Routing Algorithm for Provisioning Symmetric Virtual Private Networks in the Hose Model

Tat Wing Chim, King-Shan Lui, Kwan L. Yeung and Chi Ping Wong
Department of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam Road, Hong Kong
E-mail: {twchim, kslui, kyeung, h0148944}@eee.hku.hk

***Abstract*— A virtual private network (VPN) is a private data network where remote sites are connected over a shared provider network. In order to provide secure communications between customer sites, predetermined paths are used to forward data packets. To support quality of service (QoS), bandwidth has to be reserved on these paths. Then, finding appropriate paths in order to optimize the bandwidth used becomes an important problem. In this paper, we study the routing problem of VPNs under the hose model, where VPN endpoints specify the maximum bandwidth they need in sending and receiving data. Some previous works considered the problem under the assumption that all links have infinite capacities. We remove this constraint in our studies and develop enhancement to existing algorithms. Our simulation results show that our algorithm works very well in networks where link capacities are tight.**

I. Introduction

A. Overview of Virtual Private Network

A virtual private network (VPN) is a private data network that makes use of the public Internet [1] to maintain privacy through the use of IP tunneling technology [2] and network security protocols. VPNs can be regarded as a replacement of the expensive private leased lines. The main purpose of a VPN is to provide a company secure communication among multiple sites through the shared Internet. More detailed descriptions of VPNs can be found in [3] and [4].

To support a VPN, a service provider has to allocate predetermined paths to connect among customer sites. As customers may want to have bandwidth guaranteed, enough bandwidth has to be reserved on these paths. Therefore, finding appropriate paths and appropriate bandwidth reservation while minimizing the total bandwidth used becomes an important problem to service providers.

Two popular models for specifying customer bandwidth requirements have been proposed. They are known as the *pipe model* and the *hose model*. In the pipe model, customers are required to specify the bandwidth they need among each pair of VPN endpoints. In other words, a customer has to know the traffic between each pair of sites in advance and inform the service provider. This model is not very flexible since a customer may not be able to predict the communication patterns between VPN endpoints. Another disadvantage of this model is that the resources reserved for a pair of VPN endpoints cannot be allocated to other traffic flows. Thus, the utilization of Internet resources becomes very inefficient.

The hose model was proposed by Duffield et al. to solve the problems of the pipe model [5]. In the hose model, VPN customers just need to specify the incoming and outgoing traffic volume of each VPN endpoint (known as *ingress bandwidth* and *egress bandwidth*) instead of between every pair of VPN endpoints. The ingress bandwidth of an endpoint is the capacity required for aggregating the incoming traffic to the endpoint from other endpoints. The egress bandwidth is the capacity required for aggregating the outgoing traffic from the endpoint into the network. In other words, ingress bandwidth specifies the maximum amount of traffic an endpoint would receive per time unit while egress bandwidth specifies the maximum amount of traffic an endpoint would send out per time unit. Detailed examples showing the differences between the pipe model and the hose model can be found in [6].

The way to connect VPN endpoints is actually a routing problem. This routing problem is very important since one common goal of VPN operators is to minimize the total amount of bandwidth reserved for the network. Different structures have been proposed to connect VPN endpoints and they result in different amounts of bandwidth needed. [6] [7] [8] study using a tree to connect VPN endpoints. Other possible structures are general subgraph [7] and multi-path routing [9]. In multi-path routing, traffic between a pair of VPN endpoints is splitted among several paths. Among these three structures, tree is the most scalable since it is simple and require fewer labels in setting up the paths when using technology like MPLS. In terms of bandwidth, tree allows more sharing than a general subgraph. Although multi-path routing does require less bandwidth to support a VPN than a tree, it takes a lot of overhead in managing the splitting and the splitting itself is not an easy problem. Therefore, in this paper, we focus on using the tree structure to connect VPN endpoints.

B. Our Contributions

The problem of finding a tree to connect VPN endpoints in the hose-model has been studied by Kumar et al. in [6]. They developed algorithms trying to optimize the bandwidth needed on the tree under the assumption that all links have infinite capacities. To facilitate our discussion, we name their algorithm as the KRSY algorithm. In this paper, we enhance the KRSY algorithm to consider the case where links are of finite capacities. This capacitated version has been shown to be NP-hard [6] and to the best of our knowledge, there is no existing work solving this problem. We first make a trivial enhancement to include bandwidth constraints during the tree construction process. Then, we develop a tree resconstruction algorithm which aims at increasing the chance of finding a valid solution without violating the bandwidth restrictions. We study the performance of our algorithms using simulations and the results show that our algorithms are successful in enhancing the KRSY algorithm to work in bandwidth-constrained networks.

C. Organization of this Paper

This paper is organized as follows: a literature review is given in Section II. We describe the system model and the problem statement in Section III. We explain the KRSY algorithm in Section IV. Then we explain our algorithms in detail in Section V. Next we present our simulation results that evaluate the performance of our algorithms in Section VI. We conclude this paper in Section VII.

II. Related Works

The use of hose model for VPNs was firstly suggested by Duffield et al. in [5]. Jüttner et al. compared the bandwidth efficiency of the hose model and the pipe model in [10] assuming a range of network sizes and topologies with a linear programming based formulation. They concluded that hose model performs better in reducing blocking probability, decreasing traffic loss, and ease of implementation.

Based on the hose model, Gupta et al. studied the VPN provisioning problem under different scenarios: symmetric vs. assymetric ingress and egress bandwidths, as well as using a tree vs. using a graph to connect VPN endpoints [7]. They showed that the problem is NP-hard for assymetric ingress and egress bandwidths when a tree is used. On the other hand, the problem of finding an optimal tree becomes polynomial time solvable when each VPN endpoint has symmetric ingress and egress bandwidths while links are of infinite capacities. Kumar et al. [6] considered using a tree to connect VPN endpoints and provided a solution to solve the symmetric ingress and egress VPN provisioning problem. They also gave a 10-approximate algorithm for the NP-hard problem, where ingress and egress are different [6]. Link capacities were assumed to be infinity in this work. Gupta et al. then revisited this NP-hard problem in [8] and presented a 5.55-approximation algorithm to solve it. In a recent work [9], Erlebach et al. extended [6] further and presented an optimal polynomial-time algorithm for computing a bandwidth reservation scheme of minimum cost using multi-path routing under the assumption that link capacities are finite.

Italiano et al. studied the problem of fast recovery in a hose and single link failure model in [11]. They aimed at designing an optimal restoration algorithm to minimize the total bandwidth reserved on the backup edges.

III. System Model and Problem Statement

A. System Model

We adopt some of the notations developed in [6]. A network is modelled as a graph $G = (V, E)$, where V is the set of nodes and E is the set of *bidirectional* links among the nodes in V. (i, j) and (j, i) are considered as two distinct links. Each link (i, j) is associated with capacity L_{ij}. It is possible that $L_{ij} \neq L_{ji}$.

In the hose model, each VPN specification consists of a set of VPN endpoints $P \subseteq V$ and the ingress and egress bandwidths of each of the VPN endpoints. Ingress bandwidth is the maximum amount of traffic a VPN endpoint would receive, while egress bandwidth is the maximum amount of traffic the VPN endpoint would send. For a node $i \in P$, the hose ingress and egress bandwidths are both B_i, since we consider the case of symmetric ingress and egress bandwidths only.

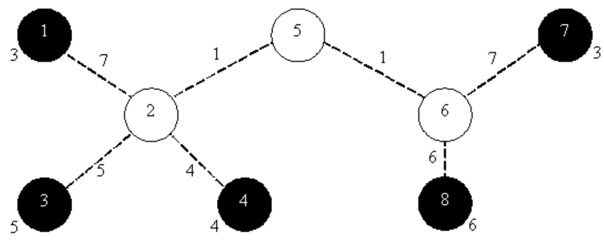

Fig. 1. An Example showing the Use of Some Notations

We use a tree to connect VPN endpoints. Formally, a tree $T = (V_T, E_T)$ is a subgraph of G where $P \subseteq V_T \subseteq V$ and $E_T \subseteq E$. Enough bandwidth has to be reserved on the links of the tree to support the VPN. To facilitate our discussion, we define $T_i^{(i,j)}$ and $T_j^{(i,j)}$, meaning the connected components of T containing nodes i and j respectively after removing (i, j) from T. Refer to the tree in Figure 1 in which dark nodes represent VPN endpoints and light nodes represent other network nodes, $T_2^{(2,5)}$ stands for the connected component consists of nodes 1 to 4 while $T_5^{(2,5)}$ stands for the connected component that is made up of nodes 5 to 8. We denote the set of VPN endpoints on $T_i^{(i,j)}$ and $T_j^{(i,j)}$ as $P_i^{(i,j)}$ and $P_j^{(i,j)}$, respectively. For example, in the tree in Figure 1, $P_2^{(2,5)} = \{1, 3, 4\}$ and $P_5^{(2,5)} = \{7, 8\}$.

Notation	Meaning
$G = (V, E)$	Graph with nodes V and edges E
L_{ij}	Capacity of link (i, j) (in the direction from node i to j)
P	Set of VPN endpoints
B_i	Ingress or egress bandwidth for node i
$T = (V_T, E_T)$	Generic notation for VPN tree
T_v	VPN tree rooted at node v
$T_i^{(i,j)}$	Component of tree T containing i when link (i, j) is removed from T
$P_i^{(i,j)}$	VPN endpoints contained in $T_i^{(i,j)}$
$C_T(i, j)$	Bandwidth reserved on link (i, j) of tree T
C_T	Sum of bandwidths reserved on all links of tree T
$H_T(i, j)$	Utilization on link (i, j) of tree T
H_T	Utilization on the most utilized link of tree T

TABLE I
Notations Used in this Paper

We now explain how much bandwidth is needed to be reserved on link (i, j) on T. Link (i, j) has to support the traffic going from $T_i^{(i,j)}$ to $T_j^{(i,j)}$. In other words, it should support the traffic from VPN endpoint a to VPN endpoint b for each $a \in P_i^{(i,j)}$ and for each $b \in P_j^{(i,j)}$. The maximum amount of traffic going out from $T_i^{(i,j)}$ is $\sum_{a \in P_i^{(i,j)}} B_a$. The maximum amount of traffic going to $T_j^{(i,j)}$ is $\sum_{b \in P_j^{(i,j)}} B_b$. Therefore, the maximum amount of traffic that would go through link (i, j) is $\min\{\sum_{a \in P_i^{(i,j)}} B_a, \sum_{b \in P_j^{(i,j)}} B_b\}$ and this is the bandwidth needed to be reserved on (i, j). We denote this value as $C_T(i, j)$. As the ingress and egress bandwidths are symmetric, $C_T(i, j) = C_T(j, i)$. Refer to the tree in Figure 1 in which the number next to each VPN endpoint represents its ingress or egress bandwidth (i.e. $B_1 = 3, B_3 = 5, B_4 = 4, B_7 = 3, B_8 = 6$), to find $C_T(5, 6) = C_T(6, 5)$, we first remove the edge $(5, 6)$ from the tree and then compare the following two values: sum of ingress or egress bandwidths of VPN endpoints on the left of node 5 (i.e. $3 + 5 + 4 = 12$) and sum of ingress or egress bandwidths of VPN endpoints on the right of node 6 (i.e. $3 + 6 = 9$) and pick up the smaller of the two (i.e. 9). The total bandwidth needed for T, namely C_T, is $\sum_{(i,j) \in E_T} C_T(i, j)$. We define the utilization of link (i, j), $H_T(i, j)$, to be $\frac{C_T(i,j)}{L_{ij}}$. Refer to the tree in Figure 1 where all edges are of capacity 10, $H_T(5, 6) = 9/10 = 0.9$. We further define H_T to be the utilization of the most utilized link on the tree, that is, H_T $=\max\{H_T(i, j)|(i, j) \in E_T\}$.

Table I summarizes the notations used in this paper.

B. Problem Statement

We now formally define the VPN routing problem. Given a graph G, a set of VPN endpoints P, and B_i for each $i \in P$. compute a VPN tree T that connects all nodes in P with the following properties:

- $H_T(i, j) \leq 1 \; \forall (i, j) \in E$.
- C_T is minimum among all possible trees.

The first property simply says that the capacity constraint should not be violated. This property makes the problem NP-hard [6]. The second property means that we would like to find an optimal tree that requires the least amount of bandwidth among all possible trees.

IV. KRSY Algorithm

In the KRSY Algorithm (for symmetric ingress and egress bandwidths case), all links in a given network topology are assumed to have infinite capacities. For each node v in the network, a breadth-first search [12] is carried out. Starting from node v, the search systematically explores edges in E to ”discover” every reachable VPN endpoint. The edges explored collectively form a tree and node v is known to be the root of that tree. Let us denote the tree as T_v. Leaves that do not correspond to VPN endpoints are then pruned from T_v. The method described in Section III-A is then used to find the bandwidth required on each link as well as for the whole tree. Among all breadth-first search trees considered, the one that requires the minimum amount of bandwidth is selected by the KRSY Algorithm.

Figure 2 shows a simple network topology for illustrating how the KRSY Algorithm works. In the figure, dark nodes represent VPN endpoints while light nodes represent other network nodes. The number next to each node represents its ingress or egress capacity which are assumed to be equal in this paper. The number next to each link represent its capacity. Among the breadth-first trees rooted at different nodes, the one rooted at node 9 has the minimum tree cost 82 and so is selected. Figure 3 shows that optimal tree computed using the KRSY Algorithm with the bandwidth needed on each edge. It can be observed that the tree is in fact not feasible under the bandwidth constraints specified in Figure 2. 15 units of bandwidth is needed on edge $(9, 1)$ while the edge only has a capacity of 12 units. This happens because the KRSY algorithm does not consider bandwidth constraints during the execution and results in an *infeasible tree*.

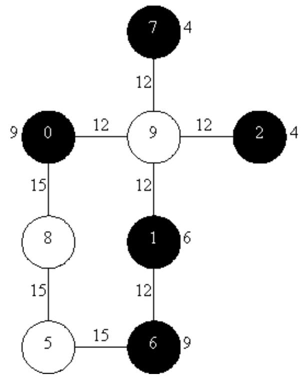

Fig. 2. A simple network showing how the KRSY Algorithm works

V. Enhanced Algorithms

In this section, we present two enhancements to the KRSY algorithm to find a feasible tree to connect VPN endpoints under bandwidth constraints. The first enhancement is a trivial one that makes sure KRSY selects a feasible tree by introducing a capacity check in the original KRSY algorithm. This enhancement is not sufficient since there are cases that a feasible tree cannot be found even such one exists. Therefore, we develop another enhancement which explores more trees and try to find a feasible one.

A. KRSY with Bandwidth constraints (KB Algorithm)

The first enhancement, KB, enhances the KRSY Algorithm to consider link capacities during the tree construction process. The tree

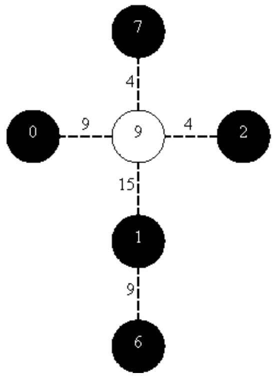

Fig. 3. Optimal tree constructed by the KRSY Algorithm

exploration process is the same, that is, for each node in the network, a breadth-first search tree rooted at that node is found. For each tree, apart from finding the bandwidth needed, we also check whether the tree is feasible, meaning whether the bandwidth required on any edge is over the capacity of that edge. Infeasible tree is not selected and KB returns the minimum cost feasible tree if there exists one or more feasible trees among the trees explored. Figure 4 shows the procedure KB-Algorithm, which is based on the KRSY algorithm. Lines 11 to 13 are introduced by us for testing bandwidth constraints. It is not difficult to see that KB has the same running time complexity as KRSY.

```
KB-ALGORITHM($G, P$)
 1  $T_{opt} \leftarrow 0$
 2  for each $v \in G$
 3  do $T_v \leftarrow v$
 4      $openQ \leftarrow \{v\}$
 5      while $openQ \neq \emptyset$
 6      do dequeue first node $u$ from $openQ$
 7          for each $(u, w) \in G$ such that $w \notin T_v$
 8          do add edge $(u, w)$ to $T_v$
 9              append node $w$ to end of $openQ$
10       prune leaves of $T_v$ that do not correspond to $P$
11       compute $H_{T_v}$
12      if $H_{T_v} \leq 1$ and $C_{T_v} < C_{T_{opt}}$
13        then $T_{opt} \leftarrow T_v$
14  return $T_{opt}$
```

Fig. 4. KB Algorithm for computing optimal tree with bandwidth constraints

Refer to the example in Figure 2, the KB Algorithm does not return any feasible tree since all the breadth-first search trees considered are not feasible. To find a feasible tree, we need to explore more trees and this leads to our second enhancement.

B. KRSY with Bandwidth constraints and tree Reconstruction (KBR)

We further enhance our KB Algorithm to include a tree reconstruction procedure so as to increase the chance of finding a feasible tree. We denote this further enhanced version as the KBR algorithm (KRSY with Bandwidth constraints and tree Reconstruction). We still find a breadth-first search tree rooted at each node in the network. In KB, if a tree is infeasible, we will not consider this tree anymore. In KBR, we try to construct a new feasible tree based on this infeasible one. A tree is infeasible because one or more edges cannot provide enough bandwidth. These edges are called *infeasible edges*. The idea of KBR is to use a *feasible path* to replace each infeasible edge. For an infeasible edge (i, j) on infeasible tree T, a path from a to b can replace (i, j) if it has the following properties:

1) $a \in T_i^{(i,j)}$ and $b \in T_j^{(i,j)}$ [1]
2) the path does not pass through any node on the tree T, except a and b
3) capacity on the path is larger than $C_T(i, j)$

The first and the second properties ensure the structure after replacing edge (i, j) by the path from a to b is still a tree. As illustrated in Figure 5, a tree can be formed by $T_i^{(i,j)}$, $T_j^{(i,j)}$, and the path from a to b. The third property makes sure the path has enough bandwidth to support the VPN. After replacing each infeasible edge by a feasble path, a new tree is formed and this new tree becomes feasible.

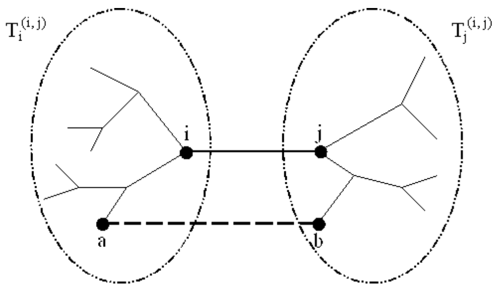

Fig. 5. Replacing an infeasible edge by a feasible path

It is possible to have more than one feasible path to replace an infeasible edge. In order to reduce the bandwidth needed, we select the shortest path among all possible choices. Given an infeasible tree $T = (V_T, E_T)$, the detailed process in finding a feasible path in $G = (V, E)$ to replace an infeasible edge (i, j) is as follows:

1) $PathSelected = nil$
2) prune the following edges and nodes from G to form G'
   - all nodes in V_T and their edges from G
   - all edges in E of capacity smaller than $C_T(i, j)$
3) for each $a \in T_i^{(i,j)}$ and for each $b \in T_j^{(i,j)}$
   a) put the following edges in G' to form G''
      - $(a, k) \in E$ if $L_{ak} \geq C_T(i, j)$ and $k \neq j$
      - $(k, b) \in E$ if $L_{kb} \geq C_T(i, j)$ and $k \neq i$
   b) find a shortest path, sp, from a to b in G''
   c) if sp is shorter than $PathSelected$, $PathSelected = sp$

There are at most $O(|V_T|^2)$ pairs between $T_i^{(i,j)}$ and $T_j^{(i,j)}$. We need to execute the Dijkstra's algorithm for each pair and so the total complexity of this process is $O(|V_T|^2(|V|log|V| + |E|)) = O(|V|^3log|V| + |V|^2|E|))$.

Figure 6 shows the procedure KBR-ALGORITHM in detail. Statements from lines 11 to 21 are introduced by us while all others are directly adopted from the KRSY Algorithm.

Refer to the same network topology as shown in Figure 2, Figure 7 shows the feasible tree computed using the KBR Algorithm with both bandwidth constraints and tree reconstruction. When the algorithm goes through the tree T_9 rooted at node 9 containing tree edges $(0, 9)$, $(1, 6)$, $(1, 9)$, $(2, 9)$, $(6, 1)$, $(7, 9)$, $(9, 0)$, $(9, 1)$, $(9, 2)$ and $(9, 7)$, it finds that $H_{T_9} > 1$. Instead of dropping that tree immediately, it performs a tree reconstruction. In particular, it finds $H_{T_9}(1, 9) > 1$ and $H_{T_9}(9, 1) > 1$. For the overloaded edge $(1, 9)$, the algorithm first decomposes the tree T_9 into two tree components $T_{91}^{(1,9)}$ and $T_{99}^{(1,9)}$. The former contains nodes $0, 2, 7$ and 9 while the latter contains nodes 1 and 6. Then the algorithm computes a feasible shortest path $sp = \{1-6-5-8-0\}$, which is disjoint with both $T_{91}^{(1,9)}$ and $T_{99}^{(1,9)}$, to connect these two tree components. Next it removes the edge $(1, 9)$ from T_9 and includes the edges $(1, 6), (6, 5), (5, 8)$ and $(8, 0)$ in sp into T_9 instead. Similarly, for the overloaded edge $(9, 1)$, the algorithm replaces the edge $(9, 1)$ by the edges $(0, 8), (8, 5), (5, 6)$ and $(6, 1)$ in T_9. The cummulated bandwidth reserved is 134.

[1]Please refer to Table I for the definitions of notations

```
KBR-ALGORITHM($G, P$)
 1  $T_{opt} \leftarrow 0$
 2  for each $v \in G$
 3  do $T_v \leftarrow v$
 4     $openQ \leftarrow \{v\}$
 5     while $openQ \neq \emptyset$
 6     do dequeue first node $u$ from $openQ$
 7        for each $(u, w) \in G$ such that $w \notin T_v$
 8        do add edge $(u, w)$ to $T_v$
 9           append node $w$ to end of $openQ$
10     prune leaves of $T_v$ that do not correspond to $P$
11     compute $H_{T_v}$
12     if $H_{T_v} > 1$
13       then for each $(i, j) \in T_v$ such that $H_T(i, j) > 1$
14              do compute the shortest path $sp$
15                 connecting $T_i^{(i,j)}$ and $T_j^{(i,j)}$
16                 if $sp$ is found
17                   then remove $(i, j)$ from $T_v$
18                        add all edges in $sp$ to $T_v$
19                        compute $H_{T_v}$
20     if $H_{T_v} \leq 1$ and $C_{T_v} < C_{T_{opt}}$
21       then $T_{opt} \leftarrow T_v$
22  return $T_{opt}$
```

Fig. 6. KBR Algorithm for computing a tree with bandwidth constraints and tree reconstruction

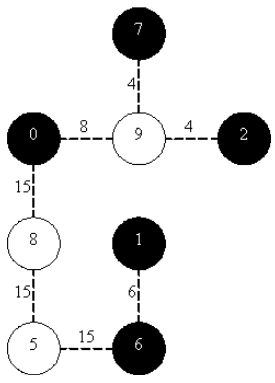

Fig. 7. Optimal tree constructed by the KBR Algorithm

VI. SIMULATION RESULTS

A. Simulation Models

To measure how effective our KBR Algorithm is, we conduct simulations. We generate two different sizes of topology for testing. For each size, we generate 1000 random topologies based on the BRITE topology generator [13]. For each topology, $|P|$ VPN endpoints are randomly picked up. The capacity of each link as well as the ingress and egress capacity of each VPN endpoint are uniformly generated with upper bounds $MaxL$ and $MaxB$ respectively. Table II summaries the parameters used in the simulation setup.

$\|V\|$	30	50
$\|P\|$	10	17
$MaxL$	20	35
$MaxB$	10	10

TABLE II
PARAMETERS USED IN SIMULATION SETUP

B. Simulation Results

Table III and Table IV show the performance of different algorithms for 30-node cases and 50-node cases, respectively. In the tables, the second column indicates the number of VPNs that the algorithm can find a feasible tree out of 1000 different VPN requests. The third column and the last column indicate the average bandwidth reserved and the average of average link utilization among all feasible trees, respectively.

From both tables, we can see that the KRSY algorithm is able to find feasible trees for only less than 40% (37.1% for 30-node topologies and 28.1% for 50-node topologies) of the testing cases. This is due to the fact that KRSY Algorithm does not have bandwidth constraints in mind during the tree-computation process. With bandwidth constraints taken into consideration, KB Algorithm can return valid solutions in more than 60% (68.5% for 30-node topologies and 65.6% for 50-node topologies) of the testing cases. Further with tree reconstruction, KBR Algorithm can return valid solutions in more than 70% (72% for 30-node topologies and 71.9% for 50-node topologies) of the testing cases. That means, the probability that our KBR Algorithm with both bandwidth constraints and tree reconstruction can return a valid solution is 120.7% higher than that of KRSY Algorithm. In that sense, our algorithm is far more efficient than KRSY Algorithm.

It is also worth noting that even though our algorithms enhance the success ratios in finding feasible trees, our algorithms do not require a lot of bandwidth to do that. The bandwidth needed is higher by 13.6% for 30-node topologies and 11.7% for 50-node topologies, which is not very serious compared with the increase in success ratios.

Algorithms	#Valid Solutions	Bandwidth Reserved	Avg. Link Util.
KRSY Algorithm	371	173.65	0.406
KB Algorithm	685	193.95	0.433
KBR Algorithm	720	197.28	0.438

TABLE III
SIMULATION RESULTS FOR 30-NODE CASES

Algorithms	#Valid Solutions	Bandwidth Reserved	Avg. Link Util.
KRSY Algorithm	281	353.50	0.336
KB Algorithm	656	385.79	0.349
KBR Algorithm	719	394.84	0.355

TABLE IV
SIMULATION RESULTS FOR 50-NODE CASES

VII. CONCLUSION

In this paper, we study the routing problem for provisioning VPNs under the hose model. The KRSY algorithm has recently been developed to solve the problem for networks of infinite capacities. The problem becomes NP-hard when edges in the network have limit in capacity. We enhance the KRSY algorithm and develop heuristics for this NP-hard problem. Simulation results show that our algorithms perform a lot better than the original KRSY algorithm in capacitated networks. Our algorithms work in symmetric VPNs only. In the future, we would like to study the problem in the context of assymetric VPNs.

REFERENCES

[1] H. Liang, O. Kabranov, D. Makrakis, and L. Orozco-Barbosa, "Minimal Cost Design of Virtual Private Networks," in *IEEE Proceedings of the CCECE '02*, 2002, pp. 1610 – 1615.
[2] W. Simpson, "IP in IP Tunneling," RFC 1853, Oct. 1995.
[3] R. Venkateswaran, "Virtual Private Networks," *IEEE Potentials*, pp. 11 – 15, Feb. 2001.
[4] B. Fox and B. Gleeson, "Virtual Private Networks Identifier," RFC 2685, Sept. 1999.
[5] N. G. Duffield P. Goyal A. Greenberg P. Mishra K. K. Ramakrishnan and J. E. van der Merwe, "Resource Management With Hoses: Point-to-Cloud Services for Virtual Private Networks," *IEEE/ACM Transactions on Networking*, pp. 679 – 692, Oct. 2002.
[6] A. Kumar, R. Rastogi, A. Silberschatz, and Bulent Yener, "Algorithms for Provisioning Virtual Private Networks in the Hose Model," *IEEE/ACM Transactions on Networking*, pp. 565 – 578, Aug. 2002.
[7] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, "Provisioning a Virtual Private Network: A Network Design Problem for Multicommodity Flow," in *ACM Proceedings of the STOC '01*, 2001, pp. 389 – 398.
[8] A. Gupta, A. Kumar, and T. Roughgarden, "Simpler and Better Approximation Algorithms for Network Design," in *ACM Proceedings of the STOC '03*, 2003, pp. 365 – 372.
[9] T. Erlebach and M. Rüegg, "Optimal Bandwidth Reservation in Hose-Model VPNs with Multi-Path Routing," in *IEEE Proceedings of the INFOCOM '04*, 2004, pp. 2275 – 2282.
[10] A. Jüttner, I. Szabó, and Á. Szentesi, "On Bandwidth Efficiency of the Hose Resource Management Model in Virtual Private Networks," in *IEEE Proceedings of the INFOCOM '03*, 2003, pp. 386 – 395.
[11] R. Rastogi G. F. Italiano and B. Yener, "Restoration Algorithms for Virtual Private Networks in the Hose Model," in *IEEE Proceedings of the INFOCOM '02*, 2002, pp. 131 – 139.
[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, "Introduction to Algorithms," book, 1990.
[13] "http://www.cs.bu.edu/brite/," .