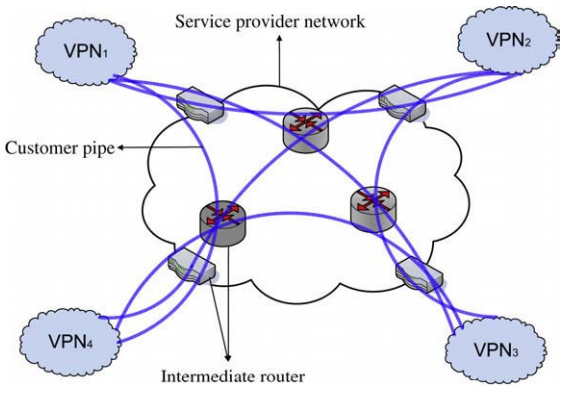

Fig. 1. VPN pipe model.

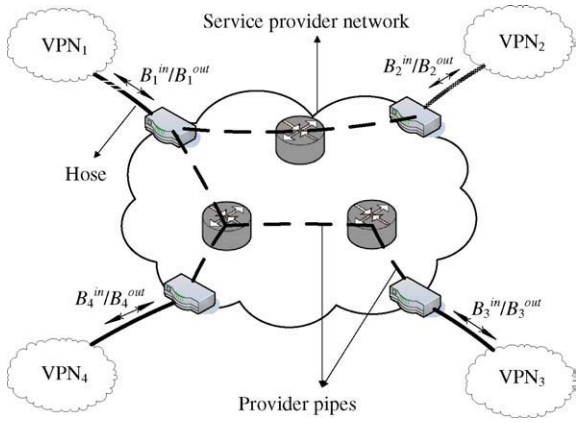

Fig. 2. VPN hose model.

VPN endpoint i is connected to the network by a hose, which is specified by its aggregate ingress and egress bandwidths (B_i^{in} and B_i^{out}, respectively). B_i^{in} is the amount of aggregate traffic from all endpoints to endpoint i and B_i^{out} 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.

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 explain in Section 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].

2. 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_p^{in} and B_p^{out}, respectively.

In this paper, we refer to the total bandwidth reserved on all links in the provisioned VPN as the provisioning cost of the solution. The basic problem of finding a reservation of minimum cost in the hose model may be subject to various conditions. First, ingress and egress bandwidth B_p^{in} and B_p^{out}, $p \in P$ may be defined in three different ways:

(1) The symmetric case: $B_p^{in} = B_p^{out}, \forall p \in P$.
(2) The sum-symmetric case: $\sum_{p \epsilon P} B_p^{in} = \sum_{p \epsilon P} B_p^{out}, \forall p \epsilon P$.
(3) The asymmetric or general case: B_p^{in} and B_p^{out} are arbitrary values.

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

(1) *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 VPN endpoint i to VPN endpoint j, $\min(B_i^{out}, B_j^{in})$ units of bandwidth is reserved on each link along the path.
(2) *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 [8].
(3) *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.
(4) *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 the 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 [15] and IP-TV [30], 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.

3. Motivation and related work

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 that 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 paper 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.

A number of provisioning algorithms for VPNs in the hose model have been proposed [2,9,21–23,27,28,32,33]. 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 [9,32],[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.
- 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 [34].

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:

$$C_T(u,v) = \min\left\{\sum_{l\in P_u^{(u,v)}} B_l^{out}, \sum_{l\in P_v^{(u,v)}} B_l^{in}\right\} \quad (1)$$

Similarly, the bandwidth that must be reserved on link (v, u) in the direction from v to u can be shown to be:

$$C_T(v,u) = \min\left\{\sum_{l\in P_u^{(u,v)}} B_l^{in}, \sum_{l\in P_v^{(u,v)}} B_l^{out}\right\} \quad (2)$$

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:

$$C_T = \sum_{(u,v)\in T} C_T(v,u) \quad (3)$$

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

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

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 3–5 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 [33] the authors enhanced the algorithm in [9,32] 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 [16,29].

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.

In [18], we proposed a new ranking approach to enhance the hose model to guarantee delay requirements between endpoints while optimizing the provisioning cost. Further in [5] we presented a new hierarchical approach, called HIST algorithm, for optimal resource provisioning in the VPN hose model. Our HIST algorithm is more efficient in terms of time complexity and provisioning cost than the one used in [18]. In this work, we aim 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.

4. Problem statement, proposed solution and methodology

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 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. In [11] we proved that OBDSTP is NP-hard.

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].

5. 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 (3) from Section 3, 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 6, 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 Section 3, *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 Section 9.

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: $\forall T \in \text{ODST} \vee \text{OBST}$,

$$\begin{aligned} Rank(T) = {} & delay\_preference \\ & \times \frac{(delay\_diameter)_T}{maximum\_allowable\_delay} \\ & + bandwidth\_preference \\ & \times \frac{(bandwdith\_cost)_T}{bandwidth\_threshold} \end{aligned} \tag{4}$$

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 (3). As explained earlier, the bandwidth threshold is the maximum bandwidth cost for trees in ODST set. In this work, the user preferences are the delay and bandwidth preference. These 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. Larger delay preference works in favor of the trees with smaller delay diameter.

As an example consider the network N_1 depicted in Fig. 3. It contains a network with six 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.

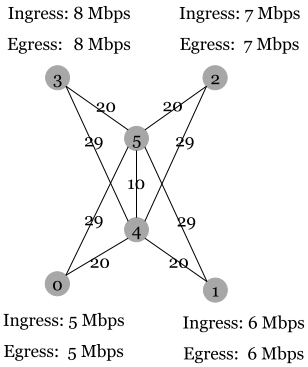

Fig. 3. Sample network N_1.

The result of phase 1 of OBDST algorithm is illustrated in Fig. 4 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 Fig. 4a which is 74 Mbps.

Now, in phase 3, we execute our HIST algorithm, explained in detail in Section 6, 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 Fig. 5 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 Figs. 4 and 5.

In phase 4, the ranking is performed for each tree in ODST and OBST sets using Formula (4). 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 Fig. 4a = 74/74 + 50/58 = 1.86
Rank of tree in Fig. 4b = 52/74 + 58/58 = 1.70
Rank of tree in Fig. 5a = 52/74 + 58/58 = 1.70
Rank of tree in Fig. 5b = 62/74 + 59/58 = 1.85
Rank of tree in Fig. 5c = 74/74 + 89/58 = 2.53

For this example, both trees in Fig. 4b and Fig. 5a 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 Fig. 4a that has a considerably larger bandwidth cost.

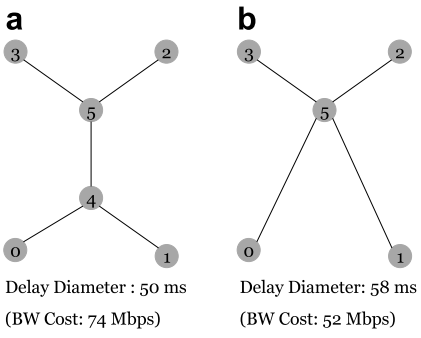

Fig. 4. ODST set, the result of performing phase 1.

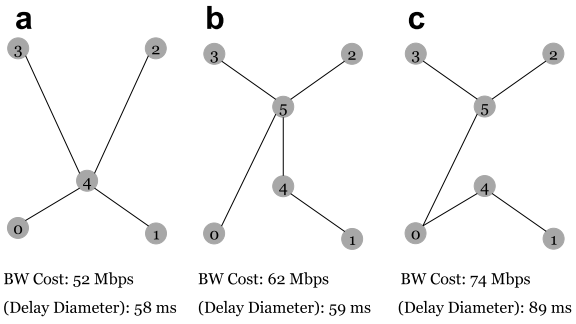

Fig. 5. OBST set, the result of performing phase 3.

6. 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 Section 3, 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 was measured to be far less than that of *AsymT* algorithm. The simulation results will be discussed in more detail in Section 9. In the following, the basic idea behind our hierarchical approach will be explained.

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.

6.1. Step 1: ITERATIVE_SPANNING_TREE procedure

As described in Section 3, 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_i^{in} and B_i^{out}, 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 $E' = \{(u, v) | u, v \in P \text{ and } u \neq v\}$ (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% 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. Fig. 6 contains the *Iterative_Spanning_Tree* procedure which builds graph G' and outputs $T_{G'}$.

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% 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. In the following section we provide a short review of their work along with our modification to the algorithm.

6.1.1. Modified_Shioura procedure

As explained in [12], 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.

Figs. 7 and 8 contain the *Modified_Shioura* and *Find_Children* algorithms based on Shioura et al.'s *all-spanning trees* and *find-children* procedures provided in [12]. 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.

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

ITERATIVE_SPANNING_TREE (P)

1. $T_{G'} = \emptyset$, $G' = \emptyset$
2. **for** each vertex $v \in P$
3. $G' \leftarrow T_{G'}$
4. add new vertex v to G'
5. add an edge from v to all other nodes in G'
6. $T_{G'} \leftarrow$ Modified_Shioura(G')
7. **return** ($T_{G'}$)

Fig. 6. ITERATIVE_SPANNING_TREE procedure.

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** ($minTree$)

Fig. 7. MODIFIED_SHIOURA procedure.

FIND_CHILDREN(T^p, k, *minTree*)

1. **if** $k \leq 0$ **then return** *minTree*
2. **for** each edge $g \in Entr(T^p, e_k)$ 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, $k-1$, m*inTree*)
8. Find_Children(T^p, $k-1$, *minTree*)

Fig. 8. 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 Fig. 8 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 2 in Section 9, 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.

6.2. 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 map each virtual edge (u, v) in $T_{G'}$ to the physical path between VPN endpoints u and v in the original network G.

Fig. 9 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.

HIERARCHICAL_EXTENSION($T_{G'}$, G)

1. *finalTree* = ∅
2. **for** each edge $e \in G$
3. weight(e) ←1
4. **for** each edge $(u, v) \in T_{G'}$
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 $g \in path_{uv}$ and g $\notin$ *finalTree*
8. add g to *finalTree*
9. weight(g) ← 0
10. **return** (*finalTree*)

Fig. 9. Hierarchical_Extension procedure.

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.

As an example, consider the network N_2 in Fig. 10. The four VPN endpoints 1–4 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 Fig. 11. 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.

Fig. 12 depicts the steps of performing the *Iterative_Spanning_Tree* procedure. Fig. 12a, c, e, and g illustrate the virtual topologies G' and Fig. 12b, d, f, and h depict the trees $T_{G'}$ in iterations 1–4, respectively.

Fig. 12a and b show the first iteration of *Iterative_Spanning_Tree* procedure while there is only one node in the virtual topology. In Fig. 12c node 2 is added to G' during the second iteration of the procedure. Only the spanning tree connecting nodes 1 and 2 is shown in Fig. 12d. Furthermore, Fig. 12e and g show the result of adding nodes 3 and 4 to G', respectively. Fig. 12f and h depict the result

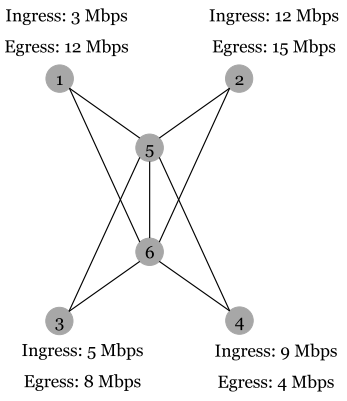

Fig. 10. Sample network N_2.

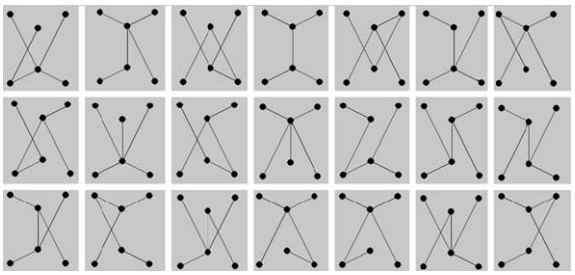

Fig. 11. Some spanning trees of N_2.

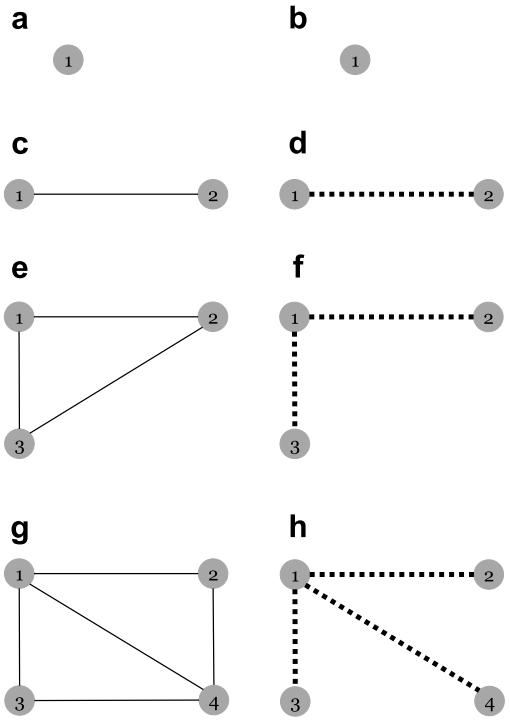

Fig. 12. Steps of performing Iterative_Spanning_Tree procedure on N_2.

of applying the *Modified_Shioura* procedure to find a spanning tree in Fig. 12e and g. The final result of *Iterative_Spanning_Tree* procedure is the tree $T_{G'}$ depicted in Fig. 12h 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 Fig. 12e is three and total number of spanning trees for the graph in Fig. 12g 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.

The result of performing the *Hierarchical_Extension* procedure is depicted in Fig. 13. Fig. 13a shows the original network graph with each link having weight equal to 1. The procedure starts by selecting an edge from its input

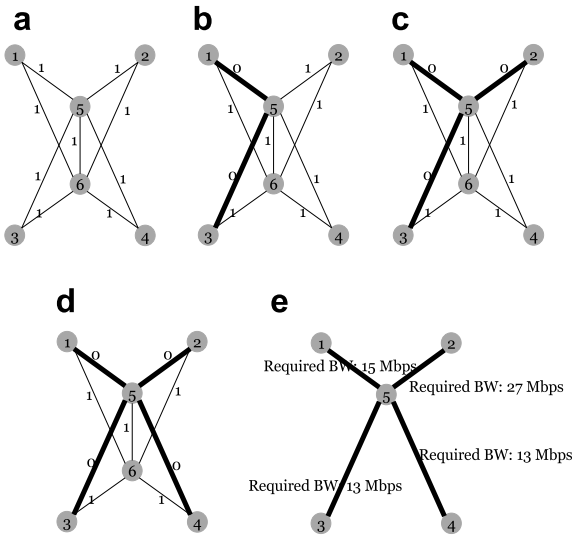

Fig. 13. Steps of performing Hierarchical_Extension procedure on N_2.

$T_{G'}$. Let us assume that the edge (1, 3) is the first edge selected from $T_{G'}$. In Fig. 13b 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. Fig. 13c and d show the result of finding the shortest path between nodes (1, 2) and nodes (1, 4), respectively. The *finalTree* is shown in Fig. 13e 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 Fig. 11) 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.

6.3. 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 5, we added another step to the final iteration of procedure *Iterative_Spanning_Tree* in Fig. 6. 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 5, 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.

7. Correctness properties

In this section, we provide proof for several properties of our algorithms. These properties are used to prove the correctness of our algorithms. We start with explaining how we satisfy the only assumption in the OBDST algorithm (Assumption 1) followed by two lemmas (Lemmas 1 and 2) proving that OBDST algorithm will always have a non-empty output. Furthermore, through Lemmas 3–6 we prove that the output of our HIST algorithm is in fact a tree connecting all VPN endpoints.

Assumption 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 3.

Lemma 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 2. *OBDST algorithm will return a non-empty tree*

Proof. According to Assumption 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.

Lemma 3. *The "minTree" returned by Modified_Shioura procedure in Fig. 7 is a non-empty spanning tree of graph G'.*

Proof. Recall from Section 6.A. 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 4. *$T_{G'}$ returned by Iterative_Spanning_Tree procedure in Fig. 6 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 $p_1, \ldots, p_k$ are 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, $1 \leqslant i, j \leqslant k$. By adding edges (p_{k+1}, p_i), $1 \leqslant i \leqslant k$ to G', there will also be a path between p_{k+1} and nodes p_i, $1 \leqslant i \leqslant k$ 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 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 5. *The "finalTree" returned by Hierarchical_Extension procedure in Fig. 9 is loop-free.*

Proof. Recall from Section 6.B. 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 Fig. 13a–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 Fig. 14a.

(*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 Fig. 14b.

(*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 Fig. 14 (c).

Both cases (*a*) and (*b*) are possible and as depicted in Fig. 13a 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 Fig. 14c 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, as illustrated in Fig. 14d.

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 6. *All VPN endpoints are connected in finalTree.*

Proof. According to Lemma 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. □

8. 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.

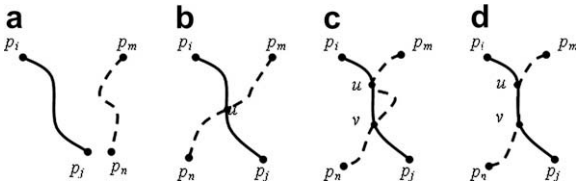

Fig. 14. Relative relation between paths.

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, 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 $1 \leqslant c \leqslant m$.

In phase 3, as explained in Section 6.C., 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 6.C., we add trees with costs less than the bandwidth threshold in the last iteration of Fig. 6 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 Fig. 6. 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 $O(p^2 + m)$.

9. Simulations

We have designed a number of simulation experiments to measure the performance of our proposed OBDST algorithm, described in Section 5, and our proposed HIST algorithm, described in Section 6.

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.

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].

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 1.

We also 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 $3000 \times 2400\ \mathrm{KM}^2$ 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.

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 runs for each topology. Thus for Waxman or real network topologies, we performed 5 runs over the same topology and averaged the results. We calculated 95% confidence intervals as:

$\hat{\theta} - t_{\alpha/2,f}\hat{\sigma}(\hat{\theta}) \leqslant \theta \leqslant \hat{\theta} + t_{\alpha/2,f}\hat{\sigma}(\hat{\theta})$, 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.[2] 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.

As mentioned in Section 5, 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 (4). 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 Figs. 15–21. As mentioned in Section 3, LCLD algorithm is the proposed approach in [8] to enhance the hose model to support the delay limit. Figs. 15–17 compare provisioning cost of LCLD algorithm with OBDST using *AsymT* algorithm, or OBDST using HIST algorithm for scenarios 1–3, respectively. The results show that OBDST using HIST algorithm requires less 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.

Figs. 18–20 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 Figs. 18–20.

Fig. 21 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.

Table 1
Rocketfuel ISP topologies used in our simulations

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

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

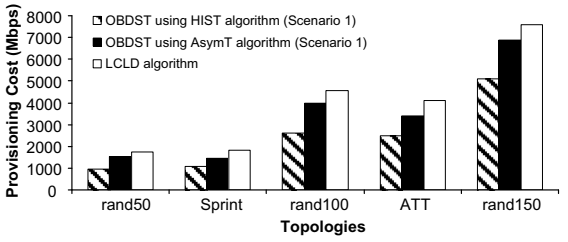

Fig. 15. The provisioning cost comparison: Scenario 1 ($\gamma = 1$).

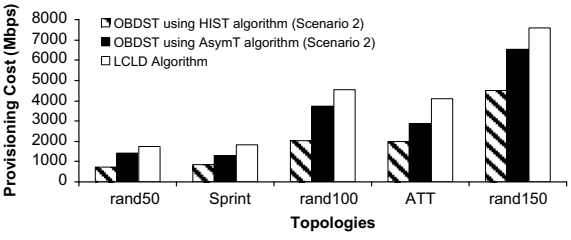

Fig. 16. The provisioning cost comparison: Scenario 2 ($\gamma > 1$).

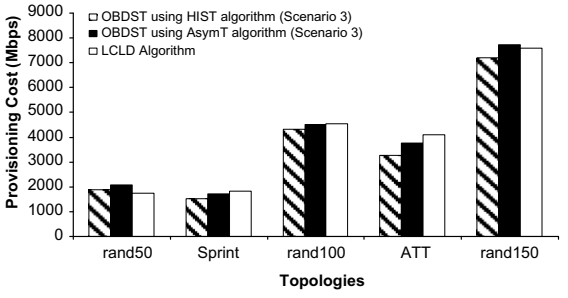

Fig. 17. The provisioning cost comparison: Scenario 3 ($\gamma < 1$).

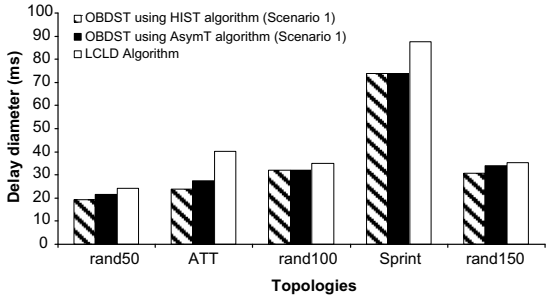

Fig. 18. The delay diameter comparison: Scenario 1 ($\gamma = 1$).

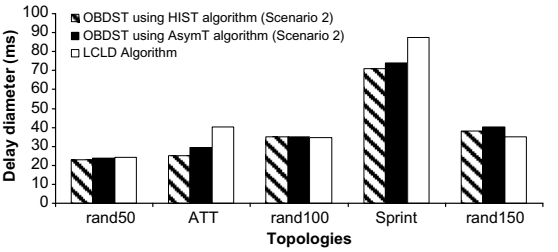

Fig. 19. The delay diameter comparison: Scenario 2 ($\gamma > 1$).

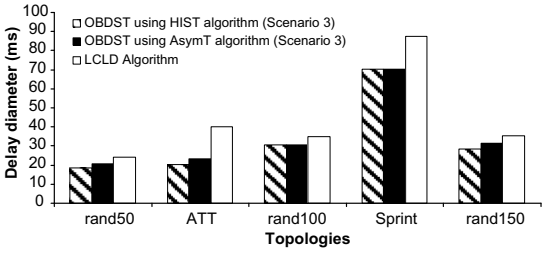

Fig. 20. The delay diameter comparison: Scenario 3 ($\gamma < 1$).

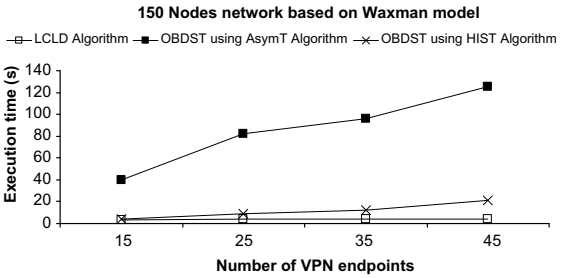

Fig. 21. Effect of number of VPN endpoints on execution time.

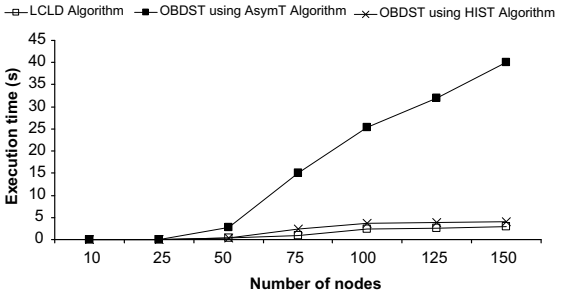

Fig. 22. Effect of number of nodes on execution time.

Fig. 22 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.

Fig. 23 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 Fig. 24 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.

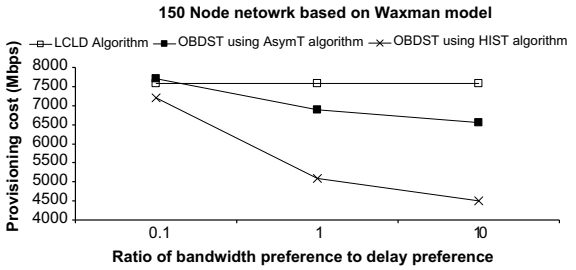

Fig. 23. Effect of γ on provisioning cost.

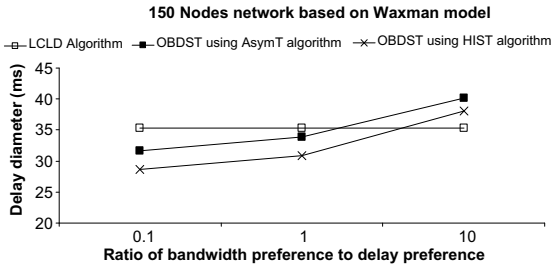

Fig. 24. Effect of γ on delay diameter.

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.

Fig. 25 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.

Table 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.

From Fig. 25 and Table 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. Using the pruning scheme decreases the total number of constructed spanning trees by nearly 50%. Thus we implement the pruning scheme and refer to HIST algorithm with pruning scheme as HIST algorithm for the rest of this section.
- 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 Figs. 26–30. Fig. 26 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.

Fig. 27 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.

Figs. 28 and 29 illustrate the effect of increasing the number of VPN endpoints on total provisioning cost and

Table 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

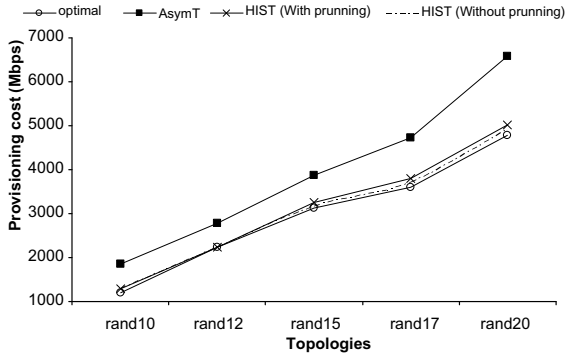

Fig. 25. Provisioning cost comparison between HIST, *AsymT* and Optimal solution.

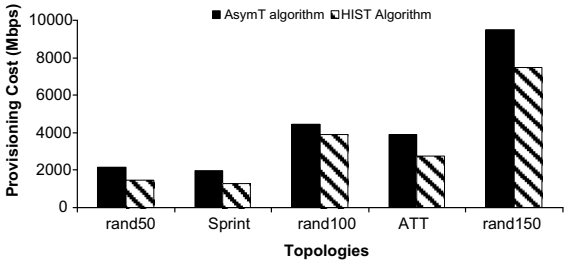

Fig. 26. Provisioning cost comparison between *AsymT* and HIST algorithms.

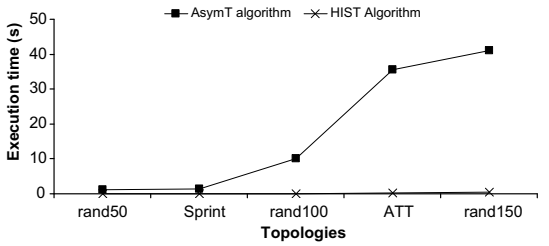

Fig. 27. Execution time comparison between *AsymT* and HIST.

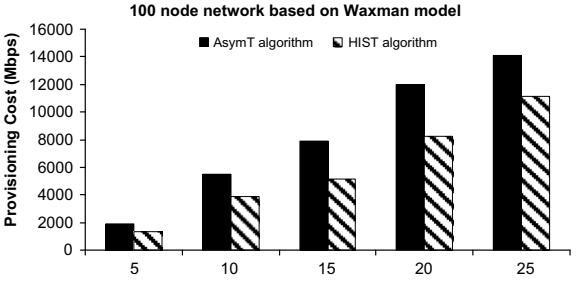

Fig. 28. Effect of number of VPN endpoints on provisioning cost.

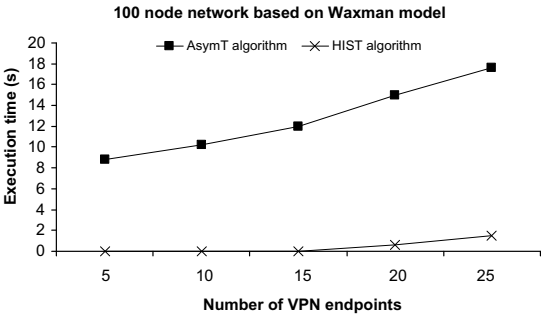

Fig. 29. Effect of number of VPN endpoints on execution time.

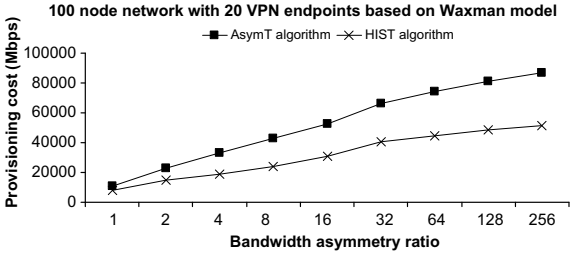

Fig. 30. Effect of asymmetry ratio on provisioning cost.

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.

Fig. 30 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*.

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.

10. Conclusions and future work

In this work, we introduced a new ranking scheme based on user preferences to reduce the total provisioning cost while meeting 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 bandwidth 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.

For future work, one could address the network virtualization problem by providing a scheme to guarantee quality of service on multiple networks, each customized to a specific purpose, running at the same time over a shared VPN tree. An interesting application of this work would be devising tactics to tackle emerging challenges in network management; such as dynamic resource provisioning based on load balancing of the traffic over network links to achieve a congestion free network

References

[1] B. Davie, Y. Rekhter, MPLS Technology and Applications, Morgan Kaufmann, San Mateo, CA, 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 29 (4) (1999) 95–108. August.
[3] S. Fotedar, M. Gerla, P. Crocetti, L. Fratta, ATM virtual private networks, Commun. ACM 38 (1995) 101–109.
[4] Rocketfuel project, Computer Science and Engineering, Univ. of Washington [Online] Available: <http://www.cs.washington.edu/research/networking/rocketfuel>.
[5] M. Ghobadi, S. Ganti, G.C. Shoja, Hierarchical provisioning algorithm for virtual private networks using the hose model, in: Proceedings of the IEEE Global Communications, Globecom 2007, Washington, DC, 2007, pp. 2467–2471 (November).
[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 10 (4) (2002) 565–578.
[10] S. Hakimi, Optimal locations of switching centers and medians of a graph, Operat. Res. 12 (1964) 450–459.
[11] M. Ghobadi, M.Sc. Thesis, Resource Optimization Algorithms for Virtual Private Networks Using the Hose Model, Department of Computer Science, University of Victoria, BC, Canada, 2007.
[12] A. Shioura, A. Tamura, T. Uno, An optimal algorithm for scanning all spanning trees of undirected graph, SIAM J. Comput. 26 (3) (1997) 678–692.
[13] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Math. 1 (1995) 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 14 (2) (2000) 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] M. Ghobadi, S. Ganti, G.C. Shoja, Resource optimization to provision a virtual private network using the hose model, in: Proceedings of the IEEE International Conference on Communications, 2007, pp. 512–517.
[19] S. Hakimi, A.F. Schmeichel, J.G. Pierce, On p-centers in networks, Transportation Science 12 (1978) 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 Commun. Magazine 45 (4) (2007) 38–44.
[22] A. Gupta, A. Kumar, T. Roughgarden, Simpler and better algorithms for network design", ACM Symp. Theory Comput. (2003).
[23] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, B. Yener, Provisioning a virtual private network: a network design problem for multicommodity flow, in: Proceedings 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, in: Proceedings of the Third 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: Proceedings of the 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, in: Proceedings of the IEEE INFOCOM, 2002, pp. 131–139.
[28] T. Erlebach, M. Ruegg, Optimal bandwidth reservation in hose model VPNs with multi-path routing, in: Proceedings of the 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 J. Select. Areas Commun. 6 (9) (1988) 1617–1622.
[32] A. Kumar, R. Rastogi, A. Silberschatz, B. Yener, Algorithms for provisioning virtual private networks in the hose model, in: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications SIGCOMM'01, vol. 31 (4), pp. 135–146.
[33] 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 Telecommun. Conf. 2 (2005) 1–5.
[34] D.S. Hochbaum, Approximation Algorithms for NP-Hard Problems, Boston, MA, 1997.

Monia Ghobadi is currently a Ph.D. student in the Computer Science Department of University of Toronto, Canada. She received her B.Sc. degree from Sharif University of Technology, Iran, and M.Sc. degree from University of Victoria, Canada, in 2005 and 2007, respectively. She was a research assistant in Parallel, Networking and Distributed Applications (PANDA) laboratory, University of Victoria from 2005 to 2007. She has been a member of High Performance Networking (HiPerNet) Group, University of Toronto, since September 2007.

Sudhakar Ganti (S'89 M'96) is a faculty in the department of Computer Science at University of Victoria, BC, Canada since 2004. Prior to joining the university he worked in the telecom industry of Ottawa, Canada for 10+ years in various positions with Nortel Networks, Newbridge Networks, Tropic Networks and Alcatel. His expertise and area of research is in Traffic Management, Traffic Engineering and Quality of Service of high speed networks. He holds several patents, published several conference, journal papers as well as a book on Quality of Service in ATM networks. He obtained his Ph.D. from University of Ottawa, Canada M.Tech from Indian Institute of Technology Kanpur, India and B.Tech from College of Engineering, JNT University, Kakinada, India.

Gholamali C. Shoja is currently a Professor Emeritus in the Department of Computer Science, University of Victoria, Victoria, BC, Canada. He received his BSEE from Kansas State University, his MSEE from Northwestern University and his Ph.D. from University of Sussex in UK. His research area is QoS in networks and multimedia systems. He is a registered Professional Engineer in BC, a Senior Member of IEEE and a voting member of ACM.