A Dynamic Managed VPN Service: Architecture And Algorithms

Ravi S.Ravindran[1], Changcheng Huang[1], K.Thulasiraman[2]
(1.Carleton University, Ottawa, 2.University of Oklahoma, Norman)

Abstract—VPN as a managed service enables the service provider to offer more demanding and revenue generating services. In this paper we try to tackle an important problem of service providers providing bandwidth service on demand on an IP/MPLS core network. We propose a managed VPN architecture for such a service highlighting the novelty in our architecture. We concentrate on an important aspect of service definition called the topology abstraction service and define a new problem called the VPN core capacity sharing problem that arises in this context. We propose three schemes to solve this problem borrowing from results from graph theory. As part of our simulation study, we evaluate each of these strategies with different call arrival scenarios and present the results.

Keywords: Managed VPN Service, Topology Abstraction

I. INTRODUCTION

With the increasing convergence of transport over IP routed networks, IP-VPN[1] as a managed service has gained a lot of momentum in the last few years. Traditional way to offer IP-VPN services used tunnel based overlay techniques. These VPN's were managed completely by the VPN customers themselves, hence were referred to as Customer Edge (CE) based VPN's. The service provider in this case only provided the capacity and QoS guarantees as agreed upon as part of the SLA, and is ignorant about any VPN existence. The new generation IP-VPN service also called peer based or point-to-cloud based VPN service, requires the Customer Edge (CE) devices to only peer with one or more neighboring Provider Edge (PE) devices. More recent IETF activities involved defining services over MPLS. IP-VPN service definition over MPLS is one of them. A good summary of recent IETF activities relating to L2/L3VPN standards may be found in [2]. We envision that future network services will have to address the needs of the potential bandwidth intensive applications that will arise in future. A good example of this is grid applications, where distributed processors might want capacity for a window of time to exchange huge data. In this paper we address one such issue, namely, providing bandwidth on demand dynamic managed VPN service.

We next briefly review the literature on works related to VPN architectures and SLA schemes differentiated on granularity of traffic demand specifications. The well-known PIPE model SLA tries to emulate layer 2 circuits over which IP traffic is tunneled. Here the virtual link is a pipe connecting two end points of the VPN network. The PIPE is a fixed capacity virtual circuit, and thus bandwidth is committed at any point in time. This scheme requires the VPN's to provide absolute traffic matrix demands. The pipes that traverse the VSP links and nodes can be decided by solving Multi-Commodity Flow formulation such as in [3] which proposes a multi objective formulation with the objective of optimizing resource and link utilization. The HOSE model was first proposed in [4]. It tries to alleviate some of the shortcomings of the PIPE model. In the case of a HOSE model the VPN customer specifies a set of end points to be connected with a common endpoint with end-to-end performance guarantee. RANGE model [5] allows the VPN to specify the requirement as a range of quantitative service, hence the VPN is not required to predict and specify any peak traffic requirements which are typically difficult to anticipate in bursty traffic conditions. A dynamic programmable VPN architecture that allows spawning dynamic VPN networks with dedicated router and link resources at the discretion of the VPN customer is proposed in [6]. Programmable VPN architectures assume access to physical router and link resources through open programmable interface, which may not be possible where strict trust issues exist as in case of an enterprise and an IP-VPN service provider. Also this kind of partitioning requires prior knowledge of demand matrix of the VPN. In all the above schemes the VSP is required to have partial or complete knowledge of the VPN demands, and the capacity is pre-provisioned based on the agreed SLA's. In a dynamic scenario we may encounter two cases: one where the VSP has no ability to predict future demands, and the other where the future demands are partially known. In this study we assume a lack of knowledge of a demand matrix a priori, and we assume the presence of a set of VPN customers for which the VSP does not pre-provision any resource before hand, but the provisioning takes place on demand. VPN service of this kind, if offered by the VSP, would co-exist with the traditional guaranteed VPN service. In [7][8] we explored the possibility of providing dynamic VPN service where the VSP would provide the VPN's a set of abstract topologies to choose from. These abstract topologies provide a view of the properties of the core network, which is used by the VPN clients in their path computation to determine end-to-end QoS paths. In this paper we propose this concept under the framework of a generalized dynamic managed shared VPN service. We would also like to point out that the concepts discussed in the paper, though centered around L3/L2 VPN's, are equally applicable to L1VPN's as was discussed in [7].

Summarizing the rest of the paper, section 2 proposes the idea of dynamic managed shared VPN service, discussing key elements that constitute this service. Section 3 proposes one of the problems called the VPN capacity sharing problem. Section 4 proposes three different algorithms to solve the capacity-sharing problem. Section 5 discusses our simulation scenarios and compares the three schemes proposed in section 4.

II. DYNAMIC SHARED VPN SERVICE

In this section we discuss key architectural ideas behind realizing a dynamic shared VPN service at a high level. The framework adopted for our architecture is an extension of the proposal in [1]. We begin our argument by assuming that the VSP has some portion of resource that it could share among VPN customers seeking bandwidth on demand.

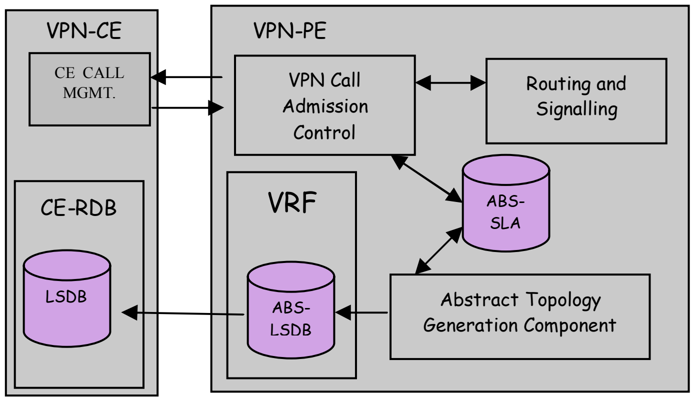

Fig 2.1, Dynamic VPN Service Components

Fig 2.1 shows the components of our architecture. The novel components of our proposal are**:** the *VPN abstract topology SLA database*, the *abstract topology generation component* and the *VPN abstract topology database* which is specific to a particular virtual routing and forwarding (VRF) instance. The abstract topology component in general deals with the management of the abstract topologies exchanged between the VSP and the VPN. The abstract SLA database is the repository for all the VPN SLAs. The abstract topology database stores VPN abstractions that are computed by the abstract topology generation component. The key idea behind providing topology abstraction is for the VPN to make a conscious decision before making a bandwidth request. These abstract graphs stored as part of the VRF extensions are part of the special abstract topology database. They are flooded by the border nodes to the VPN client nodes, which are then injected into the routing protocols like OSPF/ISIS's traffic engineering routing database, shown as CE-RDB in Fig 2.1. When the VPN client router needs to compute a route, it would apply a route computing procedure using the abstract topological information from its routing database to decide if a desirable QoS path exists. The route request is then sent to the border node, where another phase call admission control is performed to verify its adherence to pre-agreed call statistics. If satisfied the route is computed to check for the feasibility of the path before signaling phase is initiated. In [8] we introduced the notion of an abstraction SLA, and studied various abstract topologies that could be generated as part of the topology abstraction service. In this research we extend the work and use abstraction to virtually partition the core resource, without mapping the shared resource to the link or the switch resource. Even if the VSP were to pre-partition the network logically using the algorithms presented later, this information is only stored as state information in border edge nodes. This allows the flexibility to overwrite it at any moment of time, particularly in situations where a VPN needs to get priority over the resource. In [8] we had evaluated the performance of various abstract topologies and a way to provide service differentiation based on the granularity of the abstraction. In this paper we concentrate on the schemes that would be used to share the core capacity between the VPN's, which is a step prior to generating the abstract topologies. The metric we aim to abstract is the VSP's shareable core resource. In Fig. 2.6 we have captured the five steps involved in providing a shared dynamic VPN service at a very high level. These steps are from the point of view of a border node providing abstract topologies to the client. Steps 1&2 identify the set of border nodes hosting this VPN. Steps 3&4 use an abstraction scheme to generate the abstract topology, which is flooded to the respective VPN client served by the border node. This process is repeated every VPN abstract topology refresh interval, which as we stated before, is another potential abstraction parameter negotiated between a VPN and the VSP.

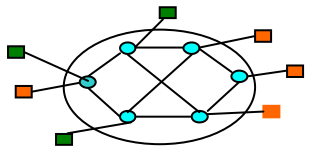

Fig. 2.2: VSP Network

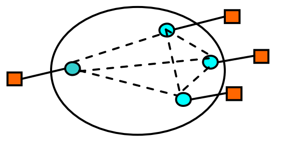

Fig. 2.3: Full Mesh Abstraction

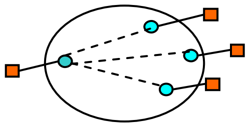

Fig. 2.4:Source Star Abstraction

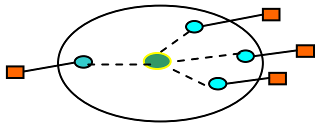

Fig. 2.5: Star Abstraction

The complexity of generating a virtual topology with residual bandwidth as the metric information depends on two key factors: the scheme used to virtualize the core resource and the abstract topology generated out of the virtualization. In [8] we concentrated on the latter part of generating virtual topologies using the widest path algorithm. This basically generated a widest path tree that was used to generate abstraction to the VPN clients. But we observed this approach to be too aggressive and that it resulted in higher call blocking. This motivated us into research for better heuristics. In the next two sections we formally define our problem and propose algorithms.

```
Abstract_Topology($G$, $B$, $VPN_{abs}$)
G: VSP Core Graph
B: Set of border node
$V_{abs}$: Set of subscribed VPN's
Step 1: For each VPN $k \in VPN_{abs}$
Step 2: Find set $PE_k \subset B$ and $CE_k \subset C$ for VPN $k$
Step 3: Apply a proposed Abstraction Scheme to
            generate Abstract Graph $G_{abs(k)}(V,E)$
Step 4: Update VPN $k$ node's $CE_k$ with $G_{abs(k)}(V,E)$
Step 5: Goto Step 1
```

Fig 2.6, Topology Abstraction Process

III. VPN Capacity Sharing Problem

We start by introducing the notations used in the rest of the paper: Given a directed graph $G(V,E)$ representing the core of the VSP, each link $e_{i,j} \in E$ is associated with total capacity of $l_{i,j,}$ and shared capacity represented as $R_{i,j}$. $l_{i,j} - R_{i,j}$ represents the resource used for guaranteed VPN services. Let B represent the set of border PE nodes, and C the set of all CE nodes connected to the VSP's network. Let the set VPN_{abs} be the set of customers subscribing to abstraction service. A border node $b_i \in B$ supports one or many of the VPN instances from set VPN_{abs}. The VPN's are connected to the border nodes through their CE nodes. For a VPN_{abs} instance k we represent the set of corresponding CE nodes as set $CE_{,k}$ and the set of border nodes as $PE_{,k}$. For a given border node $b_i \in PE_{,k}$ the set of CE nodes connected to it is denoted as $CE_{i,k}$. As part of the dynamic VPN service, each of the VPN's in VPN_{abs} is served with abstract topologies. $VPN_{abs}(b)$ is used to represent the set of VPN's hosted on border node b. For a given VPN_{abs} i we represent the abstract graph as $G_{abs(i)}(V_i, E_i)$, where V_i is the set of virtual nodes, some or all of which may map to a border node PE_i. E_i is the set of virtual links, each connecting a pair of virtual nodes $(x, y) \in V_i$. A virtual link $e \in E_i$ is associated with a vector of abstracted metric denoted as $v(e)$. Here we restrict ourselves to only one abstracted metric *i.e* virtualized core capacity. Hence for edge $e \in E_i$ connecting nodes $(x, y) \in V_i$, we have $v(e) = \{bw_{abs}[x][y]\}$. This bandwidth represents the capacity exposed by the VSP between the pair of V_i nodes connected by a virtual link. The capacity over an access link is the available physical residual capacity. The VPN's use the abstract graph $G_{abs(i)}(V_i, E_i)$ to compute end-to-end paths. Using the above notations we state the VPN capacity sharing problem as follows:

Given a set of VPNs in VPN_{abs}. Each VPN instance $i \in VPN_{abs}$ is to be provided by an abstract topology $G_{abs(i)}(V_i, E_i)$, and each virtual link is to be assigned a virtual capacity as decided by the VSP. The objective is to device a methodology for the VSP to share the VSP core resource among VPN_{abs} instances so that for a given VPN, the VSP maximizes its probability of making a correct decision of successfully computing or rejecting a path locally.

The above problem's objective has been defined from a VPN customer's perspective. It can also be framed from the VSP's perspective where the objective is to share the core capacity so as to maximize the network utilization. In case of a dynamic VPN service, maximizing network utilization is not only a function of the type of routing strategy applied in the core network, but also a function of efficiency of the scheme applied to share the shareable core capacity. A poor capacity-sharing algorithm might just be very conservative in exposing the core resource information through abstract topologies, and hence could lead to a bad resource utilization. Therefore generating an abstraction to a VPN is a two-step process. The first step involves computing a subgraph $SG_i(V_i,E_i)$ where V_i is the set of PE_i nodes and a subset of the core nodes, and E_i the subset of physical core links. The second step is to generate an abstract topology $G_{abs(i)}(V_i, E_i)$ from the sub-graph $SG_i(V_i,E_i)$. Since the metric to be abstracted is the residual capacity, the complexity of generating a topology boils down to the complexity involved in generating a subgraph for each of the VPN's. For a simple case where the subgraph is a tree, this problem boils down to generating a Steiner Tree graph with the link weights as a function of the residual capacity. This problem is known to be a NP-Complete problem, making the core capacity-sharing problem a difficult one. In the next section we propose three heuristics for the capacity-sharing problem, which is to be used in Step 3 of the abstraction process described in Fig 2.6.

IV. VPN capacity sharing algorithms

In this section we propose capacity sharing schemes that use different graph algorithms to provide abstract views of the VSP's core network. The motivation behind the abstraction schemes is to provide accurate information to the VPN customers by minimizing overlap between the virtualized core resource and at the same time maximizing the capacity multiplexing gain in the VSP's core. The key challenge of any capacity sharing schemes is to fairly share the core resource among all the VPN's and to minimize the call rejections at the associated PE nodes. We propose three approaches. The first one uses maximum capacity path for abstraction. This was suggested in [8] too. The two other novel approaches are called the max-min bound approach and tree graph approach. In the following we elaborate on each of these schemes.

Maximum Capacity Abstraction Scheme:

In this approach the VSP exposes the capacity of the widest path between two border nodes belonging to a given VPN. For a given pair of border nodes $(b_1, b_2) \in B$ belonging to a given VPN i, let $P = \{p_1, p_2 .... p_k\}$ be the set of the k paths available between the two nodes. Let $C(p_i)$ be the capacity of each path. We use $\max_{i=1...k}(C(p_i))$ as the bandwidth $bw_{abs(b1,b2)}$ between the two nodes. This problem can be solved in $O(|V|^2)$ time applying a modified form of Dijkstra's shortest path algorithm, using which a widest path tree can be computed from a border node computing the abstraction. Once the tree is computed, any desired abstraction can be generated based on the agreed abstraction SLA as defined in [8]. The total complexity of this scheme for a single VPN would be $O(|V|^2 + |ABS|)$. Here, $|ABS|$ refers to the complexity involved in generating the abstract topology. This approach is very aggressive since the same maximum capacity path is abstracted to all the VPNs. There are two reasons why this scheme might not fare well. The first is a case where one of the VPN's starts misbehaving and seeks bandwidth more aggressively. Another reason why this may not work well is when multiple calls from different VPN's arrive at the border node at the same time leading to resource contention and to higher call crank back probability. The next heuristic we propose is more conservative, which also addresses the key issue of handling multiple call from different VPN's simultaneously arriving simultaneously at a border node**.**

Mixed Bound Abstraction Scheme:

In this scheme, for a VPN k the VSP provides two capacity bounds as part of the virtual link metric, upper bound capacity (UBOUND_k) and lower bound capacity (LBOUND_k). We define UBOUND_k as the aggregate bandwidth that can be requested in between two consecutive abstract topology update instances. UBOUND_k also indicates the slice of the shared network capacity available between any two virtual nodes. This scheme is a logical fit for mesh or partial mesh type of abstractions, in which case the virtual nodes typically would map to border nodes. Here we compute the upper bound is using the max-flow computation, where the net flow splits among multiple paths. LBOUND_k on the other hand gives a single path flow capacity approximation that can be requested from the VSP. The key idea behind this scheme is to virtually expose a slice of the max flow possible between two border nodes for a given VPN. The computed max flow is shared between the VPN's as decided by their priority factor α_i for VPN i. For a given border node b, α_i satisfies, $\sum_{i \in VPN_{abs}(b)} \alpha_i = 1$. For example, for a given VPN k, and a pair of border nodes $x,y \in \text{PE}_k$, if $bw_{max}[x][y]$ is the maximum flow achievable between the two border nodes, then the VSP would expose a capacity of $\text{UBOUND}_k[x][y]$ equivalent to $\alpha_i * bw_{max}[x][y]$ to VPN i. The priority factor could be used to control the net capacity virtually allocated to a given VPN at any point of time. By default we be fair to all the VPN'S, hence assign it to the number of hosted VPN's on a border node b, i.e $\alpha_i = 1/|VPN_{abs}(b)|$.

In order to give a single path flow approximation, we compute LBOUND_k. In an aggressive mode, the $\text{LBOUND}_k[x][y]$ can be set to the capacity of the widest path. This has the drawback of higher crank back calls that are caused by multiple calls arriving simultaneously at the border node with aggregate demand exceeding the available capacity. Here we propose a less aggressive approach using M-Route flow algorithm [10] to compute the lower bound. To define loosely, M-Route flow is any flow that can be expressed as a non-negative linear sum of elementary M-Flows. M-Route flow in the context of both edge and vertex disjoint paths is defined in [10]. For our study we use the theory of M-Route edge flows. Using M-Route flow as the LBOUND the border node has a higher probability to satisfy requests simultaneously for aggregate capacity less than or equivalent to M-Route max flow. This hinges on the fact that a M-Route max flow can be decomposed into set of M-Route elementary flows. The question now is a way to choose the value of M. M is upper bounded by the maximum number of edge disjoint paths existing in a given VSP core topology, which can be obtained applying Menger's theorem. The ideal way to assign a value to M would be to analyze the call arriving pattern, and fixing it to the expected number of calls arriving simultaneously at the border node during a time window of the abstract update interval. The rationale behind this is that it is most likely that the VPN's are making requests using the same abstract topologies in this time period. Another practical way to assign M would be to initialize it to the number of VPN instances being served by the concerned border node. In order to correlate UBOUND_k and LBOUND_k for a given VPN k, the amount of LBOUND_k exposed is computed as a function of α_i, hence if $bw_{m\text{-}route}[x][y]$ is the maximum M-Route flow achievable between the two border nodes, then the VSP would expose a capacity of $\text{LBOUND}_k[x][y]$ equivalent to $\alpha_i * bw_{m\text{-}route}[x][y]$ to VPN i. An algorithm to compute a M-Route Flow, which can be found in maximum of (M-1) runs of the max-flow algorithm with a pseudo polynomial complexity of $O(M^*|V|^2|E|)$ may be found in [10]. For a given pair of border nodes x and y, we can observe that for M=1, $\text{UBOUND}_k[x][y] >= \text{LBOUND}_k[x][y]$, but this condition may not hold for M>=2. In cases where $\text{UBOUND}_k[x][y] < \text{LBOUND}_k[x][y]$, we set $\text{UBOUND}_k[x][y] = \text{LBOUND}_k[x][y]$. In cases where $\text{LBOUND}_k[x][y]$ is zero, we switch to a best effort service and set $\text{LBOUND}_k[x][y]$ to the maximum capacity value between x and y. Since the M-Route flow computation dominates the complexity, the total complexity for the mixed bound approach is $O(|B|*|M|*|V|^2|E|+|ABS|)$. Fig 4.1 shows the pseudo code to compute the UBOUND_k and LBOUND_k for a given VPN k from a border node's perspective.

```
Mixed Bound Abstraction(G, VPN(k), B_i, C_{i,k}, α_k ,M)
Step1: Find border node set $B_{i,k}$
Step 2: Init. Graph $G_{abs(k)}(B_{i,k}, E_k)$, For $\forall\, b_j \in B_{i,k}$ $e(B_i, b_j) \in E_k$,
    $\text{UBOUND}_k[B_i][b_j]=0$, $\text{LBOUND}_k[B_i][b_j]=0$
Step 3: For each virtual link $e_{x,y} \in E_k$, where $x = B_i, \forall\, b_j \in B_{i,k}$
        Set $\text{UBOUND}_k[B_i][b_j] = \alpha_k *$ MaxFlow$(G, x, y)$
        Set $\text{LBOUND}_k[B_i][b_j] = \alpha_k *$ M-RoutFlow$(G, x, y, M)$
        If($\text{LBOUND}_k[B_i][b_j] = 0$)
            Set $\text{LBOUND}_k[B_i][b_j] =$ MaxCapacity$(G, x, y)$
        If ($\text{UBOUND}_k[B_i][b_j] < \text{LBOUND}_k[B_i][b_j]$)
            Set $\text{UBOUND}_k[B_i][b_j] = \text{LBOUND}_k[B_i][b_j]$
Step 4 : Flood $G_{abs(k)}(B_{i.k}, E_k)$ to nodes $C_{i.k}$
```

Fig 4.1, Mixed Bound Abstraction Scheme

Tree Based Abstraction Scheme

Here we partition the network using trees, in such a way that there is minimum overlap between the trees computed for each VPN used to build abstract topologies. The tree approach guarantees single path abstractions, at the same time giving better guarantee of finding a path when requested for the core capacity. In order to minimize the overlapping between VPN trees, we make the edge weight as a function of the number of occurrences in a VPN tree. We explain the abstraction scheme following the steps stated in the pseudo code as shown in Fig.

4.2. In *Step 3*, before computing the Steiner graph, we initialize the link weights as function of the residual capacity. In this abstraction scheme we also introduce a new link variable called *vpnCount*(e), where $e \in E$. This variable indicates the number of times the edge has been part of a Steiner tree. We initialize the edge cost as a function of *vpnCount*(e) and the residual capacity. The idea of resetting the cost is that the links with more available bandwidth would be at lower cost compared to links with high cost and higher utilization. Since the goal is to optimize the usage of the core resource, we employ a minimum cost tree algorithm, which also correlates with the objective of preventing over subscribing of congested links. In *Step 4* we compute the Steiner tree, T_i, for a VPN i. *Step 5* shows the *vpnCount*(e_i) variable of the edge $e_i \in T_i$ being incremented indicating its occurance in the tree T_i. Making the link cost a function of *vpnCount*(e) also dissuades the future VPN tree computations from using the links used by the previous VPN tree computation. After the trees are computed, in *Step 6* we assign the virtual capacity to the virtual link of the abstract graph. For a particular VPN i, and a source root node which is also the concerned border node b_i, we summarize bandwidth between the node b_i and another border node b_j, by identifying the edge set E_i from the tree T_i that comprises the tree path. The virtual capacity of the path is the bottleneck capacity of all the edges belonging to the path. Steiner tree is a strongly NP Complete problem, but literature provides good heuristics to compute Steiner trees. We use the minimum cost Steiner tree algorithm from [9] for the tree computation. The complexity of deriving a source rooted abstract topology for a given VPN is $O(|B_k|*|V|^2+|ABS|)$.

```
Steiner Tree Abstraction Scheme(G, VPN(k), B_i, C_{i,k})
Step 1  For VPN $k$ find Border Node set $B_{i,k}$
Step 2  Initialize Graph $G_{abs(k)}(B_{i,k}, E_k)$
            For $\forall b_j \in B_{i,k}, e(B_i, b_j) \in E_k$, Set $bw_{abs}[B_i][b_j] = 0$
Step 3: For edge $e_i \in G(V,E)$,
          Set $vpnCount(e_i)=1$
          Set $Cost(e_i) = vpnCount(e_i)*(l(e_i))/(R(e_i))$
Step 4: Use a Steiner tree heuristic on nodes $B_i$ to
          generate Steiner graph $T_k(V,E)$
Step 5. For each edge $e_i \in T_k(V,E)$ and $e_i \subset G(V,E)$
          Set $vpnCount(e_i) = vpnCount(e_i) + 1$
Step 6: For each $b_j \in B_{i,k}$, Let $P = \{e_1 \ldots e_k\}$, $e_i \in T_k(V,E)$
          Set $bw_{abs}[B_i][b_j] = \min\{bw(e_1) \ldots bw(e_k))$
Step 7: Flood $G_{abs(k)}(B_{i,k}, E_k)$ to nodes $C_{i,k}$
```

Fig 4.2, Tree Graph Abstraction Scheme

V. Simulation Study

The simulation was implemented using OpNet, on a network topology of 50 nodes with an average degree of 4. 10% of the total number nodes in the core topology were chosen randomly as the PE nodes. Each of the PE nodes was configured to handle five different VPN instances. For comparing the capacity sharing schemes we use four different metrics: *Success Ratio*, *CrankBack Ratio*, *MissCall Ratio* and the *Call Performance Ratio*. *Success Ratio* is a measure of making a right routing decision using the abstraction provided by the VSP. The *CrankBack Ratio* is defined as the fraction of calls that have been cranked back because of a successful path computation at the VPN's end, but bouncing back from the border node because of insufficient capacity in the core. *MissCall Ratio* is the fraction of calls that have been rejected wrongly locally by the VPN, even when the core has sufficient resource to accept it. One can observe that success ratio and crankback ratio are correlated, and the ideal value one expects is 1 for the success ratio and 0 for the crankback ratio. An aggressive heuristic may have very good success ratio but a poor crankback record. To bring out this relationship in assessing the performance of the heuristics, we introduce the *Call Performance Ratio* metric. We define the Call Performance Ratio of the heuristic as the ratio of success ratio to the crankback ratio. So ideally, higher this value better is the heuristic. To make an objective study of the differences in these capacity sharing heuristics, for our simulations we fixed the abstract topology to be source-rooted star abstraction discussed in [8]. The simulation setup has link bandwidths for the access as well as the core initialized to 1000 units. The bandwidth call requests from the VPN client nodes were modeled as Poisson arrivals. Similarly the call holding times are negatively exponentially distributed. During the simulation it was assumed that all the core border nodes have up to date information of the core topology. The VPN abstract topology update interval was set at value less than the mean arrival rate of the calls. The bandwidth request follows a uniform distribution of [1-500]. In the graphs discussed later the x-axis traffic load is the ratio of the arrival rate of VPN bandwidth requests to the inverse of mean call holding time.

Fig 5.1 compares the Success ratio for each of the three abstraction algorithms. Here we see the mixed bound scheme performing poorly than the other two schemes, the reason being that a lot of calls were rejected locally because of the aggregate demands exceeding upper bound capacity as dictated by the VSP, even though there is resource available in the core. The maximum capacity scheme and the tree graph scheme of abstraction show a much better Success ratio compared to mixed bound scheme. Though we notice a slightly poorer performance by the tree graph approach, this is offset by a very low crank back ratio that we see later, which makes it in fact a better algorithm. Fig. 5.2 gives a comparison of the Crankback Ratio of the three schemes. As expected, we observe that the mixed bound scheme has the least Crankback ratio compared to the other schemes. The lower CrankBack ratio can be attributed to the upper bound capacity that can be used to determine the aggregate capacity the VPN could seek from the core. Hence most of the calls are rejected locally when the demands exceed the upper limit capacity. Comparing the mixed bound approach and tree graph approach, we see that the tree graph approach outperforms the mixed bound approach with regard to the Crankback ratio, which can be attributed to better abstraction strategy applied in abstracting capacity. Fig 5.3 compares the MissCall ratio of the three schemes. We see that in most cases the total number of missed calls is twice the ratio of the mixed and the tree graph schemes. The higher number of missed calls can be attributed to higher calls being rejected locally by the VPN's even though there is sufficient resource to satisfy the requests. Fig 5.4 gives a comparison of the performance metric, which gives a clearer idea of the heuristic's performance. We see that of the three heuristics the mixed bound heuristic performs the best, followed by the tree graph heuristic. But the performance of the Mixed Bound heuristic is offset by the complexity of the algorithm which is at least O(|E|) times expensive than the Maximum Capacity or the Tree Graph heuristic.

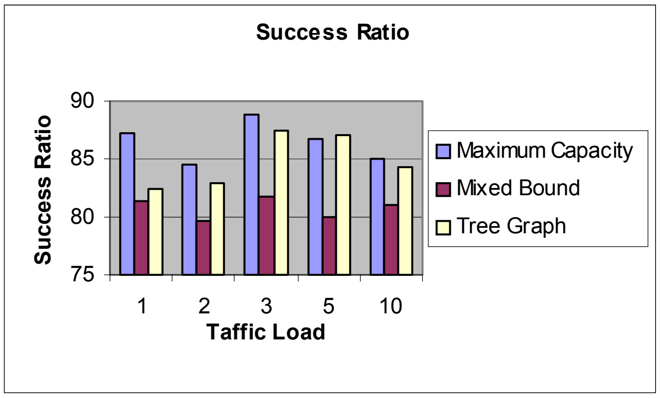

Fig 5.1, Success Ratio

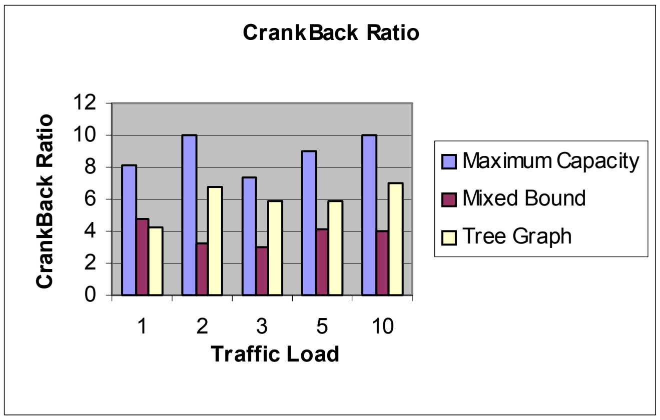

Fig 5.2, CrankBack Ratio

VI. Conclusions

In this paper we proposed a dynamic managed VPN service and noted its differences from the traditional VPN definitions. An architecture as an extension to the existing IP-VPN solution was proposed. We then defined the core capacity sharing problem in a dynamic managed VPN service context. As a way to solve this problem we proposed three heuristics. The maximum capacity abstraction, which is an aggressive way to sharing resource, has satisfying Success ratio, but its Crankback ratio is almost twice those of the other two schemes. Mixed Bound approach tries to address the drawbacks of the maximum capacity scheme by defining a virtual upper bound and lower bound for the bandwidth that can be requested from the VSP. This scheme was the most conservative of the three schemes with poor Success Ratio and high algorithm complexity. The third scheme we proposed uses Tree Graphs, using Steiner trees as a way to virtualize capacity applying a method to minimize overlap of the trees that were later used to generate abstract topologies. This scheme also faired better in all the four performance metrics in comparison to the other proposed schemes.

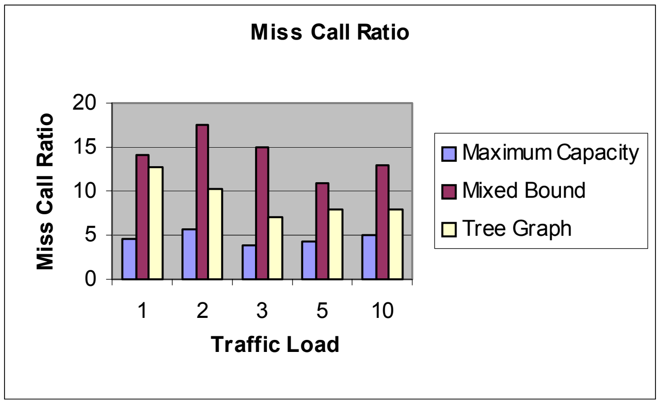

Fig 5.3, Miss Call Ratio

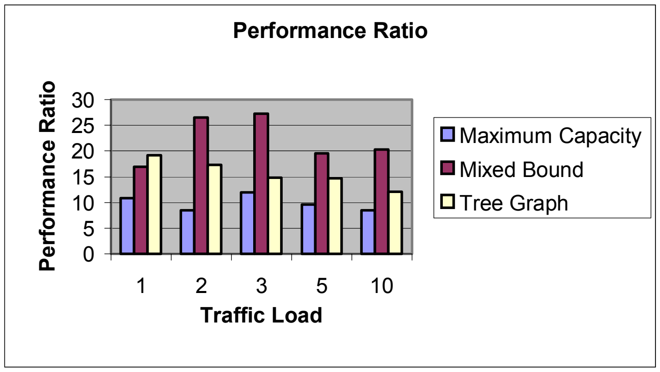

Fig 5.4, Performance Ratio

VII. References

[1] IETF draft, Eric Rosen et al,"BGP/MPLS IP VPN's"

[2] Paul Knight, Chris Lewis, "Layer 2 and 3 Virtual Private networks: Taxonomy, Technology and Standardization Efforts", IEEE Communication Magazine, June 2004

[3] Chun Tung Chou, "Traffic Engineering for MPLS-based Virtual Private Networks", IEEE, 2002

[4] N.G.Duffield, P.Goyal, A.Greenberg, P.Mishra et al, "A flexible model for resource management in VPN", in Proc. ACM SIGGCOMM, 1998, pp 95-108

[5] Ibrahim Khalil , Torsten Braun, "Edge Provisioning and Fairness in VPN-DiffServ Networks", IEEE, ICC 2000

[6] Rebecca Issacs, " Light Weight Dynamic Programmable VPN", IEEE-OPENARCH, 2000.

[7] Ravi Ravindran, Peter Ashwood-Smith et al, "Multiple Abstraction Schemes for Generalized Virtual Private Networks", IEEE-CCECE-2004, Niagara.

[8] Ravi Ravindran, ChangCheng Huang, K.Thulasiraman, "Topology Abstraction as VPN Service", IEEE, ICC, 2005.

[9] Kou, L., G. Markowsky, L.Berman, "A fast Algorithm for Steiner Trees", Acta Informatica, Springer-Verlag, 1981: vol. 15, pp.141-145.

[10] W. Kishimoto, M.Takeuchi, "On M-Route Flow in Networks", ICCS/ISITA'92.