Provisioning Virtual Private Networks in the Hose Model with Delay Requirements

Lei Zhang, Jogesh Muppala, Samuel Chanson
Dept. of Computer Science, Hong Kong Univ. of Science and Technology
Clearwater Bay, Kowloon, Hong Kong
{zhanglei, muppala, chanson}@cs.ust.hk

Abstract

Virtual Private Networks (VPN) provide a cost-effective means of meeting the communication needs among several sites. The hose model for VPN configuration alleviates the scalability problem of the pipe model by reserving bandwidth for traffic aggregates instead of between every pair of endpoints. Existing studies on quality of service (QoS) guarantees in the hose model deal only with bandwidth requirements. In this paper we enhance the hose model to specify delay requirements between endpoints. Three categories of algorithms, namely the pipe mesh, the multiple source-based trees, and the shared tree approaches, are then proposed for VPN provisioning. We investigate methods of implementing the shared tree approach to meet the delay requirements with low provisioning cost and small computation overhead.

1. Introduction

Two popular models have been proposed for supporting QoS in Virtual Private Networks (VPNs): the "pipe" model [2] and the "hose" model [4]. The hose model has good characteristics such as flexibility, multiplexing gain and ease of specification [5]. In [4], a hose is realized with a source-based tree resulting in a factor of 2 to 3 in capacity savings over the pipe model. In [7], the hoses are implemented with a single shared tree and the algorithms attempt to optimize the total bandwidth reserved on the edges of the tree. In [6] the bandwidth efficiency of the hose model was studied, where the over-provisioning factor is evaluated in networks with various sizes and node densities. In [3], a multi-path provisioning approach was proposed.

This work described in this paper has been supported by the Research Grants Council of Hong Kong SAR, China (Project No. HKUST6177/04E)

However, no work on the hose model has considered delay, which is becoming important in VPNs with delay-sensitive applications such as Voice over IP [9]. In this paper we enhance the hose model to support delay requirements at the VPN endpoints. We propose three approaches for the enhanced hose model: the pipe mesh approach, the multiple source-based trees approach, and the shared tree approach. For the shared tree approach we consider the following questions: (1). How to make the shared tree support the delay requirements? (2). How to achieve high statistical multiplexing gain given that the delay requirement is satisfied? (3). How to reduce the computational overhead? The first issue is solved by formulating a minimum diameter Steiner tree (MDStT) problem. The second problem is proved to be NP-hard and we use heuristics to build trees with low provisioning cost. The third issue is tackled by a pruning technique.

The rest of the paper is structured as follows. The enhanced hose model, supporting both delay and bandwidth requirements, is described in Section 2. Section 3 presents three approaches for the enhanced hose model and discusses their advantages and disadvantages. The three problems with the the shared tree approach are further studied in Sections 4, 5, and 6 respectively. Simulation results comparing the performance of the proposed algorithms are presented in Section 7. Finally, Section 8 concludes the paper.

2. The Enhanced Hose Model

We model the network as a graph $G = (V, E)$ where V is the set of nodes and E is the set of bidirectional links connecting the nodes. Each link (i, j) is associated with two QoS metrics – the bandwidth capacity L_{ij} and the delay D_{ij}. The delay value of a path is defined as the sum of the delay values of all links along the path.

The VPN specification in the hose model includes [7]: (1) A subset of the nodes $P \subseteq V$ corresponding to the VPN endpoints, and (2) for each node $i \in P$, the

associated ingress and egress bandwidths B_i^{in} and B_i^{out}, respectively. Note that the terms "ingress" and "egress" are taken with respect to the VPN endpoints. This model can be enhanced to include a delay requirement in two ways: (1) Associate a delay requirement D_i with each node i, which specifies the maximum delay from this node to every other node in the VPN, or (2) Group applications that use the VPN into different delay classes characterized by their end-to-end delay requirements that must hold between every pair of end points. We adopt the latter approach in this paper.

The network identifies a set of delay classes; each class j is characterized by its end-to-end delay requirement D_j, $j = 1 \ldots L$. Without loss of generality, we order the L delay classes as: $D_1 < D_2 < \ldots < D_L$. In practice, these delay classes are obtained by measuring the characteristics of typical applications over the VPN. For each class j, with a delay constraint D_j, we need to find the corresponding ingress and egress bandwidth requirements $B_{i,j}^{in}$ and $B_{i,j}^{out}$ at each $i \in P$.

Therefore, the VPN specification in the enhanced hose model consists of the following three components: (1) A subset of the nodes $P \subseteq V$ corresponding to the VPN endpoints, (2) For each delay class j, the delay requirement D_j, which specifies the maximum end-to-end delay allowed between any pair of VPN endpoints, (3) For each $i \in P$ and each D_j, the associated ingress and egress bandwidths $B_{i,j}^{in}$ and $B_{i,j}^{out}$, respectively.

For clarity of presentation, the provisioning of one specific delay class, with its given delay requirement D and the associated ingress and egress bandwidths B_i^{in} and B_i^{out} for each VPN endpoint i, is discussed subsequently. The provisioning a VPN network can be viewed as provisioning each of the L delay classes.

3. Implementing the Enhanced Hose Model

Three general approaches for implementing the enhanced hose model are considered: (1) The "pipe mesh" approach, (2) The "multiple source-based trees" approach, and (3) The "shared tree" approach. The "pipe mesh" approach [4] implements the hoses with a mesh of pipes between the VPN endpoints. This can be viewed as the traditional pipe model and is included mainly for comparison purposes. The second approach builds a source-based tree to implement each hose of the VPN endpoints. A total of $|P|$ source-based trees are needed. The third approach uses a single shared tree for all the hoses in the VPN.

3.1. Pipe Mesh

In this approach a hose is implemented by a mesh of pipes between ingress and egress routers of the VPN. For a pipe from ingress i to egress j, $\min(B_i^{out}, B_j^{in})$ of bandwidth is reserved on each link along the path.

In order to minimize the reserved bandwidth for a pipe, we need to minimize the number of hops without violating the delay constraint D. We call this the Delay-Constrained Min-Hop Problem, which is a special case of the Delay-Constrained Least-Cost Problem when all the link costs are equal to 1. The problem is solvable by the Constrained Bellman-Ford (CBF) algorithm [13], which finds independent Min-Hop paths from a source to a set of destinations subject to delay constraints. The provisioning cost of the pipe mesh is the sum of bandwidth reservations of all links.

3.2. Multiple Source-Based Trees

In this approach, we use $|P|$ source-based trees to realize the hoses, one tree per hose. For a given source based tree T rooted at the VPN endpoint i, we denote by T_v the connected component of T containing node v when link (u, v) is deleted from the tree. In this case, the traffic passing through link (u, v) can only originate from i to the other endpoints in T_v. The traffic that i can send is bounded by B_i^{out}, 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) of T is given by $C_T(u, v) = min\{ B_i^{out}, \sum_{j \in P \cap T_v} B_j^{in} \}$. Therefore, the total bandwidth reserved for tree T is given by Eq. (1):

$$C_T = \sum_{(u,v) \in T} C_T(u,v) . \quad (1)$$

In the above equation only links directing away from 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 (v, u). Since we are interested in minimizing the total bandwidth reserved for tree T, the problem of computing the optimal source-based tree for endpoint i can be expressed as follows:

Optimal Delay-Constrained Source-Based Tree Problem: Given a set of VPN endpoints P with their associated ingress and egress bandwidths and the delay requirement D, compute a source-based tree T rooted at endpoint i whose leaves are the other VPN endpoints. The objective is to minimize C_T while satisfying the delay requirement, $\max_{j \in P \setminus i} delay(i, j) \le D$.

Theorem 1: The Optimal Delay-Constrained Source-Based Tree Problem is NP-hard. (See [14] for proof.) ■

We select the QDMR algorithm proposed in [8] to construct the source-based trees. QDMR is a fast and scalable heuristic for generating low-cost delay-bounded multicast trees. The total bandwidth reserved

using multiple source-based trees can be calculated by adding the bandwidth reserved for each tree.

3.3. Shared Tree

This approach implements the $|P|$ hoses with a single shared tree. The tree structure is used because it is scalable and simplifies routing and restoration. Furthermore, a shared tree allows the bandwidth reserved on a link to be shared by the traffic between the two sets of VPN endpoints connected by the link.

For a given shared tree T, we denote by T_u/T_v the connected component of T containing node u/v when link (u, v) is deleted from the tree. Note that the traffic passing through link (u, v) can only originate from the endpoints in T_u and terminate at the endpoints in T_v. The traffic that endpoints in T_u can send is bounded by $\sum_{j\in P\cap T_u} B_j^{out}$, and the traffic that T_v can receive cannot exceed $\sum_{j\in P\cap T_v} B_j^{in}$. Thus the bandwidth to be reserved on link (u, v) of T is given by $C_T(u, v) = min\{ \sum_{j\in P\cap T_u} B_j^{out} , \sum_{j\in P\cap T_v} B_j^{in} \}$. The total bandwidth reserved for tree T is therefore given by formula (2):

$$C_T = \sum_{(u,v)\in T} C_T(u,v) . \quad (2)$$

Note that unlike eq. (1), where only the links directing away from the root of the tree are counted, all links in the tree (in both directions) are considered in eq. (2). Since we are interested in minimizing the total bandwidth reserved for tree T, the problem of computing the optimal shared tree can be formulated as:

Optimal Delay-Constrained Shared Tree Problem: Given a set of VPN endpoints P with their associated ingress and egress bandwidths and the delay requirement D, compute a shared tree T connecting all the VPN endpoints with the objective of minimizing C_T while satisfying the delay requirement, $\max_{i,j\in P} delay(i, j) \le D$.

Theorem 2: The Optimal Delay-Constrained Shared Tree Problem is NP-hard (See [14] for proof) ■

Compared to the previous two approaches, the shared tree approach makes the best use of statistical multiplexing to reduce the provisioning cost. Only one tree is needed for the whole VPN with p hoses. The simulation results in Sec. 7.2 justify this, showing a significant reduction in provisioning cost. Furthermore, routing along the shared tree is simple and restoration of the tree structure is easy.

We develop heuristics for the Optimal Delay-Constrained Shared Tree Problem using a center based shared tree approach. The procedure can be divided into three phases as follows:

Phase 1: The graph is examined to identify a set of candidate centers satisfying the delay requirement. The set is called the Candidate Center Set (CCS).

Phase 2: The Candidate Center Set is pruned to reduce the computation overhead of the heuristic. Depending on the need to control overhead, the set can be unchanged, partially-pruned, or totally pruned. By "totally pruned", we mean the set is reduced to only one candidate center after pruning.

Phase 3: Trees that do not violate the delay requirement D are constructed for each of the nodes in the pruned Candidate Center Set. The tree with the minimum reserved bandwidth is chosen.

Three problems are of interest: (1). In Phase 1, how to make the shared tree support the most stringent delay requirements? (2). In Phase 2, how to reduce the computation overhead as far as possible by pruning without incurring additional provisioning cost? (3). In Phase 3, how to achieve high statistical multiplexing gain given a specific candidate center? These three problems are solved in sections 4, 5, and 6 respectively.

4. Meeting the Delay Requirement

In this section, we first propose two heuristics, namely the RC and DC heuristics to find all candidate centers satisfying the delay requirement. These two heuristics cannot guarantee a feasible tree will be found when the delay requirement is too stringent. We solve this by formulating the minimum diameter Steiner tree problem and propose a new algorithm MDStT.

4.1. The Radius Constrained (RC) Heuristic

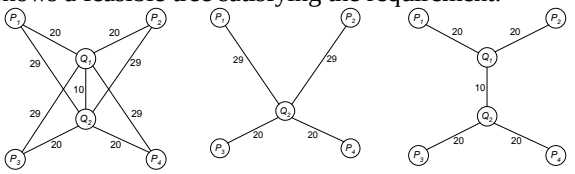
The radius r for node v is defined as the largest least-delay value from v to the VPN endpoints. The RC heuristic includes a node c in the candidate center set if its radius r satisfies $r \le D/2$. This guarantees that the maximum delay between any two VPN endpoints along the least-delay tree rooted at c cannot exceed twice the radius, i.e., D. Therefore, a shared Steiner tree is constructed to span the p endpoints with c as the center.

4.2. The Diameter Constrained (DC) Heuristic

The diameter d for node v is defined as the sum of the two largest least-delay values from v to the VPN endpoints. Note that this definition of the diameter for a node is different from the definition of the diameter for a tree in Section 4.3. The DC heuristic includes a node c in the candidate center set if its diameter d satisfies the delay requirement D ($d \le D$). If the condition is satisfied, the delay between any two VPN endpoints along the least-delay tree rooted at node c does not violate the delay requirement D.

4.3. Optimal Solution: The MDStT Algorithm

The above two heuristics, while fast and simple, cannot guarantee finding a feasible tree even if one exists. In the example given in Figure 1, the delay requirement among the VPN endpoints ($P_1 - P_4$) is 50. Nevertheless, the RC and DC heuristics are only able to support a delay requirement up to 58 with the candidate center located at Q_2 with a radius value of 29. This already violates the delay requirement. Figure 1(c), shows a feasible tree satisfying the requirement.



(a) Network topology (b) Shortest path tree by the DC heuristic (c) The feasible tree

Figure 1. Problem with the RC and DC heuristics. Only link delays are displayed.

We define the diameter of a Steiner tree connecting the VPN endpoints as the maximum delay between any two endpoints. Then the minimum delay requirement that can be supported by the shared tree can be found by finding the minimum diameter Steiner tree.

Mathematically, the minimum diameter Steiner tree (MDStT) problem can be formulated as follows:

Minimum Diameter Steiner Tree problem: Let $G = (V, E)$ be an undirected graph, where V is the set of nodes, and E the set of links. A subset of nodes $P \subseteq V$ represents the set of target destinations. Also let $|V| = n$ and $|E| = m$ and $|P| = p$. Suppose each link $e \in E$ is associated with a nonnegative weight d_e. A Steiner tree is a subgraph $T(V', E')$ of G which is a tree and $P \subseteq V'$. The diameter $D(T)$ of T is defined as the longest of the shortest paths in T among all node pairs in P. The *Minimum Diameter Steiner Tree* (MDStT) Problem is to find a Steiner tree of G with the minimum diameter.

We first prove 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. Note that 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.

We let $A(G)$ denote the continuum set of points on the edges of G. For any point x in $A(G)$, which may or may not be a vertex of G, and a destination node $v \in P$, we let $d_G(x, v)$ denote the length of a shortest path in G between x and v. For each x in $A(G)$ define $F(x) = max_{v \in P} d_G(x, v)$. The *absolute subset 1-center* problem (AS1CP) in graph G is to find a location x which minimizes $F(x)$. A point x^* in $A(G)$ is an absolute subset 1-center for subset P in G if the function F attains its minimum at x^*.

Theorem 3: Let x* be an absolute subset 1-center of P in G and let T(x*) be a shortest path tree connecting x* to all nodes in P. Then T(x*) is a minimum diameter Steiner tree connecting all nodes in P. (See [14] for proof.)■

We develop our MDStT algorithm based on the algorithms for the absolute center problem [1]. The main idea 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 local center is defined as the point minimizing $F(x)$ among every possible point on the link. To find the local center for a specific link (A, B), the functions from any point x to VPN endpoints $F_i(x)$ are first computed. Then $F(x)$ can be obtained by taking the upper envelope of these $F_i(x)$, $i = 1, 2, \ldots, |P|$. The local center is easily identified as the point minimizing $F(x)$.

The MDStT algorithm finds the minimum diameter Steiner tree in a general graph, and hence supports the lowest delay requirement using a tree structure in the enhanced hose model. Given the distances from the nodes to all VPN endpoints, the RC and DC heuristics need only $O(|V|)$ time to identify the center, compared to $O(|E|\ p + |V|p\log p)$ of the optimal MDStT algorithm which is more expensive. The following theorem shows that the RC and the DC heuristics are 2-approximations of the optimal MDStT algorithm with respect to the minimum delay requirement supported.

Theorem 4: The diameters of the trees constructed by the RC heuristic and the DC heuristic are at most twice the diameter of the minimum diameter Steiner tree constructed by the MDStT algorithm. (See [14] for proof.)■

5. Lowering Provisioning Cost

Phase 1 finds a set of candidate centers satisfying the delay requirement D. This set consists of network nodes if RC and DC heuristics are used, but will contain only one virtual node if the MDStT algorithm is used, which is the absolute center identified for the minimum diameter Steiner tree. The objective of Phase 2 is to build a low-provisioning-cost tree rooted at a specific candidate center. Four types of trees are considered in this work.

5.1. Least Delay Tree

The Least-delay (LD) Tree uses the least-delay tree rooted at the center to connect the VPN endpoints. The

delay values from the center to the VPN endpoints are minimized in this case. Because of the way the candidate center is chosen, the delay requirement D will always be satisfied. However, no effort is made to minimize the total bandwidth reserved.

5.2. QDMR (QoS Dependent Multicast Routing) Tree

The QDMR tree [8] tries to minimize the number of hops in the constructed tree in order to reduce the bandwidth reserved. The delay constraint from the center to any of the VPN endpoints is set to $D/2$.

5.3. LCLD (Least Cost Least Delay) Tree

The third variation is to construct a LCLD tree [11]. Given a center c, each VPN endpoint v tries to connect to the center using its LCLD path. The LCLD path from VPN endpoint v to the center c goes along the Min-Hop path as long as the delay bound $D_{shared} = D/2$ is not violated. It then switches to the Least-Delay path from the current node to the center when going further along the Min-Hop path would violate the constraint. Switching to the least-delay path from the current node will satisfy the delay constraint and would not cause backtracking.

Theorem 5: The shared tree constructed using the LCLD approach is loop free and satisfies the delay requirement D. (See [14] for proof.) ■

5.4. BFS (Breadth-First-Search) Tree

The LCLD tree can reduce the provisioning cost as shown by simulation results in Sec. 8. However, going along the min-hop path and switching to the least-delay path may not always be the best choice. The Breadth-First-Search (BFS) tree first finds independent delay-constrained min-hop (DCMH) paths from the center to the endpoints using the CBF algorithm [13]. Then these DCMH paths are merged to form the final BFS tree.

A by-product of this merging procedure is that there may be loops resulting from simple union of these DCMH paths. Therefore, we need to form an induced graph by the union of the paths, and perform a final round of the shortest-path algorithm from the center to the VPN endpoints in the induced graph. This procedure would eliminate all loops and leave only a Steiner tree spanning the VPN endpoints.

6. Reducing Computation Overhead

In large networks with relatively loose delay requirements, the number of candidate centers obtained in Phase 1 may be quite large. This will cause high computation overhead in Phase 3. The objective of Phase 2 is to reduce the number of candidate centers in the CCS, thereby reducing the computation overhead in Phase 3. Let us denote the minimum radius of the centers in the CCS by r_{min}. The radius of each candidate center v in the original CCS is checked to see if $|r-r_{min}| \le \delta r_{min}$, where δ is a predefined threshold. Candidate centers failing to satisfy the condition are deleted from the set. If δ is set to the extreme value 0, the CCS will be pruned to only one candidate center with the minimum radius. If computation overhead is not a major concern, or the number of centers obtained in Phase 1 is small, Phase 2 can be omitted and all nodes in the original candidate center set are used in Phase 3.

7. Simulation Results

We conducted a number of simulation experiments to measure the performance of the three approaches described in Section 3 and the algorithms proposed for the shared tree approach in Sections 4, 5, and 6. The results show that the shared tree approach is able to support a given delay requirement with a scalable tree structure at lower reserved bandwidth compared to the other two approaches. Given the advantages of the shared tree approach, we further study the performance of its various implementation alternatives.

7.1. Network Topologies

Two sets of topologies were used in our simulations. The first set is taken from the Rocketfuel project [10]. Among all the topologies, we selected four tier-1 ISP topologies as listed in Table 1 below. They represent real-world toplogies. The link delays of these topologies were computed based on their geographical distances. The setting of delay values is reasonable since transmission delay and queueing delay values are very small in these ISP backbone networks.

Table 1. Rocketfuel ISP topologies used in the simulations.

	Name	Tier	Dominant Presence	Degree	# of nodes
701	UUNet	1	US	2569	83
209	Qwest	1	US	887	58
1239	Sprint	1	US	1735	52
7018	ATT	1	US	1490	115

The second set was randomly generated using the Waxman Model [12]. Since we can easily control the size of the topologies, we use them to study the effect of network size on algorithm performance. In this model, the nodes are placed on a 3000×2400 Km^2 plane, roughly the size of the USA. The probability for two nodes to be connected by a link decreases exponentially with the Euclidean distance between them according to the following probability function:

$$P_e(u,v) = \rho \exp[-l(u,v)/(L\theta)] \quad (3)$$

where L is the maximum distance between any two nodes in the network and $l(u, v)$ is the distance between u and v. The parameter θ controls the ratio of short links to long links, while the parameter ρ controls the average node degree of the network. A large value of θ increases the number of long links, and a large value of ρ results in a large average node degree. In the experiments, θ and ρ were set at 0.15 and 2.2 respectively. These values were selected to obtain random networks which closely resemble real networks. Like the Rocketfuel topologies, the link delay values of the random networks were calculated according to their geographic distances. Link capacities were randomly chosen from one of the three: OC3, OC12, and OC48.

For both sets of topologies, the VPN endpoints were randomly selected from the network nodes. The number of VPN endpoints was set to be 10% of the total number of nodes in the network unless explicitly specified. The bandwidth requirement of each VPN endpoint was uniformly chosen between 2 and 100 Mbps. A parameter r is associated with each endpoint, representing the ratio between the ingress and egress bandwidth requirements. This asymmetry ratio varies from 1 to 256 in our simulation experiments. The delay requirement of the endpoints was generated uniformly between 20*ms* and 100*ms*. Each simulation result given below is the average of 10 rounds of experiments.

7.2. Efficiency of the Shared Tree Approach

The performance of the pipe mesh approach, the multiple source-based tree approach, and the shared tree approach were compared. Moreover, the effect of varying the network size and the number of VPN endpoints on performance was also investigated. In our study, the provisioning cost (the total bandwidth reserved) and the minimum delay requirement that can be supported were used as performance indices. The second metric is of interest because it describes the ability of meeting stringent delay requirements.

Minimum delay requirement supported

Figure 2 shows the minimum delay requirement that can be supported using each of the three approaches. We used both rocketfuel and random topologies. The name of the random topologies indicates the number of nodes in the network (e.g., "ran100" is a random topology with 100 nodes). The number of VPN endpoints in the networks is fixed at 10% of the total number of nodes. The first three bars are the three variations of the shared tree approach, with different methods of forming the candidate center set in phase 1. The last bar shows the performance of the pipe-mesh approach and the multiple source-based trees approach.

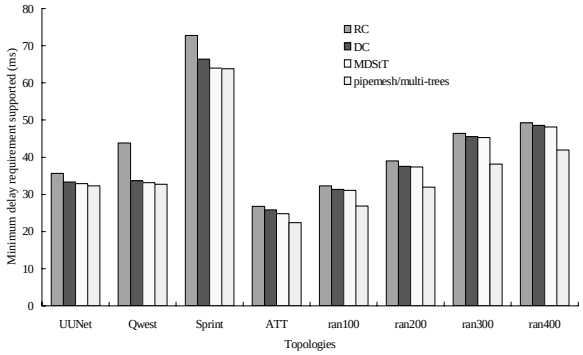


Figure 2. Minimum delay requirements supported by the approaches.

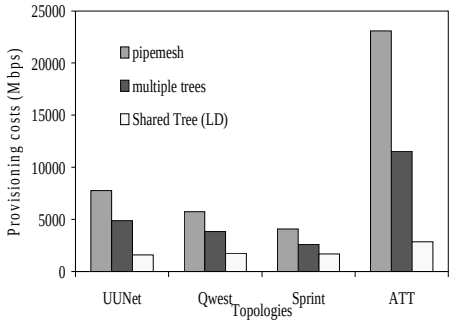
From the figure, the following observations can be made. First, the pipe-mesh approach and the multiple source-based trees approach perform identically with respect to supporting the minimum delay requirement. This is because when the delay requirement is stringent, the source-based trees are identical to the least-delay trees. These two approaches support smaller delay requirements than the shared tree approach. Second, the minimum delay requirement supported by the shared tree approach is higher than the other two approaches, although the difference is not large, especially for real-world topologies.

Total bandwidth reserved

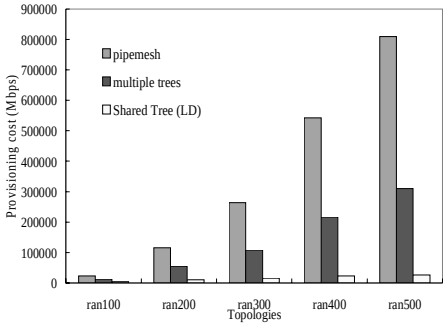
Figure 3 shows the provisioning costs of the three approaches. The network parameter settings were the same as before. For each network configuration, the delay requirement is chosen randomly from [20, 100*ms*] as long as all three approaches can find feasible solutions. For the shared tree approach, we used the LD tree in phase 2. The shared tree approach exhibits better performance than the other two approaches, reducing the provisioning cost by a factor of 2 or more for a wide range of network parameters.

7.3. Meeting Delay Requirements

Figure 2 also shows the minimum delay requirement that can be supported using the three technologies in phase 1 of the shared tree approach. The MDStT algorithm supports the minimum delay requirement that can be supported by a tree structure. This provides a lower bound for the RC and DC heuristics. The performance of the two heuristics is also near-optimal although they are only 2-approximations in theory. In most cases, the DC heuristic performs close to the optimal MDStT algorithm. This suggests that the DC heuristic is a good choice in phase 1 since it is fast and yields near-optimal tree diameters.



(a)



(b)

Figure 3. The provisioning costs of the approaches.

7.4. Lowering Provisioning Cost

The performance of the four implementation techniques in Phase 3, i.e., LD tree, QDMR tree, LCLD tree, and BFS tree, is exhibited in Figure 4. For each network configuration, the minimum delay requirement D_{min} that the four heuristics can support was the same. The delay requirements had to be carefully selected to show the different performance of the heuristics. If it is too small, the latter three trees will be very similar to the LD tree. If it is too large, the delay constraint will be meaningless and the LCLD tree and the BFS tree will become the min-hop tree, which definitely reserves less bandwidth than the LD tree approach. In Figure 4, the delay requirement was set to $1.15 \times D_{min}$. The results show that using the LCLD tree and the BFS tree in Phase 3 needs less bandwidth reservation than using either the LD tree or the QDMR tree. The difference is small for rocketfuel topologies. This is because the sizes of these topologies are small, which leaves less room for LCLD and BFS trees to go along alternative paths other than least-delay paths. The same is true with the randomly generated graph "ran100". When the network size increases, we see obvious advantage of LCLD and BFS trees over LD and QDMR trees.

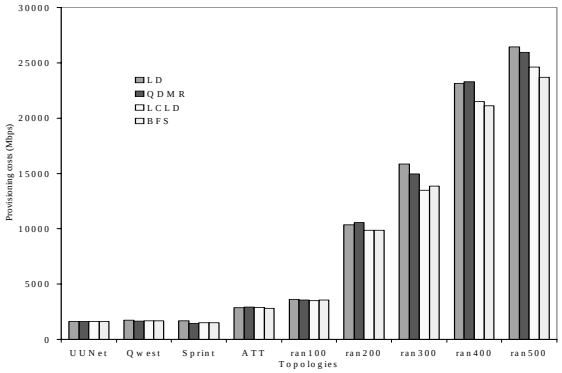


Figure 4. The provisioning costs of the four heuristics of the shared tree approach.

7.5. Reducing Computation Overhead

The pruning technique in Phase 2 is a compromise between performance and overhead. Table 2 depicts the relation between provisioning cost and the threshold δ used to prune the set. The table also shows the number of centers left in the CCS when the value of δ increases. We used one of the rocket fuel topologies, the Sprint topology in the simulation. Delay requirements are set at 1.5 of the minimum value supported by the shared tree approach. The DC heuristic is used in Phase 1 and the LCLD trees are used in Phase 2.

When δ was 0, only 1 center was left in the CCS. When δ was increased to 0.18, 12 out of 36 candidate centers were left in the pruned set. This means we only need to build 12 trees in Phase 3 instead of 36 if the set is not pruned, and these 12 trees already give optimal provisioning cost. Further increasing the value of δ and including more nodes in the set did not improve the cost. This leads to the conclusion that a small δ (< 0.2) can significantly reduce the computation overhead without sacrificing much provisioning cost. In most cases, the overhead would be around 1/3 of the original.

Table 2. Performance of the pruning technique

δ	0	0.1	0.18	0.2	0.4	0.8
Number of centers	1	11	12	14	23	36
Provisioning cost	1408	1098	1061	1061	1061	1061
Overhead over non-pruning	2.8%	31%	33%	39%	64%	100%
Excessive cost	32.7%	3.5%	0	0	0	0

7.6. Asymmetric Ingress-Egress Bandwidth Requirements

In real networks, the ingress and egress bandwidth requirements are not necessarily the same. Using the same experiment settings as [7], we model this asymmetry with an asymmetry ratio *r*, which defines the ratio between the ingress/egress bandwidth requirements. Figure 5, shows the provisioning cost of

the four heuristics with different tree types for the shared tree approach as the asymmetry ratio increased from 1 to 256. The number of nodes was 200 and the number of endpoints was fixed at 5. The provisioning costs of the heuristics increase with the asymmetry ratio while their relative ranking remains the same.

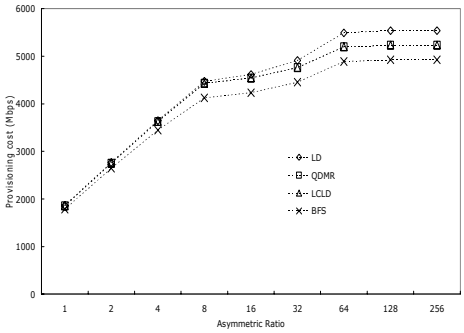


Figure 5. Effect of asymmetry ratio.

8. Conclusions

We extend the original hose model to incorporate delay requirements. Three approaches, namely the "Pipe Mesh" approach, the "Multiple Source-based Trees" approach, and the "Shared Tree" approach are proposed for provisioning VPNs in the enhanced model. For the "Shared Tree" approach, because of its scalability and ease of routing and restoration, several heuristics are developed using different techniques to implement the three phases in constructing the tree.

Simulation is used to evaluate the performance of the approaches. The "Pipe Mesh" approach satisfies the most stringent delay requirements, but requires more bandwidth to be reserved than the other approaches. The "Multiple Source-Based Trees" approach reserves less bandwidth than the "Pipe Mesh" approach since statistical multiplexing can be used within each of the trees, especially at the roots. The "Shared Tree" approach requires the least total bandwidth to be reserved, but the minimum delay requirement that can be supported is higher than the other two approaches.

The results suggest that the "Shared Tree" approach can be tuned to improve performance. In Phase 1, we obtain the minimum delay requirement that a tree can support by transforming it into the MDStT problem. The RC and DC heuristics are 2-approximations of MDStT algorithm and yield near-optimal tree diameters. In Phase 3, the LCLD and BFS trees require less total bandwidth to be reserved than the L trees and the QDMR trees. Finally, the pruning threshold δ in Phase 2 can be tuned to control the computation overhead while still yielding near-optimal provisioning cost.

References

[1] R. Cunnighame-Greene, The Absolute Center of a Graph, *Discrete Applied Mathematics*, 7 (1984), pp. 275-283.

[2] B. Davie, and Y. Rekhter. *MPLS Technology and Applications*. San Mateo, CA: Morgan Kaufmann, 2000.

[3] T. Erlebach, M. Ruegg, *Optimal Bandwidth Reservation in Hose-Model VPNs with Multi-Path Routing*, INFOCOM 2004.

[4] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe, *A Flexible Model for Resource Management in Virtual Private Networks*, In Proc. ACM SIGCOMM, 1999. pp. 95-108.

[5] A. Gupta, A. Kumar, J. Kleinberg, R. Rastogi, and B. Yener, *Provisioning a Virtual Private Network: A Network Design Problem for Multicommodity Flow*, In Proc. ACM STOC, 2001. pp. 389-398.

[6] A. Juttner, I. Szabo, and A. Szentesi, *On Bandwidth Efficiency of the Hose Resource Management Model in Virtual Private Networks*, In Proc. INFOCOM 2003.

[7] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, *Algorithms for Provisioning Virtual Private Networks in the Hose Model*, IEEE/ACM Trans. on Networking, vol. 10, issue 4, August 2002. pp. 565-578.

[8] I. Matta, and L. Guo, *QDMR: An Efficient QoS Dependent Multicast Routing Algorithm*, *In Journal of Communications and Networks*, Real-time Technology and Applications Symposium, 1999. pp. 213-222.

[9] P. P. Mishra, H. Saran, *Capacity Management and Routing Policies for Voice over IP Traffic*, *IEEE Network*, vol. 14, no. 2, pp. 20-27, March/April 2000. pp. 20-27.

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

[11] H. F. Salama, D. S. Reeves, and Y. Viniotis, *A Distributed Algorithm for Delay-Constrained Unicast Routing*, IEEE/ACM Trans. on Networking, vol. 8, issue 2, April 2000. pp. 239-250.

[12] B. M. Waxman, *Routing of Multipoint Connections*, IEEE Journal on Selected Areas in Communications, vol. 6, issue 9, December 1988. pp. 1617-1622.

[13] X. Yuan, *On the Extended Bellman-Ford Algorithm to Solve Two-constrained Quality of Service Routing Problems*, in ICCN'99, Oct. 1999.

[14] L. Zhang, J. Muppala and S. T. Chanson, *Provisioning Virtual Private Networks in the Hose Model with Delay Requirements*, Tech. Rep. HKUST-CS05-07, Dept. of Computer Science, HKUST, 2005.