Multi-VPN Optimization for Scalable Routing via Relaying

MohammadHossein Bateni, Alexandre Gerber, MohammadTaghi Hajiaghayi, and Subhabrata Sen

***Abstract*—Enterprise networks are increasingly adopting Layer 3 Multiprotocol Label Switching (MPLS) Virtual Private Network (VPN) technology to connect geographically disparate locations. The any-to-any direct connectivity model of this technology involves a very high memory footprint and is causing associated routing tables in the service provider's routers to grow very large. The concept of *Relaying* was proposed earlier [9] to separately minimize the routing table memory footprint of individual VPNs, and involves selecting a small number of hub routers to maintain complete reachability information for that VPN, and enabling non-hub spoke routers with reduced routing tables to achieve any-to-any reachability by routing traffic via a hub.**

A large service provider network typically hosts many thousands of different VPNs. In this paper, we generalize Relaying to the multi-VPN environment, and consider new constraints on resources shared across VPNs, such as router uplink bandwidth and memory. The hub selection problem involves complex trade-offs along multiple dimensions including these shared resources, and the additional distance traversed by traffic. We formulate the hub selection as a constraint optimization problem and develop an algorithm with provable guarantees to solve this NP-complete problem. Evaluations using traces and configurations from a large provider and many real-world VPNs indicate that the resulting Relaying solution substantially reduces the total router memory requirement by 85% while smoothing out the utilization on each router and requiring only a small increase in the end-to-end path for the relayed traffic.

I. Introduction

Enterprise networks are increasingly adopting Layer 3 MPLS VPN technology, to connect geographically disparate locations. This technology offers direct any-to-any reachability, via a provider IP network, among different sites of an enterprise customer. However, this reachability model imposes a very high memory footprint (details in Section II) and is causing routing tables in provider routers to grow very large (e.g., some VPNs can contain more than 10,000 routes). Consequently, router memory availability has become a key bottleneck when provisioning customers on a Provider Edge (PE) router at the boundary of the provider's network.

To alleviate this bottleneck, the idea of Relaying [9] was introduced: it reduces the PE memory footprint of a VPN by having a small number of hub PE routers to maintain full reachability information, and enabling non-hub PEs (spoke PEs) to reach other routers by relaying through a hub (details in Section II). Intuitively, Relaying is motivated by the key observation that traffic matrices (i.e., matrices of traffic volumes between each pair of PEs) in VPNs are typically very sparse [12], [11]. This is a result of various factors such as the predominance of client-server applications and typical corporate structures commonly placing application servers at a few central locations. The sparse communication pattern implies we might optimize routing table sizes for the common case communications.

Selecting the hub routers for Relaying involves making complex tradeoffs along multiple dimensions. On the one hand, minimizing the number of hubs is desirable as it can reduce the VPN memory footprint as well as upfront installation and maintenance costs for hubs. On the other hand, traffic between two spoke PEs is rerouted along an indirect path via an intermediate hub PE, and thus potentially traverses longer path in the provider network. Such relayed traffic can (i) experience longer latencies, (ii) impose additional overhead on the provider network which has to carry the traffic over a longer distance, and (iii) cause additional load on the links towards and from the hub and on the hub itself. In addition, since a hub router is critical to maintaining reachability between its spoke PEs, Relaying needs to be resilient to common failure scenarios. The memory usage, increased latency and bandwidth overheads, additional loads on the hub and its links, and reliability requirements all impose different constraints on the Relaying problem, and a solution involves carefully navigating this multidimensional space for a "sweet spot" region.

The earlier effort [9] explored Relaying for a single VPN, and developed practical heuristics (without a provable worst case guarantee) for determining the hub selection for an individual VPN that minimized the total number of hubs, while ensuring that constraints on additional end-to-end latency were not violated. That work did not consider factors like the additional relay traffic load on a hub PE and its links, or constraints on the memory and bandwidth resources arising from the needs of other VPNs also served by the same provider network.

A service provider network typically hosts many thousands of different VPNs. In this paper, we generalize Relaying to the multi-VPN environment. Different VPNs can have very different characteristics in terms of size, traffic patterns, and routing table sizes. While the routing entries are specific to each VPN, the provider network and individual PEs are shared resources serving many different VPNs. Decisions made for one VPN can impact the fate of others in various ways., e.g.,

A short version of this work appears in [3].

M. Bateni is a graduate student in Department of Computer Science, Princeton University, Princeton, NJ 08540; Email: mbateni@cs.princeton.edu. This work was done while he was a summer intern in AT&T Labs.

A. Gerber, M. Hajiaghayi, and S. Sen are with AT&T Labs–Research, Florham Park, NJ 07932; Emails: {gerber,hajiagha,sen}@research.att.com.

in terms of the available free PE memory, or available free capacity on the uplinks of a PE. We formulate the multi-VPN Relay hub selection problem as a constraint optimization problem that takes into account both memory and bandwidth constraints at each PE, the installation and maintenance cost of a hub, the cost of transporting traffic across the network, and the need for maintaining reliability for each VPN in the face of PE failures. This optimization framework addresses a richer set of constraints and is more general than the formulation considered for the single VPN case in [9].

We develop three algorithmic approaches to solve the optimization problem: (i) Integrated MULTI, our main algorithm, uses a mix of *group knapsack* and *reliable set cover* ideas; (ii) Localized MULTI—a simpler and faster variant of Integrated MULTI; and (iii) Generalized SINGLE—for baseline comparison, we adapt and generalize the LCVSR heuristic from [9] to the multi-VPN problem. While the heuristics from [9] performed well for the single VPN case, the main multi-VPN algorithm (Integrated MULTI) introduced here has provable bounds and enables us to study the various resource tradeoffs on a stronger footing. We extensively evaluate the algorithms using real traffic traces, routing information, and topologies from a large number of VPNs. See Section IV-D for a highlight of the results of our analysis.

Other than single VPN Relaying, we are not aware of any other effort for scaling Layer 3 VPN routing architecture that reduce routing table sizes and satisfy requirements on latency and load. In the context of the public Internet, a number of recent efforts including CRIO [15], LISP [6], and ROFL [5], have addressed the problem of developing scalable routing architectures (more references in [9]). These works did not propose specific algorithms for generating complete indirection configurations that satisfy given user-defined performance constraints on latency and load.

The remainder of the paper is organized as follows. Section II is an overview of the Relaying concept and formulates our optimization problem. Section III presents the main hub selection and spoke assignment algorithm, its proof of correctness and finally establishes theoretical hardness results which show that our guarantees are the best possible unless P = NP. Section IV presents our experimental evaluations. Finally, Section V concludes the paper.

II. Background

In this section, after presenting the technologies leveraged, we formulate the optimization problem that we solve in later sections.

A. MPLS VPN and Relaying overview

Each customer site in an MPLS VPN (see Fig. 1(a)) connects to one or more provider edge (PE) routers in the provider network. Each customer edge (CE) router announces its own address blocks (i.e., routes) to the PE router(s) it is connected to. Each PE router in turn advertises all routes it received from its directly connected CE routers, as well as the routes for the CE-PE access links, to all other PEs in the same VPN. Each PE maintains all the advertised routes for that VPN in a distinct Virtual Routing and Forwarding (VRF) table for later packet delivery. For instance, in Fig. 1(b), a VPN with 5 sites has to install 5 routes on each of the 5 PEs . This setup ensures direct end-to-end reachability across the provider network, since each PE has routing knowledge to forward a given packet to the PE that is directly connected to the CE advertising the destination address of that packet. Outgoing customer traffic from a CE is encapsulated via MPLS at the local PE and carried across the provider network, decapsulated at the remote PE router and handed off to the customer router.

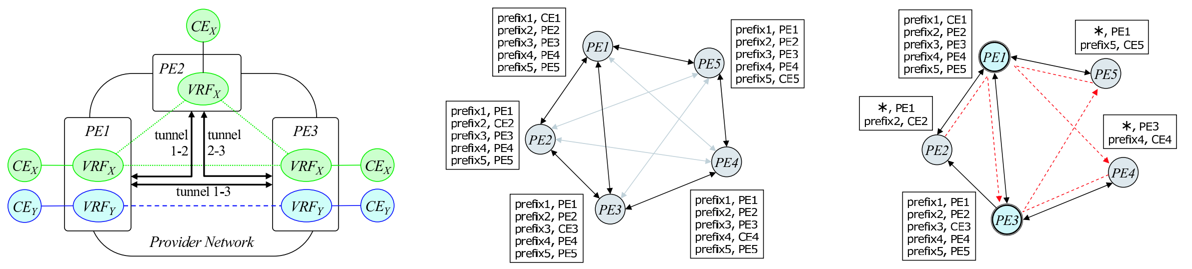
A provider network hosts many hundreds or thousands of such VPNs, and a VPN can consist of hundreds or thousands of sites. A single PE often serves hundreds of VPNs and needs to maintain a per-VPN routing table, containing routing information for all advertised routes for each VPN. Thus, the VPN routing tables in PE routers grow very fast as the number of customers (i.e., VPNs) and the number of routes per customer increases. The problem is particularly challenging considering that each VPN has an entire private IPv4 address space for itself and customers have to advertise at least two routes per site (one route for the site and another for CP/PE link, e.g., a company with 5000 sites will inject at least 10,000 routes on each PE that it is connected to). As a result, PE router memory space required for storing VPN route has become a critical bottleneck in provisioning new customers.

Relaying [9] was introduced as a solution to contain the impact on memory of the any-to-any connectivity made available to MPLS VPNs. The PEs supporting a given VPN are categorized as either hub PEs (hubs) or spoke PEs (spokes). The hubs store all the routes advertised within a VPN, as PEs do in typical VPNs today, while the spokes need only to keep the local routes of the sites attached to the PE and one unique default route, that points to the closest PE. Therefore, when a spoke receives a packet from a local site, it forwards it to a hub that forwards it to its final destination. This approach significantly reduces the memory on the spoke PEs (see Fig. 1(c). Unfortunately, this comes at the cost of additional traffic being relayed on the backbone and increases latency for the VPNs. A key strength of Relaying is that the provider can implement the technique simply by modifying the configuration of routers in the provider network, without requiring changes to the router hardware and software. Deploying Relaying involves (i) *hub selection*: selection of a set of PEs as the hubs for a VPN and (ii) *hub assignment*: determining for each spoke, its corresponding hub.

B. Multi-VPN Relaying Problem Formulation

In the case of the multi-VPN Relaying optimization problem, a more thorough cost model needs to be created to navigate the solution space and new constraints need to be added to those for single-VPN Relaying.

First, the cost model should reflect the fact that every router to be deployed has fixed costs to cover the hardware purchase, the deployment and the maintenance regardless of the usage of that equipment. Second, the cost should include the capacity of the router utilized by the VPNs. More efficient usage of a router's resources is preferable as it frees up more resources



(a) (b) (c)

Fig. 1. (a) MPLS VPN service with three PEs; two customer VPNs (X, Y) exist, (b) Direct reachability, (c) Reachability under Relaying.

for new additions. The utilization cost is a function of multiple resources: the memory utilization that is driven by the number of routes installed (i.e., ratio of memory used on a PE to the total memory available), the CPU utilization that is driven by BGP sessions, customer features and traffic load, and the number of spare ports available. Finally, while Relaying can reduce the amount of memory, it will increase the distance traversed by traffic that is relayed. This is a direct cost for the provider as it will involve additional resource consumption in the backbone. Since the added distance traversed in the backbone is approximately proportional to the added latency and will turn out to be one of the constraints, this network transport cost will be expressed as a function of latency. Our objective is to minimize the cost of these dimensions, which can be combined by evaluating their monetary costs.

The objective to minimize is therefore

$$\alpha|H| + \beta \sum_{ijk} v_{ijk} l_{ijk} + \gamma \sum_{i} u_i, \tag{1}$$

where α, β and γ reflect the monetary costs, $|H|$ is the number of hub PEs, u_i is the utilization of PE i, v_{ijk} is the volume exchanged between PE i and PE j for VPN k, l_{ijk} is the additional latency due to the additional distance $d(i, hub(i,k)) + d(hub(i,k), j) - d(i,j)$ where $hub(i,k)$ is the hub assigned to PE i for VPN k and $d(i,j)$ is the distance between PE i and PE j.

While minimizing this cost function, we require that the solution should not violate some key constraints:

- Reliability for a constant number of PE failures: each spoke site is to be assigned to ρ hubs, for a constant integer $\rho > 0$, to provide a topology that is survivable when we might have up to $\rho - 1$ PE failures;
- No packet incurs a latency increase of more than θ so that the experience of the end user is not impacted by the Relaying architecture;
- The uplink bandwidth of no PE is saturated; and
- The memory usage of each PE is bounded by configurable utilization upper limits.

An additional objective can be to balance the routing table sizes across PEs so that the network provisioning team can more easily provision new customers on any PE. This can be achieved by changing the memory constraint and by lowering the memory utilization allowed. Finally, in this paper, we seek practical *approximation algorithms with approximation factors* C, i.e., algorithms which are not necessarily optimal but their solutions have costs within a factor C of the optimum.

C. Additional assumptions

To solve this constraint optimization problem, some assumptions will be made. First, since memory is currently the bottleneck on PEs, we will assume that the PE utilization is simply the memory utilization on the PE, which is proportional to the number of routes installed. Second, the cost model will assume that the transport cost is linear and proportional to the distance traversed and that the cost of traffic traversing the network for the weighted average distance of today's traffic is \$4 per Mbps [14]. In our model, the cost of PEs will vary linearly with the memory utilization and be amortized over 36 months: the upfront cost for a PE will be \$200K when empty and reach a total cost of \$400K if the memory is fully utilized (see [2] or [1] for sample prices that will vary depending on the configuration). A sensitivity analysis of the cost input in Section IV will show that even a ten-fold change in the ratio of bandwidth cost vs. router cost or memory cost does not change the solutions considerably.

III. Theoretical results

In our terminology, a *virtual PE* is a pair (PE, VPN), where VPN is installed on PE. In our relaying solution, any virtual PE should be assigned to a number of hubs. The resemblance may tempt us to model this problem using the *facility location* (FL) problem.[1] We can have a client for each virtual PE, and a facility for each PE (as a candidate hub). One challenge here is that although latencies are metric in our problem, we need to resort to non-metric connection costs to accommodate for the maximum permissible latency increase. We show our problem cannot be approximated to within a factor $o(\ln N)$, where N is the number of virtual PEs; see Subsection III-C. Nevertheless, *metric facility location*[2] has constant factor approximation algorithms.

If we relax the latency threshold, bandwidth and memory constraints, as well as reliability requirements, we can cast this basic model of the problem as a (non-metric) FL instance. The only known work studying facility location problems with "unsplittable hard capacities" for servers is that of Bateni and Hajiaghayi [4]. Even in this work, however, metricity is

[1]In a facility location problem, we have facilities with associated opening cost, and clients which should be connected to open facilities. The goal is to find the optimal set of facilities to open, and connections to build between clients and opened facilities, so as to minimize the open facilities' opening costs plus the connection costs.

[2]Metric facility location is a variant of the facility location problem in which connection costs satisfy metricity.

assumed. In this section, we show how to tackle the problem in presence of all these extra constraints. In a high level view, our algorithm combines the greedy approach for solving *Set cover* and the Dynamic Programming (DP) technique of the *Knapsack* problem; furthermore, we need to generalize these techniques to a fault-tolerant setting. In standard set cover, we are given a collection $\mathcal{C}$ of subsets of a finite set $\mathcal{U}$, and the goal is to find a sub-collection $\mathcal{C}' \subseteq \mathcal{C}$ of minimum size such that every element in $\mathcal{U}$ belongs to at least one member of $\mathcal{C}'$, i.e., $\mathcal{U} = \cup_{S \in \mathcal{C}'} S$. In standard Knapsack, we are given n of items, each having a value and a weight, and a knapsack capacity c. The goal is to find a maximum-value subset of items whose total weight is at most c.

To summarize, the constraints in our model are as follows:

1) We should avoid assigning a PE to a hub causing a latency increase of more than θ;
2) Each PE has a memory limit; it defines the maximum number of routes that can be stored in the PE;
3) Each PE has a bandwidth limit. The total traffic through the PE should not exceed the limit; and
4) We might have a reliability parameter, say ρ, dictating that ρ hubs should be provisioned for any virtual PE. We usually work with $\rho = 2$, though our algorithms can be easily generalized to any constant integer $\rho > 0$.

A. Main algorithm

We first simplify the objective function before stating our algorithm. We deal with this simplified version in the rest of the section. We show that any instance $\mathcal{I}_1$ with the objective (1) can be transformed into an instance $\mathcal{I}_2$ which has a simpler objective function we define here:

$$\sum_{i \in H} f_i + \sum_{i,j,k} v_{ijk} l_{ijk}. \tag{2}$$

The new objective (2) does not have the third term of the general objective (1) (the one for memory usage of hubs); neither does it have the multipliers α, β and γ. It only accounts for the initialization costs of selected hubs, H, and for the extra bandwidth cost. Note that we allow different costs for different facilities. The proof of the following lemma follows from a standard reduction (see for instance [10]) and we defer its proof to the appendix.

Lemma 1: For any constant $\epsilon' > 0$, we can transform in polynomial time an instance $\mathcal{I}_1$ having an objective in the form (1) to an instance $\mathcal{I}_2$ with the simplified objective form (2) with a blowup factor at most $1 + \epsilon'$.

The general idea of the proof is as follows: If a particular router R_i is made a hub and gets u_i utilization, then it pays a price of $\alpha + \gamma u_i$ in (1). In the second instance, we place multiple routers to model R_i of the first instance. Each of these new routers has a particular memory limit (spanning the range from zero to U_i, the total memory of R_i); the cost of such a copy in the new instance is $\alpha + \gamma U_{ij}$ where U_{ij} is the new memory limit of the copy. We can then show that a small number of copies suffices to get the said bound of the above lemma.

Our main algorithm follows a framework of greedily picking an (approximately) most "efficient" hub to which we assign a subset of virtual PEs and repeating until we satisfy all the virtual PEs. A virtual PE is satisfied if it is assigned to a hub. At each step of the algorithm, we have a set of yet unsatisfied virtual PEs and we are to find the most "efficient" hub (i.e., a hub whose selection cost divided by the number of virtual PEs it satisfies is minimum) for some of them. We can guess the location of the hub (conceptually, but algorithmically we try them one by one), and then find the best assignment for the hub. Fixing the candidate hub, and unsatisfied virtual PEs, the subproblem (of finding the best assignment for this hub)—which we will call *Density Group Knapsack*—is as follows: there are n virtual PEs (items) partitioned into k VPNs (groups). Each VPN has a memory footprint s_i (number of routes needed for this VPN in each hub), and each virtual PE has a bandwidth cost c_j (summation of traffic times latency increase for the traffic generated at this particular virtual PE). There is also a global one-time-pay cost F (i.e., $F = f_h$ if the hub is fixed to be h). Let $\mathcal{V}$ be the set of VPNs. The set of virtual PEs corresponding to VPN i is denoted by $\mathcal{P}_i$. We seek to

- pick some VPNs $\mathcal{V}' \subseteq \mathcal{V}$ whose total memory footprint does not exceed M, the memory capacity of the current hub, namely $\sum_{i \in \mathcal{V}'} s_i \leq M$,
- deploy their memory footprint in this hub's memory,
- and then serve using this hub some virtual PEs from each picked VPN, i.e., we serve a subset of $\cup_{i \in \mathcal{V}'} \mathcal{P}_i$,
- so as to minimize the ratio of cost to number of virtual PEs served.

As an extension, we are also given a bandwidth usage v_j for each virtual PE and the sum of bandwidth usages of virtual PEs we pick should not exceed the bandwidth limit B. To give the idea of the algorithm, we ignore the issues of bandwidth limits and reliability requirement (fault-tolerance) for now.

Assuming costs are small integers, we can solve the problem using a *dynamic programming (DP)* technique. Let $c(i, \ell)$ be the sum of the bandwidth costs of the ℓ cheapest virtual PEs in VPN i. Obviously, if we are to choose ℓ virtual PEs from a VPN, we will pick the cheapest ones. It is straight-forward to use $c(i, \ell)$ values to fill the table $D(i, m, c)$, defined as the minimum memory usage required to serve m items from VPNs 1 through i by a bandwidth cost c. Finally, we can go over the table for $i = k$ (k is the number of VPNs) and pick the best ratio cell whose memory usage does not exceed M. Let us once more stress that the intuitive ideas given in this paragraph is for a special case of the problem, in which we have extra assumptions about input values and furthermore bandwidth and reliability requirements are completely ignored.

To lift the assumption about small integers, we use ideas similar to those for a *Polynomial-Time Approximation Scheme (PTAS)*, i.e., a $(1+\epsilon)$-approximation algorithm for any $\epsilon > 0$, for the Knapsack problem. Now, we modify the instance $\mathcal{I}_2$ (for a particular hub) to be an instance $\mathcal{I}_3$, in which all bandwidth cost and bandwidth usage values have been discretized to be integer multiples of some granularity (one for cost and one for usage). We show in Lemma 2 that these two instances are roughly equivalent. Our goal here is to find an assignment to this hub of minimum "density," i.e., the ratio of cost to the number of virtual PEs satisfied. Let the cheapest virtual PE have (bandwidth) cost L. The best density is at most $F+L$, since one possibility is to only serve the cheapest virtual PE. So, all virtual PEs costlier than $F+L$ can be ignored. For a given $\epsilon' > 0$, we can define granularity to be $\tau = \epsilon'(F+L)/N$, and costs are rounded down to the next integer multiple of τ. Similarly to the granularity for costs, we define the bandwidth usage granularity $\kappa = \delta B/N$, where B is the bandwidth limit of the current hub.

Lemma 2: The value of the optimum solution for $\mathcal{I}_2$ is no better than that of $\mathcal{I}_3$. Besides, any solution of $\mathcal{I}_3$ with bandwidth cost C is a solution for $\mathcal{I}_2$ in which memory usage is the same, whereas bandwidth cost is at most $(1+\epsilon')C$ times that of $\mathcal{I}_3$, and bandwidth usage is at most $(1+\delta)B$.

Proof: The total error caused by the rounding of costs is at most $N\tau = \epsilon'(F+L)$. We next note that the total cost (rather than density) is at least $F+L$, as we have to pay the facility cost F and the cheapest virtual PE has bandwidth cost L. So, the error in cost is at most an ϵ' factor of the cost. As for bandwidth usage, the solution for $\mathcal{I}_3$ uses at most B. The error is at most $N\kappa = \delta B$. ■

The number of distinct cost values would be at most N/ϵ', and the number of distinct bandwidth usage values is at most N/δ. Hence, one can divide these values by τ and κ respectively to get an instance in which bandwidth cost and usage values are polynomially bounded integers. Now, we can proceed with a dynamic programming algorithm to solve an instance of $\mathcal{I}_3$.

Algorithm 1 is more elaborate than the idea given above. One major complication arises because of the bandwidth constraints. To update the DP cells, we need to compute the least cost of serving a virtual PEs from a certain VPN having a particular bandwidth usage. Had we not had the bandwidth constraints, we could just sort the virtual PEs in each VPN according to their cost and pick the a cheapest ones. The procedure COSTWITHCAP solves this subproblem. In particular, the procedure fills using a DP approach the values $T'[vpn, cnt, bw]$ that store the minimum cost of serving `cnt` virtual PEs (i.e., $A'[vpn, cnt, bw]$) of the VPN `vpn` with bandwidth usage of `bw`.

Algorithm 1 ComputeAssignment

```
{$f_i$: cost of building/maintatining PE $i$ as a hub
 $b_i$: bandwidth limit on PE $i$
 $m_i$: memory capacity on the routing table size of PE $i$
 $s_i$: memory usage of VPN $i$ (routing table)
 $d(i,j)$: latency on the path between PEs $i$ and $j$
 $v_{ijq}$: traffic volume between PE $i$ and PE $j$ for VPN $q$
 $\theta$: the maximum permissible increase in latency}
$\epsilon' \leftarrow \epsilon/3$
Transform the objective into one of (2)
while there is an unassigned virtual PE router do
  Set BestRatio $\leftarrow \infty$, BestSet $\leftarrow \emptyset$, BestHub $\leftarrow 0$
  for $h = 1$ to $n$ do {i.e. for each candidate hub}
    Let $S$ be the set of unsatisfied (less than $\rho$ times
    assigned) PE routers $s$ (possibly including $h$ itself) for
    which $h$ is not currently a hub, yet it is feasible, i.e.,
    $d(s,h) + d(h,d) - d(s,d) \leq \theta \ \forall s, d$ with $v_{sd} > 0$
    for each virtual PE $j$=(PE $s$, VPN $q$) in $S$ do
      Let $v_j$ be total traffic it needs to send
      Let $c_j$ be the bandwidth usage cost for this assign-
      ment i.e., $\sum_d v_{jdq}[d(j,h) + d(h,d) - d(s,d)]$
    end for
    Discretize $c_j$ and $v_j$ vals w.r.t. precisions $\epsilon'$ and $\delta$
    Let $\tau$ and $\kappa$ be the respective granularities
    for vpn $= 1$ to $k$ do
      run COSTWITHCAP(vpn)
    end for
    Run MAINDP(h)
    Run EXTRACTSOLUTION(h)
  end for
end while
Assign BestSet to BestHub
```

B. Proof of correctness

In this section, we prove the following main theorem.

Theorem 3: Algorithm 1 (COMPUTEASSIGNMENT), given any fixed $\epsilon > 0$ and $\delta > 0$, runs in polynomial time and reports a solution whose cost does not exceed $(1+\epsilon)(1+\ln \rho N)\text{cost}(\mathsf{OPT})$. The solution assigns ρ hubs to each virtual PE, such that bandwidth constraint of a hub is violated by at most a factor δ, and there is no memory limit violation.

First we need rephrasing a seminal result:

Lemma 4 ([8]): Suppose that an algorithm works in iterations and in iteration i, finds and adds to the partial (infeasible) solution a hub that covers a subset S of (previously unsatisfied) virtual PEs. Let u_i be the number of unsatisfied virtual PEs before iteration i. If for every i, the ratio of the added cost to $|S|$, called *density*, is at most $c \cdot \frac{\text{cost}(\mathsf{OPT})}{u_i}$ for some parameter c, then the total cost of the solution output by the algorithm is at most $c \cdot (1+\ln n)\text{cost}(\mathsf{OPT})$, where n is the total number of virtual PEs to be satisfied.

Next we need several lemmas about the performance and correctness of the subroutines we use. For the rest of this subsection, we work with an instance of $\mathcal{I}_3$. The algorithm COMPUTEASSIGNMENT runs in iterations. At each iteration, we find the most efficient hub for the remaining virtual PEs. A remaining virtual PE is one that has not yet been assigned to ρ hubs. These virtual PEs are identified, and the values for feasible assignments are computed. The main part of the algorithm—procedure MAINDP—runs a DP to compute the solution to a *Density Group Knapsack* instance. However, procedure COSTWITHCAP is run before that, to update an auxiliary table needed by MAINDP. Afterwards, EXTRACTSOLUTION can go over the DP table and find the most efficient assignment.

Lemma 5: Procedure COSTWITHCAP computes, in polynomial time, for any VPN `vpn`, integer `count` and bandwidth usage `bwidth`, a solution $A'[\text{vpn}, \text{count}, \text{bwidth}]$ with cost $T'[\text{vpn}, \text{count}, \text{bwidth}]$, which is the least cost of satisfying `count` PEs from given VPN with the specified bandwidth usage.

Proof: Let, for the sake of contradiction, $A'[\text{vpn}, \text{count}, \text{bwidth}]$ be the cell with least value for `count` with a wrong value. Suppose the best solution is

Algorithm 2 CostWithCap(vpn)

```
{For simplicity, we assume the PEs in this VPN are numbered from 1 to $n$}
Initialize $T'$[vpn,count,bwidth] to $\infty$ for all values
$T'$[vpn,0,0] $\leftarrow 0$ and $A'$[vpn,0,0] $\leftarrow \emptyset$
for cnt $= 1$ to $n$ do {$n$ is # of elements in this vpn}
  for bw $= 0$ to $b_i$ step $\kappa$ do
    for $i = 1$ to $n$ do
      if $v_i \le$ bw and
      $T'$[vpn,cnt,bw] $>$ $T'$[vpn,cnt-1,bw$-v_i$]$+c_i$ then
        $T'$[vpn,cnt,bw] $\leftarrow$ $T'$[vpn,cnt-1,bw$-v_i$]$+c_i$
        $A'$[vpn,cnt,bw] $\leftarrow$ $A'$[vpn,cnt-1,bw$-v_i$]$\cup\{i\}$
      end if
    end for
  end for
end for
```

Algorithm 3 MainDP(h)

```
Initialize DP table $T$[cost, vpn, served, bandwidth] $\leftarrow \infty$
$T[0,0,0,0] \leftarrow f_h$
$A[0,0,0,0] \leftarrow \emptyset$
for vpn $= 1$ to $k$ do
  for feasible values of (srvd, cst, bw) do
    $T$[cst, vpn, srvd, bw] $\leftarrow$ $T$[cst, vpn-1, srvd, bw]
    $A$[cst, vpn, srvd, bw] $\leftarrow$ $A$[cst, vpn-1, srvd, bw]
    for (s, b) where $1 \le$ s $\le$ srvd and $0 \le$ b $\le$ bw do
      c $\leftarrow$ $T'$[vpn, s, b]
      if c $\le$ cst and $T$[cst, vpn, srvd, bw] $>$
                $T$[cst-c, vpn-1, srvd-s, bw-b] $+ s_{\text{vpn}}$ then
        $T$[cst, vpn, srvd, bw] $\leftarrow$
                $T$[cst-c, vpn-1, srvd-s, bw-b] $+ s_{\text{vpn}}$
        $A$[cst, vpn, srvd, bw] $\leftarrow$
                $A$[cst-c, vpn-1, srvd-s, bw-b] $\cup$ $A'$[vpn, s, b]
      end if
    end for
  end for
end for
```

Algorithm 4 ExtractSolution(h)

```
MAX COST $\leftarrow f_h + \min_{j \in S} c_j$
for served $= 1$ to $N$ do
  for cost $= 0$ to MAX COST step $\tau$ do
    for bandwidth $= 0$ to $b_i$ step $\kappa$ do
      if $T$[cost, vpn, served, bandwidth] $< m_i$
      and cost/served $<$ BestRatio then
        BestRatio $\leftarrow$ cost/served
        BestSet $\leftarrow$ $A$[cost, vpn, served, bandwidth]
        BestHub $\leftarrow i$
      end if
    end for
  end for
end for
```

a set A^*, and let i^* be the last virtual PE in A^*. Then $A'[\text{vpn}, \text{count} - 1, \text{bwidth} - v_{i^*}]$ has the correct value, and thus when the algorithm considers $i = i^*$ the correct solution A^* should be accepted. The running time is $O(N^3/\delta)$, which follows from the simple for-loop structure of the algorithm and the discretization process. ■

Lemma 6: Procedure MAINDP computes, in polynomial time, the least memory usage for a given hub to service a specific number of routers obeying its bandwidth capacity.

Proof: We prove a stronger result: for all values $0 \le i \le k$, we compute the least memory usage of a solution to serve m virtual PEs from VPNs 1 through k that has cost c and bandwidth usage u; the solution would be $A[c, i, m, u]$ whose value is $T[c, i, m, u]$. Once again, for the sake of contradiction, consider a cell $T[c, i, m, u]$ having a wrong value whose i is minimum. Clearly $i \neq 0$. Let A^* be the best solution in this case. If A^* does not have any virtual PE from VPN i, we have a contradiction by minimality of the counterexample. Otherwise, A^* uses some m' virtual PEs from VPN i. Lemma 5 and the update rule makes sure that we consider this with the correct values for bandwidth usage and cost of these m' virtual PEs (we guess them). This is a contradiction as we should consider A^* and get the optimum solution in our table. The running time is $O(kN^5/\epsilon'\delta^2)$. ■

The last remaining step before proving our main theorem (Theorem 3) is the *reliability* or *fault tolerance* feature needed in our algorithm. At this point, we need to show that Lemma 4 is also valid in case of fault tolerant optimization. In our algorithm, each virtual PE is assigned to ρ, say two, hubs. It arbitrarily picks one of them unless it is forced to switch to another one due to node failures. This problem is closely related to *Reliable Set Cover* defined below. In *Reliable Set Cover*, we are given a collection $\mathcal{C}$ of sets from a ground set $\mathcal{U}$ of n elements. The goal is to pick the minimum cost subcollection $\mathcal{C}' \subseteq \mathcal{C}$ so as to cover each element at least ρ times; i.e., $|\{S : S \in \mathcal{C}', e \in S\}| \ge \rho$ for any element $e \in \mathcal{U}$. The natural algorithm RELIABLEGREEDY works in

Algorithm 5 ReliableGreedy

```
$\mathcal{C}' \leftarrow \emptyset$
for all $i \in \mathcal{U}$ do
  $\rho_i \leftarrow \rho$ {remaining covering requirement of $i$}
end for
while $\exists i \in \mathcal{U}$ with $\rho_i > 0$ do
  Let $S \in \mathcal{C}$ have maximum efficiency = $\frac{\text{cost}(S)}{|S|}$
  $\mathcal{C}' \leftarrow \mathcal{C}' \cup S$
  $\mathcal{C} \leftarrow \mathcal{C} - S$
  for all $i \in S$ do
    $\rho_i \leftarrow \rho_i - 1$
  end for
end while
```

iterations and picks the most *efficient* set at each iteration, i.e., the set whose ratio of cost by number of new covered elements is minimized. Note that each element is to covered ρ times. Hence, as long as an element is not completely satisfied, it is considered a newly covered element whenever it is included by a set. We state and prove this lemma—a generalization of Lemma 4—in this abstract formulation.

Lemma 7: Suppose that an algorithm works in iterations and in iteration i, finds and adds to the partial (infeasible) solution a set that covers a subset S of (previously unsatisfied) elements. Let u_i be the number of unsatisfied elements (taking into account the multiplicity as a weight) before iteration i. If for every i, the ratio of the added cost to $|S|$, called *density*, is at most $c \cdot \frac{\mathsf{cost}(\mathsf{OPT})}{u_i}$ for some parameter c, then the total cost of the solution output by the algorithm is at most $c \cdot (1 + \ln \rho n)\mathsf{cost}(\mathsf{OPT})$, where n is the total number of elements to be satisfied.

The proof of the lemma appears in the appendix. The proof is done in a similar fashion to that of Lemma 4. The main difference here is that each element can be selected multiple times. At each step, we show that the cost of the greedy algorithm is comparable to that of the OPT. The factor ρ thus appears in the guarantee and in the statement u_i needs to take multiplicities into account.

Now we summarize with the proof of our main theorem.

Theorem 3: We let $\epsilon' = \epsilon/3$. By Lemma 1 transforming to the instance with simpler cost function causes an increase in cost of at most a $1 + \epsilon'$ factor. For solving a least density subproblem, Lemma 2 ensures that the discretization does not increase the cost by more than $1+\epsilon'$. We can solve the discrete version of the subproblem with the simpler cost function, due to Lemma 6. Lemma 7 completes the proof, noting that $(1 + \epsilon')^2 < 1 + 3\epsilon' = 1 + \epsilon$. ∎

Finally we emphasize that in this section, we considered *soft capacities*, i.e., we allow multiple routers to be placed at a facility, each obeying the routing table size limit. Insisting on *hard capacities*, i.e., if we cannot replicate routers, makes the problem very hard. Indeed, we cannot get any approximation guarantee on costs if we have hard capacities unless P = NP.[3] Hence, soft capacity relaxation is necessary for tractability, yet in practice, we can resort to heuristics which disallow soft capacities without harming the solution in our current data set.

C. Hardness of approximation

In this section, we show that the algorithm in the previous section provides the best guarantee one can hope for and it is essentially optimal. Here, we do not even resort to bandwidth or memory usage restriction. This also implies that the hardness is true for the single-VPN special case.

Theorem 8: The problem of minimizing total cost while obeying the hard threshold on latency θ is not approximable to within $(1 - o(1)) \ln n$ unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$) (or to within $c \ln n$ for some constant $c < 1$, unless P = NP). Indeed, for any $c' > 0$, even if we allow a latency of up to $c'\theta$, we would not get an approximation ratio of better than $\Theta(\ln n)$ for cost.

[3]Any bounded approximation ratio implies we can solve the Knapsack problem (or its special case called the *Subset Sum* problem) which is NP-hard: the idea is to have two candidate hubs, each with memory capacity of M. We have a total of $2M$ routes for different VPNs and we want to know whether there is a way to assign VPNs to these two virtual PEs such that the memory usage in each virtual PE is not violated. This is equivalent to finding a subset of a set of integers whose sum is exactly half the total sum.

Proof: Suppose we are given an instance of *Set Cover* with n sets S_i, and elements e_j. In our relaying instance, we have a virtual PE for each set and $2n^2$ ones for each element. The element virtual PEs corresponding to an element e_i of *Set Cover* are labeled v_{ij}, v'_{ij} for $1 \leq j \leq n^2$. Each pair v_{ij}, v'_{ij} need to send some flow to each other. We pick $\alpha = 1$ and $\beta = \gamma = 0$. Distance between any two virtual PEs is $D > 0$ unless one is a set S_i and the other is a node corresponding to an element $e_j \in S_i$. In this case, their distance is $D/2$. Note that the distances satisfy metricity. Finally, θ is set to be 0.

Take any pair v_{ij}, v'_{ij}; their hub can only be one of v_{ij}, v'_{ij} or some set containing e_i. We claim that the optimal solution only uses set nodes as hubs. If some v_{ij} is picked as a hub, then there should be at least n^2 hubs among the nodes corresponding to element e_i; otherwise, one pair connects to a set node and all the other nodes could do the same. However, the cost of this solution is at least n^2, while there is a solution with n hubs (namely, the one that picks all the set nodes).

Hence, each element connects to at least one set, and our cost is proportional to the number of sets. We can then invoke the established hardness of *Set Cover* to draw the conclusion [7], [13]. Note that any multiplicative relaxation of $\theta = 0$ would give the same result. ∎

IV. Evaluations

A. Relaying Algorithms

We evaluate the performances of the following algorithms for determining the selection of hubs and assignment of hubs to spokes for the different VPNs:

- "Status Quo" (STATUS) which shows the current situation of the network without any optimization.
- "baseline" (BASE) which is a generalization of the algorithm proposed by Kim et al. [9] to a multi-VPN setting, in which optimization for each VPN is done independently. In this algorithm, VPNs are dealt with in a random order. For each VPN, we run the single-VPN optimization of [9] to minimize the total memory usage, using the residual bandwidth and memory on the PEs; the use of residual capacities is to avoid violating those hard thresholds.
- "multi-VPN optimization" (MULTI) that is the algorithm we proposed earlier in this section.

MULTI is our major contribution and has some variations: Integrated (INT) and localized (LOCAL). The former runs an optimization once for the whole instance, whereas the latter runs the optimization in a "localized" manner. At each step of the localized algorithm, part of only *one* VPN can be handled, whereas each step of INT can deal with parts of different VPNs. In other words, each step of INT can handle multiple steps of LOCAL. As is expected, the optimization for LOCAL is simpler and runs faster; however, the results are inferior to those of INT, because LOCAL cannot make the more global optimization—it might well happen that making a collective decision for all the VPNs is better than that of making it individually for each VPN. LOCAL* and INT* are unconstrained versions of the above algorithms, in the sense that a new VPN might be assigned to a PE which is not currently part of the VPN. In the constrained version, on the other hand, each PE can only be a hub for those VPNs that currently have some CE attached to it from that particular VPN.

To implement the algorithms, we need to tackle the issue of large running time. Although the algorithms run in polynomial time, the running time guarantees are still too large. We notice that the memory usage of a single VPN is usually not large compared to the memory capacity of PEs. Were this always the case, a good approximation would be to solve fractional knapsack (using an LP-solver such as CPLEX for a natural linear programming) rather than pseudopolynomial time dynamic programming solutions to knapsack. However, this is not the case. Fortunately though, the memory usages can be divided into two groups: the VPNs whose memory usage is large and those with small memory usage. The former group need a dynamic programming solution, whereas a fractional knapsack solution is sufficient for the latter group. We mix these ideas with a couple of heuristics to make the running time of our algorithms tractable.

All our algorithms run in a reasonable amount of time, which ranges from 10 seconds to the worst case of an hour, depending on the parameters used. These running times correspond to instances with hundreds of PEs and thousands of VPNs. Since we only need to run the optimization occasionally, the times are quite acceptable.

B. Setting

We used traffic, configuration and topology data from a large tier-1 ISP, collected during a work-week of June 2008. The data set corresponded to hundreds of PEs, and thousands of VPNs. Size of VPNs ranges from a few to hundreds of PEs and thousands of CEs.

In the evaluations below, unless otherwise specified we use the following default parameters settings: (1) the constraints on the resources at each PE are 100% of the memory (this allows us to compare our scheme which can handle explicit constraints on memory usage to others) and 80% of the bandwidth (leaving some room for handling traffic variability); (2) We limit the increase in path length due to relaying to 200 miles which corresponds to the threshold (θ) for permissible additional unidirectional latency being 2.5 msec. Most network applications today can easily tolerate increases in excess of this value. The selection of θ was also guided by our desire to limit the extra load on the network due to relayed traffic traveling longer distances; and (3) the default fault-tolerance requirement is the non-reliable instance where each spoke in a VPN needs to be assigned to a single hub PE.

C. Simulation results and analysis

The algorithms BASE, LOCAL, INT, LOCAL* and INT* cost 80%, 73%, 66%, 36% and 30% of that of "status quo" (STATUS) respectively. The fraction of PEs selected as hubs are 79%, 72%, 64%, 36% and 31% respectively for the different schemes. BASE and MULTI (all variations) reduce the total memory usage to about 15% of STATUS QUO, for the default parameter settings. The bandwidth cost for these solutions are within 1.5% of each other.

We note that our results (including hub assignments) were relatively stable when using data for different days and exhibited less than 1% of changes, possibly due to the relative stability of VPN communication behaviors.

It might be the case that there are poorly utilized hubs very close to each other. While optimizing using INT*, we may be able to shut down many of these hubs and maintain only a few. The cost of INT and INT* can substantially differ as this is not allowed by INT; the current hubs have poor utilization because each has its own set of small VPNs.

In the following, we mainly consider the MULTI algorithms which, and as shown above, realize the lowest overall cost. We always compare the total cost and total memory usage to those for STATUS, and present the values as percentages.

1) Memory utilization limit: Fig. 2 shows the impact of different memory utilization limits on the total cost incurred by different solutions. Across the range of utilization limits, we note that the INT variant always performs better than the LOCAL variant. Also the unconstrained solutions substantially outperform the constrained ones. For example, given a 50% memory limit, the cost for INT* is less than half that for INT. For a given setting, the total number of routes for each scheme were very similar, the main difference in the schemes was in the number of hubs required. We also note that memory utilization has very little impact on the total cost. Further investigation shows that as we decrease the memory utilization threshold, each scheme does need a few more hubs, but the total memory usage stays nearly the same.

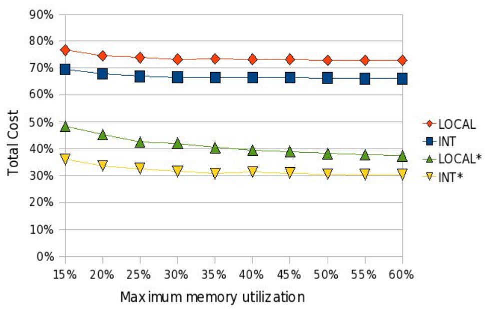
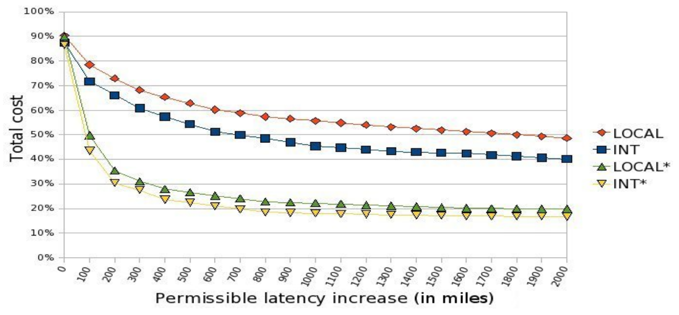


Fig. 2. Effect of maximum memory utilization on the total cost.

Our evaluations showed that the total cost did not change considerably when we change bandwidth limit values within a reasonable range. This implies that the links in the network still have sufficient free bandwidth compared to the traffic demand that they can accommodate the additional relayed traffic load.

2) Permissible latency threshold θ: Fig. 3 illustrates the change in cost and total routes installed caused by varying θ. As latency increases to about 2000 miles, we reach a steady state for the non-constrained algorithms where the solutions incur about 15% of the cost for STATUS. The constrained solutions, on the other hand, approach a state where we still incur roughly 40% of the original cost. We also notice that the total number of routes is essentially independent of the algorithm and the constraint, but more importantly is a function of the permissible latency increase. Even for small θ



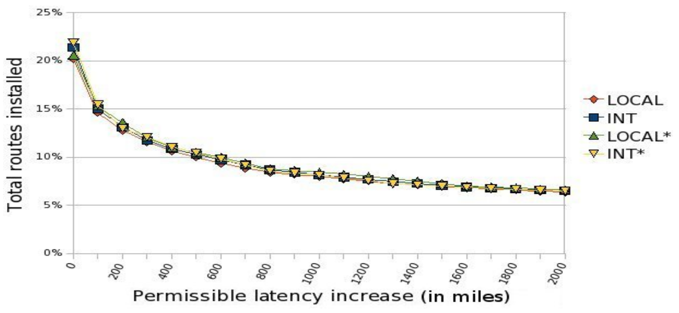


Fig. 3. Effect of varying θ on total cost and memory.

in the few hundred miles range, the total memory requirement reduces dramatically to a small fraction $10-15\%$ of what is needed today.

3) Fault-tolerance: We consider three levels of reliability requirements: 1, 2 and 3hub-reliability. Recall that ρ-hub reliability requires that each spoke in each VPN have ρ candidate hubs—this ensures any-any connectivity even when $\rho-1$ hubs fail. Obviously, designing such a network with high reliability is more costly. In Figs. 4 and 5, we see how cost of the design increases with these extra requirements. This is evaluated at different values for permissible latency increase and memory utilization limit. In each case, the other parameters are set to defaults. We note that 3hub-reliability INT* costs less than 1hub-reliability LOCAL; i.e., if we relax the constraint to allow new VPNs in a PE, we can achieve much higher reliability requirement with the same cost. Also, in general neither the number of routes nor the total cost increase linearly with the reliability parameter; see Figs. 4 and 5 and Table I.

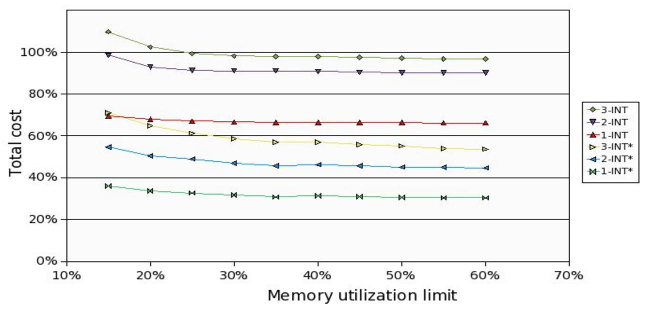
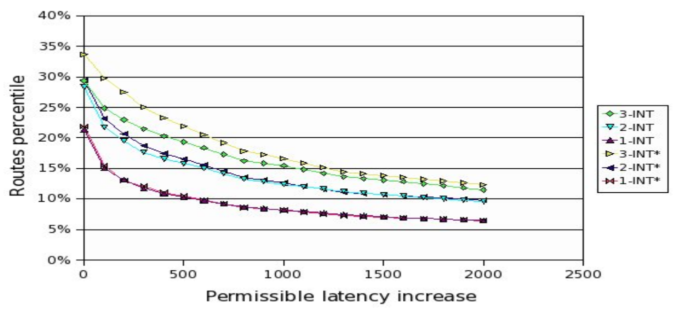


Fig. 4. Comparing solution for different reliability scenarios (1, 2 or 3)

We also found that the total number of routes needed is not significantly impacted by the memory utilization limit. However the memory usage is impacted by the fault-tolerance level and the choice between versions of MULTI, see Table I.

TABLE I
EFFECT OF FAULT-TOLERANCE ON NUMBER OF TOTAL ROUTES NEEDED.

New VPNs in PEs	1hub reliability	2hub reliability	3hub reliability
Allowed	13%	21%	27%
Not allowed	13%	20%	23%



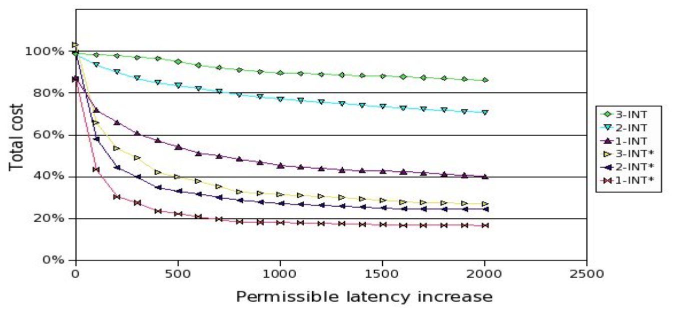


Fig. 5. How do total memory usage and total cost vary with θ?

We next conduct an experiment to explore how the solution for a 2-hub-reliability instance differs from the 1-hub-reliability one. We begin with the 2-hub solution of MULTI and derive a spoke-to-hub assignment by allowing each spoke PE to select one of its assigned hubs at random. We measure the change in memory usage among all the routing table entries required for the 2-hub solution, we only keep those needed to support this spoke-to-hub assignment and remove all others. The results show that 20% of the hubs used no more than half their current memory, while 50% of them utilize more than 90% thereof. The total number of routes we need to store in this case is 79% of what the algorithm provisions for 2-hub-reliability. The remaining 21% of the entries account for the additional overhead of getting 2-hub-reliability seamlessly. Despite this apparent saving, we do not suggest routers should remove these extra tables from their memory and load them only in case a particular hub fails. The reason is that this increases management overhead and disruption time.

Nevertheless, this means that although we have provisioned for a 2-hub-reliability case with all the infrastructure present, we can drop the cost to 15% more than the optimal, by just randomly picking among the two hubs and removing the unnecessary routes. In other words, though our solution is 2-hub-reliable, we can easily extract from it a 1-hub-reliable solution whose cost is very close to the optimum.

4) Distribution of memory and bandwidth utilization: We compare the memory usage for STATUS to the solution proposed by our algorithm (INT*). Fig. 6 illustrates how our algorithm has smoothed out the usage at different routers. The maximum per-PE memory usage was 30% for INT*

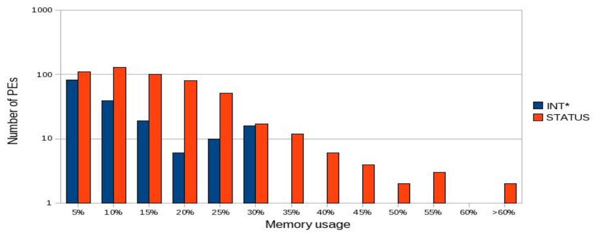


Fig. 6. Improvement in memory utilization distribution.

whereas for STATUS a significant number of PEs had much higher utilization. We stress this effect was not an explicit major objective of our algorithm, but that careful choice of parameters can lead to this desirable effect.

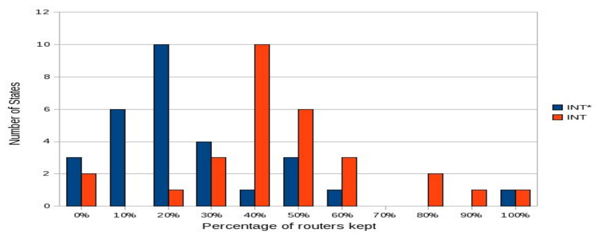


Fig. 7. Geographic spread of hubs

5) Location of hubs: Now, we look at the geographic spread of hubs selected by our algorithms. Note that the hub location is constrained to locations where PEs are already present. We compare the constrained scheme INT vs. the unconstrained scheme INT*. We consider states with at least four routers. Fig. 7 shows that for algorithm INT, most states need to keep more than half of their current PEs as hubs, while in the INT* solution, most states need less than half the routers. This finding shows we have reduced the number of hubs in almost all the States, and not in a few select geographical locations.

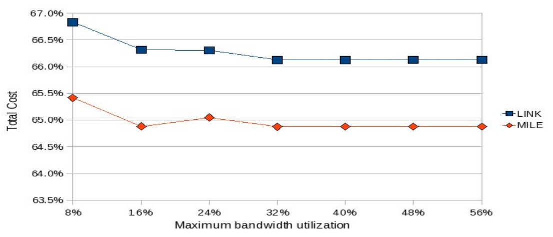


Fig. 8. Comparison of different bandwidth cost models.

6) Stability of cost model: In Fig. 8, we compare between two different models of routing costs and show that the results are very close. One labeled as LINK measures the cost of routings per link. Each link, regardless of its length, costs $\$4$ per month per Mbps, while the other one, marked as MILE, charges this $\$4$ price on the weighted average distance we have in our data. This is done, in order to make the costs more realistic. However, we note that the results are not too different and this further demonstrates the stability of our model. Indeed, we have done more experiments to measure the stability of our model. It turns out that even a ten-fold change in ratio of bandwidth cost vs router cost or memory cost does not change the solutions appreciably (i.e., less than 1%).

D. *Summary of results*

To summarize the main results, our analysis shows that

- There is consistent, significant reduction in total cost (using any assignment to the parameters), as we use better algorithms; i.e., STATUS, BASE, LOCAL, INT, LOCAL* and INT*. The last algorithm decreases the total cost by a factor three from STATUS for the default parameter values.
- Although we try to minimize the unified objective (1), we are also able to achieve an 85% reduction on total memory usage which is close to a theoretical limit.
- Reducing the memory limit from 100% to 35% changes the total cost by less than 5%, whereas the change from 35% to 15% imposes a 25% increase of the total cost. Memory or bandwidth usage limits do not have a substantial effect on the total number of routes.
- Better reliability can be guaranteed by provisioning backup hubs for each PE, while the increase in cost is not substantial. The total cost, number of hubs and total memory usage is increased by less than 50% to get 2-reliability and less than 70% to get 3-reliability.
- One constraint in [9] was that a VPN could not be assigned to a hub if the PE (candidate hub) was not part of the VPN. Our experiments show that relaxing this constraint can lead to substantial reductions of more than 50% in total cost.
- Our cost model is stable, in the sense that changing several parameters to within a reasonable range does not change the assignments by much.

V. Conclusion

In this paper we focus on reducing the large service provider memory footprints of Layer 3 MPLS VPNs for the common scenario where a single provider network's memory and bandwidth resources are shared by many thousands of different VPNs. We generalize the concept of VPN Relaying, which enables routers to reduce routing tables significantly by offering indirect any-to-any reachability among PEs, to this multi-VPN setting. We formulate multi-VPN Relay hub selection as a constraint optimization problem that takes into account both memory and bandwidth constraints at each PE, the installation and maintenance cost of a hub, the cost of transporting traffic across the network, and the need for maintaining resilience to PE failures for each VPN. We develop solution algorithms with guaranteed bounds on performance and our evaluations indicate that our Relaying solutions lead to substantial cost savings.

Our multi-VPN Relaying technique is readily applicable in today's network and works in the context of existing routing protocols without requiring any changes to either router hardware and software, or to the customer's network. Network administrators can easily deploy the technique by modifying only routing configurations.

References

[1] *Sample prices of cisco 12416 components.* Google Shopping, http://www.google.com/products?q=cisco+12416&btnG=Search+Products&hl=en&show=dd/, (25.08.2008).

[2] *Sample prices of juniper m320 components.* Google Shopping, http://www.google.com/products?q=juniper+m320&btnG=Search+Products&hl=en&show=dd/, (25.08.2008).

[3] M. Bateni, A. Gerber, M. Hajiaghayi, and S. Sen, *Multi-VPN Optimization for Scalable Routing via Relaying*, in Proc. IEEE INFOCOM, April 2009.

[4] M. Bateni and M. Hajiaghayi, *The assignment problem in content distribution networks: Unsplittable hard-capacitated facility location*, in Proc. of ACM-SIAM SODA, 2009.

[5] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, *ROFL: Routing on Flat Labels*, in Proc. ACM SIGCOMM, September 2006.

[6] D. Farinacci, V. Fuller, D. Oran, and D. Meyer, *Locator/ID Separation Protocol (LISP).* Internet-Draft (work in progress), November 2007.

[7] U. Feige, *A threshold of ln n for approximating set cover*, J. ACM, 45 (1998), pp. 634–652.

[8] D. S. Johnson, *Approximation algorithms for combinatorial problems*, J. Comput. System Sci., 9 (1974), pp. 256–278.

[9] C. Kim, A. Gerber, C. Lund, D. Pei, and S. Sen, *Scalable vpn routing via relaying*, in ACM SIGMETRICS, 2008, pp. 61–72.

[10] A. Meyerson, K. Munagala, and S. Plotkin, *Cost-distance: two metric network design*, in IEEE FOCS, 2000, p. 624.

[11] S. Raghunath, S. Kalyanaraman, and K. K. Ramakrishnan, *Trade-offs in Resource Management for Virtual Private Networks*, in Proc. IEEE INFOCOM, March 2005.

[12] S. Raghunath, K. K. Ramakrishnan, S. Kalyanaraman, and C. Chase, *Measurement Based Characterization and Provisioning of IP VPNs*, in Proc. Internet Measurement Conference, October 2004.

[13] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in ACM STOC, 1997, pp. 475–484.

[14] C. Wilson, *Cogent throws down pricing gauntlet*, (2008). TelephonyOnline, http://telephonyonline.com/ethernet/news/Cogent_price_cuts_06112008/, (11.06.2008).

[15] X. Zhang, P. Francis, J. Wang, and K. Yoshida, *Scaling IP Routing with the Core Router-Integrated Overlay*, in Proc. International Conference on Network Protocols, 2006.

Appendix

Proof of Lemma 1: Take any objective which has the three terms corresponding to initialization cost, memory usage cost and bandwidth cost. The objective can be decomposed into a sum of independent terms for different hubs. For each hub, we can combine the first and third terms (initialization and memory usage) to get a linear increasing function of memory usage, say $f(m) = c_0 + c_1 m$. We assume $c_0 \geq c_1$.[4] We can replace this PE with a number of different PEs, with different capacities $m_0 = 0, \ldots, m_k = M$, with respective (new) initialization costs $f(m_0), \ldots, f(m_k)$, where M is the capacity of the PE in the original instance, and m_i's are picked such that $f(m_i) = (1 + \epsilon')f(m_{i-1})$ for $i > 0$. Thus, the algorithm picks an appropriate PE from this list and uses it to solve the instance. If it is using M' memory units, such that $m_j < M' \leq m_{j+1}$, then the algorithm will pick the copy with capacity m_{j+1} and pay $f(m_{j+1}) \leq (1+\epsilon')f(m_j) \leq (1+\epsilon')f(M')$. In addition, k is not too large. We note that $f(m_k) = (1+\epsilon')^k c_0$. The fact that $f(M) \leq c_0 + c_1 M \leq (1+M)c_0$, gives $k \leq \lceil \log_{1+\epsilon'}(1+M) \rceil$, which shows the blowup k is only polynomial in the instance size. This results in a polynomial-time reduction as desired. ■

Proof of Lemma 7: After an element e is ever covered by a set S, we cross out e from S for the rest of the algorithm. Let $a_1, a_2, \ldots, a_{\rho n}$ be the elements in the order covered. Note that each element appears exactly ρ times in this list, for the first ρ times covered. Define α_t as the density of the set covering a_t at that specific point in time. Then, the cost of our solution is $\sum_{t=1}^{\rho n} \alpha_t$. We claim that $\alpha_t \leq \text{cost}(\mathsf{OPT})/(\rho n - t + 1)$. This is done by considering the optimal solution OPT and crossing out all the *multi-elements* satisfied thus far. There are no fewer than $\rho n - t + 1$ multi-elements left. So, one of the sets in OPT should have density no worse than $\text{cost}(\mathsf{OPT})/(\rho n - t + 1)$. The bound on α_t follows. Hence, the cost of our solution is no more than $\text{cost}(\mathsf{OPT}) \sum_{t=1}^{\rho n} 1/(\rho n - t + 1)$. ■

[4]Otherwise, we can just redefine $c_0 := c_1$ which gives a factor 2 approximation of $f(m)$. However, this is an impractical case.