Routing and Provisioning VPNs based on Hose Traffic Models and/or Constraints.

Gustavo de Veciana , Sangkyu Park, Aimin Sang and Steven Weber

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712
{gustavo}@ece.utexas.edu

Abstract

In this paper we investigate various questions associated with routing and provisioning virtual private networks (VPNs) based on hose traffic models or constraints, i.e., using aggregate (hose) traffic characterizations in/out of VPN end-points. This appears to be a promising simple approach to dealing with traffic engineering for VPNs. We evaluate the possible efficiencies derived from spatial multiplexing, i.e., over possibly varying routings of traffic loads consistent with hose specifications, and temporal multiplexing, i.e., statistical multiplexing of bursty loads, on shared VPN resources. We explore how these impact VPN routing and provisioning to meet 'coarse grained QoS targets' and how routing a sequence of VPNs on a shared network might be done to extend the 'life-time' of these decisions (i.e., so as to avoid re-routing) as the number of, and demands on, VPNs grow. Finally we discuss some natural extensions of the hose service model, specifically, the inclusion of multiple classes and cutset constraints, and how application layer multicasting mechanisms might balance loads over a VPN's hoses.

1 Introduction

The focus of this paper is on using 'hose models' as a means to describe traffic demands on a virtual private network (VPN) and how traffic engineering, i.e., routing and provisioning might be carried out in this context. By hose model, we refer to characterizations of the aggregate traffic in/out of a VPN's end-points versus using a full origin-destination traffic matrix. This reduction in the required amount of information introduces uncertainty on what are the precise traffic loads on the network, which for some cases may make traffic engineering inefficient, while for others, may enable higher efficiency and robustness to changing loads.

The 'hose model' was first introduced in the context of VPNs by [1], wherein four advantages were discussed: ease of specification, flexibility, multiplexing gain and characterization. Let us briefly summarize the arguments made therein. First, such a traffic characterization is simpler, in that one need only specify aggregates at ingress/egress points versus explicitly determining the demands between all origin destination pairs. This not only provides the customer with a natural service model but significantly reduces the amount of data required – data which in many cases one may be unable to measure/characterize reliably. Secondly, provisioning based on hose models/constraints enables some flexibility in that the actual traffic demands offered to the network may vary as long as they are consistent with the hose characterization. That is to say, specifying traffic based on the hose model, and in turn dimensioning resources to meet these demands, facilitates dimensioning resources over a set of possible traffic matrices. Given the dynamic characteristics of traffic, one may indeed wish to allocate resources so that a set of possible traffic loads (matrices) can be supported. Unfortunately this flexibility may come at a resource cost. The third observation is that provisioning resources based on hose specifications may allow additional multiplexing at the access points as well as within the network– this may alleviate the resource cost of the flexibility mentioned above. Finally, due to the smoothing resulting from aggregation at VPN endpoint's hoses, characterization of the traffic may be facilitated.

Routing structures for VPNs. This initial work, and subsequent research, e.g., [2, 3], have significantly advanced our understanding of hose based VPN routing and provisioning, as well as the associated restoration problem [4]. As first discussed in [1] one might consider routing VPN traffic in several ways:

Pipe Model: a single route is determined (based on shortest path or other criteria) between each pair of points.

Ingress (Egress) Tree Model: each ingress (or egress) of the VPN has a tree associated with it that carries traffic to (from) all egress (ingress) points. The tree might be routed based on shortest paths, or some other VPN or endpoint dependent metric. A VPN with n end points would be served by n possibly overlapping trees.

Shared Tree Model: traffic between VPN end points is routed along a *common* tree. Various criteria might be used to route such trees, e.g., minimize the overall provisioning cost, use Steiner tree based on an appropriate link metric, or attempt to minimize congestion on the network.

Mesh: an arbitrary routing of traffic, allowing the possibility of *splitting* traffic flows (e.g., for load balancing) among paths connecting VPN end points. If no splitting is permitted, a mesh reduces to a collection of individual pipes.

Depending on the routing structure, VPN dependent *routing state* may need to be maintained in the network, otherwise traffic would follow paths determined by the associated IP (or other standard) routing protocol. Indeed, explicit routing would be required to support routing structures specifically engineered for a given VPN. This can be achieved via source routing or by setting up and maintaining VPN dependent routing information in the network – e.g., maintaining appropriate MPLS labels. Since overheads associated with such explicit routing are a concern, e.g., maintenence of MPLS labels requires non-negligible signaling among routers, shared trees are deemed to be advantageous [5].

Provisioning VPNs. To determine the resources that need to be *provisioned* [1] for a given routing structure one needs to know whether VPN resource sharing state is available in the network and have a characterization/specification of the VPN's traffic. If the network supports VPN specific state for resource sharing, e.g., RSVP and MPLS labels, then pipes or trees that traverse a common link can share a capacity reservation on that link. Such sharing is particularly desirable when there are spatio-temporal variations in the VPN's traffic loads. The degree to which traffic load patterns are known, and the routing structure selected impact the extent to which such resource sharing can be exploited. For example, if a detailed traffic characterization is available (or measured) at each link, and sharing is supported, then resources might be appropriately provisioned to meet the aggregate VPN traffic demands on each link. By contrast if only hose descriptors are available, and no resource sharing is supported, provisioning is likely to be conservative and more costly. As will be discussed in the sequel, when only hose

[1]We do not distinguish here between explicit resource provisioning on behalf of a VPN and resources which are reserved through accounting mechanisms, and against which admission decisions would be made.

descriptors are available and resource sharing is supported for traffic on a shared VPN tree, an efficient provisioning of resources can be made, see [2, 3]. In summary, various provisioning problems can be considered, depending on the routing structure, whether resource sharing is supported, and what is known of the VPN's traffic loads.

On dynamic resizing. Our discussion assumes that one is provisioning and possibly routing a VPN based on a prior characterization or constraints on the offered load. Such a characterization may be challenging to obtain due to spatio-temporal variations in the loads. For this reason, dynamic resizing of (possibly shared) VPN resources based on real-time (predictive) traffic measurements at network resources has been proposed [1]. Note however, that the additional efficiencies of resizing can only be reaped if other traffic is present to take advantage of the time-varying capacity being freed up. In order to admit new VPNs on the network, and guarantee reasonable QoS one will still need to make offline allocations so as to ensure that surges in traffic can be met through resizing. Thus, the benefits of resizing depend on the bandwidth sharing/scheduling mechanisms being used across VPNs, the availability of 'best effort' traffic to use up the time varying capacity being made available, and the ability to perform aggressive VPN admission decisions. For example, if a work conserving service discipline is used to share resources across VPNs while ensuring a guaranteed rate to each, then resizing may not be worthwhile.

Summary of paper. In this paper we focus on the shared tree solution for VPNs, as it exhibits nice properties in terms of reducing the state in the network, facilitating re-routing upon failure, and enabling spatio-temporal multiplexing on network resources. In §2 we consider provisioning of shared trees based on an a-priori characterization of VPN traffic demands. Our goal is to investigate the spatio-temporal multiplexing gains enabled by shared trees versus the pipe model. Next, in §3 we reconsider the results of [3, 2] for routing a shared tree to minimize the provisioning cost while explicitly accounting for statistical multiplexing. We also consider the routing and provisioning of a series of shared trees to balance loads on a network. We argue that because shared trees would typically have high resource requirements on internal links, load balancing is likely to be more important than the minimizing of provisioning costs. Finally we conclude with §4 and §5 wherein we discuss some extensions to the hose based VPN service model, and discuss application level multicasting and load balancing over a VPN.

2 Provisioning VPN shared trees

We shall use the following notation to discuss provisioning and routing of shared VPN trees. The network is modeled as a graph $G(N,L)$ with a set of nodes N and a set of *bidirectional* links $L \subset N \times N$. A VPN corresponds to a set of end-points $E \subset N$ which are to be interconnected by a *shared tree* T on the graph – the tree is specified by a set of bidirectional links $T \subset L$. The removal of any link $(n,m) = l \in T$ partitions the tree into two sub-trees and thus the set of end-points into two sets L_l and R_l respectively. Finally we let $c_{n,m}$ and $c_{m,n}$ denote the capacity provisioned on link $l = (n,m)$ in the directions n to m and m to n respectively.

We let $D = (D_{i,j} \ : \ i, j \in E)$ denote the full (possibly unknown) traffic load matrix for the VPN, where $D_{i,j}$ denotes the (possibly random) demand from VPN end-point i to j. One can determine 'hose' vectors H^{in}, H^{out} characterizing the *aggregate* incoming and outgoing traffic loads at each of the VPN's end-points, as $H^{in} = Dv^T$ and $H^{out} = vD$, where v denotes a row vector with all 1's. Note that the demands D and associated hose vectors $H = (H^{in}, H^{out})$ may be either random or fixed depending on the type of VPN load characterization or constraints that are available.

Provisioning a VPN for fixed demands. Suppose VPN loads D correspond to a single fixed matrix d, e.g., corresponding to the *peak (or mean)* point-to-point loads on the network. These might be measured in Erlangs for voice traffic or Mbps for data traffic. In this case the associated ingress/egress hose vectors, denoted by h^{in}, h^{out}, capture the aggregate average or peak offered traffic loads at VPN end-points with the assumption of no losses/blocking. Note that while each matrix d results in a a unique pair of hose vectors $h = (h^{in}, h^{out})$, the converse is not true. Thus if a VPN tree is provisioned based on h alone, there would be sufficient resources to support a family of matrices corresponding to all traffic splittings consistent with hose specifications at the end-points. The basic idea for determining the capacity required on a given link of a shared VPN tree T is to find the worst case peak (or mean) offered load it could see. Recall that any link $(n,m) = l \in T$ partitions the end points into two sets L_l and R_l. Assuming there are no losses, the peak (or mean) offered load $\rho_{n,m}$ across the link, from n to m, is given by

$$\rho_{n,m}(h) = \min[\sum_{i \in L_l} h_i^{in}, \sum_{j \in R_l} h_j^{out}], \tag{1}$$

with a similar expression for the offered load $\rho_{m,n}$ in the reverse direction. Eq. 1 corresponds to the minimum of the maximum incoming load from end-points in L_l and the maximum load end-points in R_l could sink. Depending on the application, the capacity required at this link might be $\rho_{n,m}(h) + \rho_{m,n}(h)$, or might correspond to function of $c_{n,m}(\cdot)$ of the offered loads in each direction. These functions would typically be concave reflecting the economies of scale achieved by increasing the loads on a given resource. Thus the total provisioning cost $\phi_T(h)$ for a shared VPN tree T with hose load vectors h would be given by

$$\phi_T(h) = \text{ProvisioningCost}(T,h) = \sum_{l=(n,m) \in T} c_{n,m}(\rho_{n,m}(h)) + c_{m,n}(\rho_{m,n}(h)). \tag{2}$$

We note that for concave link provisioning functions the overall function ϕ_T is concave. If $c_{n,m}$ are linear then ϕ_T is radially homogenous, i.e., $\phi_T(\alpha h) = \alpha \phi_T(h)$ –scaling of the hose vector results in a scaling of the provisioning cost.

For example, consider a VPN intended to support IP telephony where the traffic loads at the hoses have been specified in Erlangs, and are necessarily symmetric, i.e., $h^{in} = h^{out}$. Quality of service on such a VPN might be specified in terms of end-to-end call blocking probability δ_b and voice quality δ_v for successful calls. Thus, to assure appropriate quality of service, sufficient resources would need to be allocated to meet the call blocking constraint and admission control would need to be conducted to ensure voice quality. To simplify the provisioning problem one might simply ensure that the call blocking probability on each link is acceptable. The function $c_{n,m}(\cdot)$ in this case could be computed based on the inverse Erlang function, an appropriate allowable call blocking probability per link, and an appropriate packet level multiplexing model for voice connections with silence suppression. This function would be concave and capture, on the one hand spatial multiplexing of resources on link (n,m), i.e., connections sharing that link but not the same path, and on the other hand temporal multiplexing of the resources by active connections.

Provisioning for a distribution of demands. Alternatively D may denote a random matrix capturing the spatio-temporal variations, i.e., burstiness, in the VPN's traffic load. In this case the hose vector H would be random, i.e., a statistical model for the aggregate traffic at the VPN's end-points. Provisioning a VPN to satisfy the traffic loads D translates to provisioning so as to meet a possibly large set of possible load scenarios, while provisioning based on H alone adds the additional uncertainty associated with the manner in which traffic might be split.

For example, the matrix D might consist of jointly Gaussian random variables with marginals $D_{i,j} \sim N(\mu_{i,j}, \sigma^2_{i,j})$ and covariance $K_{(i,j),(k,l)} = \mathrm{Cov}(D_{i,j}, D_{k,l})$ for all end-point pairs i, j and k, l. The hose vectors, $H = (H^{in}, H^{out})$ are simply linear combinations of elements of D and would thus also be jointly Gaussian. The variance and covariance terms for the joint distribution of the hose vectors, would reflect various types of correlations among bursty end-point demands. For example, positive correlations among demands may reflect bursty traffic associated with servers, and/or information distribution across VPN end-points. Negative correlations among demands at a given endpoint might reflect the role of capacity constraints outside the VPN, e.g., rate adaptation across such flows. Measuring the statistics of aggregate demands and possibly even the correlations among them seems reasonable, whence one might ask how these would impact the overall VPN provisioning costs.

Suppose then, that the loads on a VPN have been modeled as described above, where the random variables model bandwidth demands in Mbps and the objective is to ensure low probability of overflow, say δ, on all VPN resources. For purposes of evaluating the impact of such traffic fluctuations on the provisioning cost we will consider several approaches:

1. **customer pipe:** use pipe model (routed using shortest hop paths), provision to support multiplexing within pipes only, i.e., for all endpoint pairs (i, j), place sufficient capacity c along their routes such that $P(D_{i,j} > c) \leq \delta$.

2. **provider pipe:** same as above, but provision each link based on the aggregate load of all pipes traversing the link to meet the QoS constraint, i.e. allow multiplexing across pipes.

3. **shared tree with hose multiplexing:** route a shared tree using mean loads as hose descriptors and the algorithm proposed in [2], see §3. Then determine the peak capacity required at hoses to meet the QoS requirement, i.e., $h = (h^{in}, h^{out})$ such that $P(H^{in}_i > h^{in}_i) \leq \delta$ and $P(H^{out}_i > h^{out}_i) \leq \delta$, for all end-points, then provision the remaining links in the tree based on h using Eqs. 1 and 2, and $c_{n,m}(\rho) = \rho$.

4. **shared tree with VPN multiplexing:** use a shared tree as above, but provision all resources, i.e., both end points and VPN internal links based on a statistical characterization of loads offered to each link In this case the offered load at each link $l = (n, m) \in T$ in the tree is assumed to be random and given by $\rho_{n,m} = \min[\sum_{i \in L_l} H^{in}_i, \sum_{j \in R_l} H^{out}_j]$, and the capacity $c_{n,m}$ is computed so that $P(\rho_{n,m} > c_{n,m}) \leq \delta$. For simplicity, in our experiments we assumed the correlations between H^{in} and H^{out} were are small, so we assumed the vectors were independent, however in general such correlations should be accounted for more carefully.

The results are based on five 30-node topologies generated using 'pure random model' where the average degree at each node is 4.0 [10]. VPNs with a given number of end-points were selected based on selecting random locations on the graph. Provisioning costs were computed based on the four methods described above using a stringent QoS requirement of $\delta \approx 10^{-6}$. A distribution of load matrices associated with D was defined based on $D_{i,j}$ i.i.d. Gaussian with a common mean and variance. This induces a distribution on the hose vectors H. The graph on the left of Fig. 1 exhibits the reduction in provisioning costs relative to the customer pipe model as the mean and standard deviation of the loads are increased linearly, i.e., VPNs seeing increasingly bursty loads. All results are averages over multiple VPN instances. As can be seen, the provider pipe solution is best, yet in this case the full matrix D has used in provisioning. The shared tree solutions achieve fairly good performance, i.e., about 25-30% reduction in capacity. The graph on the right shows the reduction in provisioning cost relative

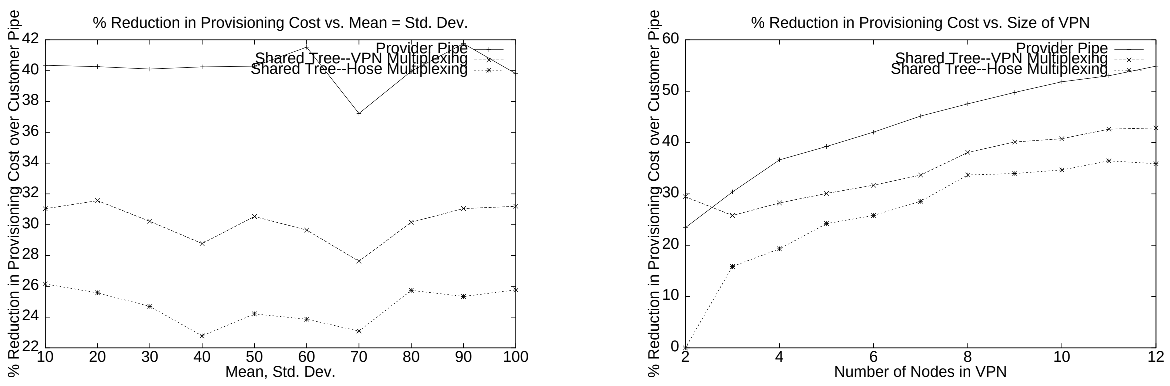

Figure 1: Reduction in provisioning costs for different provisioning styles relative to the customer pipe approach.

to the customer pipe model for VPNs including an increasing number of endpoints. As can be seen again the provider pipe model achieves the best performance. The shared tree options also achieve fairly consistent gains 20-40% and are fairly close to each other, suggesting that the temporal multiplexing at realized at the hoses, combined with spatial multiplexing in the network achieves a fairly good capacity savings.

3 Routing VPNs - minimizing cost or reducing congestion.

As mentioned earlier there are various goals one could set for routing shared trees. In this section we study two cases: routing to minimize provisioning cost with statistical multiplexing, e.g., as might be the case for our IP telephony example, and routing to balance loads on the network. Below we focus on the case where hose parameters are symmetric, which should be adequate if the asymmetry is not severe. Additional work is underway to consider this in more details. However, the results of [2, 3] for the asymmetric case when $c_{n,m}$ are linear suggest this is likely to be difficult.

Routing to minimize provisioning cost - symmetric case. We pose the problem of routing a shared tree so as to minimize provisioning cost, as in [2, 3], but with the addition of a concave provisioning function:

Problem 1 *Given a network $G(N,L)$, a set of end-points $E \subset N$, and deterministic and symmetric hose descriptors $h^{in} = h^{out}$, to find a tree T connecting the end-points E which minimizes the provisioning cost ProvisioningCost(T,h) (See Eq. 2), where $c_{n,m}(\cdot)$ are concave and symmetric.*

A solution to this problem can be sought as follows. For each node $r \in N$, determine the minimum cost routing for multiple commodities h_i^{in} $(\forall i \in E)$ to r the only sink. The cost in this case is sum of concave functions of the total flow carried on links towards the root. Note because the cost functions are concave the optimal routing to the sink is necessarily a tree, i.e., no splitting of flows is desirable, see e.g., [6]. Let T^r denote one such tree for the root r, and RootedCost(T^r) the minimal cost of multi-commodity problem associated with r. Among the $|N|$ trees obtained, one for each possible sink or root r, find one that has a minimal RootedCost. It can be shown that this tree also minimizes the ProvisioningCost and hence is an optimal shared tree for Problem 1. A proof of this based on the elegant argument in [2] (for the case where $c_{n,m}(\cdot)$ are linear functions) can be found in [7], wherein a simple greedy heuristic is proposed to obtain quick solution to the multi-commodity problem(s) mentioned above.

Our objective in minimum cost routing with concave link provisioning functions was to assess the extent to which routing of shared trees might impact and the economies of scale one can achieve. Due to space constraints we have not included these results but we summarize their qualitative characteristics. The cost savings achieved from a routing that is sensitive to statistical multiplexing gains were found to be highest for VPNs with 'moderate' numbers of end points [7]. In retrospect, this is not surprising, since VPNs with moderate numbers of nodes provide opportunities for careful routing to increase statistical multiplexing without significant overheads versus VPNs with small numbers of end-points. By contrast, for VPNs with large numbers of end-points, the gain was not significant relative to the large number of links needed to realize the connectivity.

Online routing for shared VPN trees. If the provisioning cost of a VPN translates to a service cost, then minimizing the provisioning cost may be desirable from the customer's point of view. However the provider may have other concerns. Indeed, a service provider will want to overlay various VPNs on his network. Thus a good VPN routing, from the provider's perspective, is one that allows as many of a sequence of VPN requests to be admitted. Moreover it would be preferable not to require re-routing, even if the customers subsequently increase their demands, i.e., renegotiate their hose parameters. As such one might ask how to route a sequence of shared-tree VPN requests so that the lifetime of these decisions is extended as long as possible.

Online routing of connection requests is hard but well studied, see e.g., [8]. To see how this would work for routing shared trees, we propose two simple heuristics and make no effort to prove (competitive) optimality. In particular for each request to route a new VPN, we determine the minimum RootedCost tree over all possible roots, but this time with link costs that are *convex* functions of the residual capacity on each link. Specifically we considered two such cost functions. The first was $1/r_{n,m}$ where $r_{n,m}$ denotes the capacity that has *not* been allocated on link (n,m). [2] Thus finding trees that minimize the overall cost of this form will tend to avoid links that are congested. The second link cost function we considered was $v_{n,m}^{\alpha}/r_{n,m}$ where $v_{n,m}$ denotes the number of VPNs that are currently flowing through link (n,m) and $\alpha > 0$. The rationale for using this cost function was that a link with a high residual bandwidth but which already has a large number of VPNs going through it should be deemed congested, particularly when VPN demands are allowed to grow. Our experiments bear this out.

We conducted a fairly extensive set of experiments to evaluate routing of multiple shared VPNs one of which is described below, see [7] for details. The results are based on the same topologies considered earlier. We generated a sequence of VPN requests, each corresponding to either a new VPN or growth in load in a current VPN. We varied γ the probability that a request corresponds to a new VPN from 0.1 to 0.9. For simplicity each VPN had five randomly selected end-points, and loads were normalized so that the load associated with a growth in demand in an old VPN was commensurate with the load associated with a new VPN. As our performance metric we used the lifetime of the decisions, which we define as the total number of requests that were served before a request had to be denied, or VPNs would need to be rerouted. Traffic on each new VPN was symmetric with a load uniformly distributed in [10,30] with mean 20 Mbps for each hose. Growth in demands for an old VPN, corresponded to incrementing all hoses by random amount which is uniformly distributed in [10,30] with mean 20 Mbps. The capacity of the network links was 1 Gbps. We compared several routing algorithms with respect to the average lifetime they achieved over 50 different request sequences with varying γ. The

[2]In fact our heuristic for finding minimum RootedCost trees, attempts to minimize this congestion cost after the new VPN is routed.

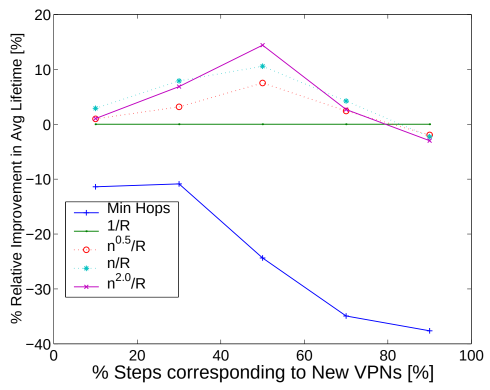

Figure 2: Lifetime referred to $1/r$ routing strategy vs percent new VPNs per step.

routing algorithms that were compared are the minimum provisioning cost routing discussed earlier, routing based on $1/r_{n,m}$ (denoted 1/R), and based on $v^{\alpha}_{n,m}/r_{n,m}$ with $\alpha = 0.5, 1$ and 2 as described above. Figure 2 shows the performance improvement or degradation versus the performance achieved by our baseline congestion dependent policy, in this case the $1/R$ policy. For this experiment, the minimum provisioning cost routing takes a penalty of 10-40% which is increasing with the fraction of new VPNs per step. The routing policies that are sensitive to the number of VPNs that have been routed on a given link proved to be effective only if there was a balance between new VPNs and VPN growth, i.e., $\gamma \approx 0.5$. When this was the case increasing α to 2 provided a 15% improvement over the $1/R$ scheme. In retrospect this makes sense, as routing in a manner that is sensitive to future growth in VPN loads should work best when a significant fraction of such growths, yet also a significant number of requests correspond to growths in loads and there are a significant number of routing decisions to make, i.e., $\gamma = 0.5$ is a balance between these two objectives.

4 Extending the hose service model - cutset and multi-class constraints.

In practice a vector h of hose descriptors might reflect contracts between the service provider and the VPN customer. A service contract specifies a distributed set of interfaces between the customer and the provider, i.e., how much traffic can enter and leave the network at each interface. Such contract would presumably be policed, and leading to excess traffic to be discarded or tagged for potential discarding elsewhere. The provider would presumably make a commitment to deliver all traffic conforming to the hose descriptors, and possibly delivering excess traffic if extra capacity is available in the network.

One problem with the shared tree solution is that it may require the allocation of a large amount of bandwidth in the 'center' of the tree. For example, in the worst case a link may need to be provisioned to meet about half the sum of the hose bandwidth agreements. For some VPNs this might be quite large, requiring the service provider to allocate a significant amount of bandwidth on a given resource, both to support the VPN and its restoration upon failure. A simple solution is to extend the service model to allow VPN customers or service providers to specify load constraints across specific (say transatlantic) links, each being taken as a 'virtual hose', and route the remaining traffic using the algorithms discussed previously.

Another useful extension to the VPN hose service model would be to allow specification of hose constraints for multiple traffic classes, i.e., a set of vectors $h^c, c \in C$. Once again, it may still be of interest to route the traffic on a common shared tree T. Provisioning T to satisfy multi-class constraints is a straightforward extension of Eq. 1 and 2, performed on a per class basis. However, determining a minimum cost tree to be *shared* by multiple classes appears to be difficult, yet it has some similarities with the asymmetric case considered in [2, 3]. One exception is the case where $c_{n,m}$ are linear and the hose vectors are colinear, in which case the minimum provisioning cost routing algorithm discussed above would be optimal for all classes.

5 Hose based VPN Service Model- A customer's perspective.

In addition to providing peer-to-peer connectivity for various types of traffic among end-points, VPNs might be used to multicast and cache large data files. If the VPN provider does not support network level multicasting, as is the case today, the VPN customer will need to resort to realizing multicasting himself. When this is the case, an attempt can be made to balance the load in a manner that is consistent with the endpoint hose parameters, and so as to minimize overall delays. Thus we conclude this paper by studying an example of how the customer might change the loads offered to a VPN given the hose parameters he has negotiated.

Consider a large file of size F to be distributed among a set of n end-points $E = \{1, 2, \ldots n\}$. Suppose without loss of generality that endpoint 1 is the originator of the file, and for simplicity that each endpoint $i \in E$ has a symmetric hose constraint $h_i = h_i^{in} = h_i^{out}$. A customer has several options to distribute such a file. The first is to transmit the file individually to each of the $n-1$ recipients. Since its hose constraint is h_1 in the best case, i.e., no other traffic in the network, this would requiring transmitting $(n-1)F$ units of data, for a duration of time $(n-1)F/h_1$ before *all* end points would approximately have received the file. In addition to the potentially high delay, this approach imposes a large and sustained load on endpoint 1's hose. Compare with the case in which the network supports network layer multicast. In this case only F data units would flow through the hose and the minimum time would be F/h_1. What other options does the application have?

One alterative is to resort to relaying the file or subfiles. The benefits here can be both in terms of distributing the ingress load across other endpoint hoses, which might be beneficial from a congestion or load balancing point of view, while also reducing delays. Consider the simple case where the file is forwarded from one endpoint to another, e.g., from 1 to 2, 2 to 3, until $n-1$ forwards the file to endpoint n. In this case each endpoint hose would see F units (both in and out), and the best case delay would be $F\sum_{i=1}^{n} 1/h_i$ if the file needs to be completely received prior to transmission. We shall refer to this as *forwarding*. Alternatively if data is *relayed* by end-points as it is received, the best case delay would be $F/\min[h_i | i = 1, \ldots n]$. Note that the ingress load to the VPN has been distributed equally among all hose end-points. Note also that if hose constraints are equal, then the delay achieved by relaying is essentially that of network level multicasting. If there are unequal hose constraints then the overall transfer delay is dictated by the bottleneck hose.

A further alternative is to consider splitting the file into $n-1$ sub files, each with a size proportional to the hose constraint associated with the endpoint that will serve as its relay, i.e., the size of the subfile sent to hoses $i = 2, \ldots n$ would be given by $f_i = \frac{h_i}{\sum_{j=2}^{n} hj} F$. Suppose further that the data originator initiates transmission of the subfiles to each endpoint concurrently and sends at rates proportional to their sizes, whence the reception rate to node i would be $r_i = \frac{h_i}{\sum_{j=2}^{n} hj} h_0$. In turn node i could relay the subfile to the remaining $n-2$ nodes through its hose with capacity h_i. Assuming full and fair use of the hose capacity the relay node i could achieve a rate of $h_i/(n-2)$ to each subsequent endpoint. If the condition

$$r_i = \frac{h_i}{\sum_{j=2}^{n} h_j} h_1 \leq \frac{h_i}{n-2} \quad \Rightarrow \quad h_1 \leq \frac{\sum_{j=2}^{n} h_j}{n-2}$$

is satisfied then the ideal overall delay would be F/h_1, i.e., that achieved by network level multicasting. In particular this would allow optimization of performance when there is heterogeneity in the hose parameters at each endpoint, yet one wishes to achieve the optimal performance allowable given the sender's endpoint. Note however that loads have been distributed in a different manner for this case, each relay node would see F units of data on its egress hose, and $2f_i$ units of data on its ingress hose – the congestion incurred by each endpoint is proportional to the hoses size, a reasonable criterion for balancing the loads on a shared VPN.

The key idea here is that in addition to the service provider's point of view on provisioning and routing VPNs we have discussed earlier, the customer may want to use VPN applications which are sensitive to either the loads on the VPN or the service level agreements made with the provider. Thus for example, in the case of application level multicasting of data files the customer may have a degree of latitude to balance the loads on the VPN, so as to minimize interference with ongoing peer-to-peer traffic.

References

[1] N.G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, and J. Merwe, *A Flexible Model for Resource Management in Virtual Private Networks,* Proceedings of ACM SIGCOMM 1999, September, 1999.

[2] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi, and B. Yener, *Provisioning a virtual private network: a network design problem for multicommodity flow,* Proceedings of ACM STOC 2001, pp. 389-398, July, 2001.

[3] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener, *Algorithms for Provisioning Virtual Private Networks in the Hose Model,* Proceedings of ACM SIGCOMM 2001, August, 2001.

[4] G. F. Italiano, R. Rastogi, B. Yener, *Restoration Algorithms for Virtual Private Networks in the Hose Model,* IEEE INFOCOM, 2002.

[5] Anupam Gupta, Amit Kumar, Rajeev Rastogi, *Traveling with a Pez Dispenser (Or, Routing Issues in MPLS),* Proc. 42nd IEEE Symposium on Foundations of Computer Science, 2001.

[6] D. Karger and M. Minkoff, *Building Steiner trees with incomplete global knowledge*, Proceedings of 41st IEEE FOCS, 2000.

[7] S. Park, *Traffic Engineering in Multi-service Networks : Routing, Flow Control and Provisioning Perspectives,* Dissertation, the University of Texas at Austin, 2002.

[8] S. A. Plotkin, *Competitive Routing of Virtual Circuits in ATM Networks,* IEEE Journal of Selected Areas in Communications, Vol 13, No 6, pp.1128-1136, 1995.

[9] R.L. Easton, P.T. Hutchinson, R.W. Moncello, R.W. Muise, *TASI-E communications system,* IEEE Trans. Commun., pp. 803-807, 1982.

[10] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, *How to Model an Internetwork,* IEEE Infocom, vol.2, pp. 594-602, 1996.