ELSEVIER

Computer Networks 36 (2001) 137–151

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Customizable virtual private network service with QoS

L. Keng Lim, Jun Gao, T.S. Eugene Ng, Prashant R. Chandra, Peter Steenkiste *, Hui Zhang

Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Abstract

In this paper, we propose and implement Virtual Network Service (VNS), a value-added network service for deploying virtual private networks (VPNs) in a managed wide-area IP network. The key feature of VNS is its capability of providing a customer with a VPN that is customizable with management capabilities and performance properties comparable to a dedicated physical network. In addition, VNS ensures confidentiality of data and principals through the use of IPSEC. The main technique underlying VNS is the *virtualization* of routers in both control and data planes. Virtualization of the control plane enables customizable routing and signaling per VPN. On the data plane, packet forwarding and link bandwidth are virtualized. Virtualization of the forwarding mechanism on the data plane enables forwarding of traffic according to each VPN's topology and policies. Virtualization of the link bandwidth enables each VPN to have guaranteed quality of service (QoS) and customized resource management policies. We have developed a VNS prototype for deployment on the CAIRN network. The VNS prototype implements several resource management mechanisms including packet scheduling, signaling and runtime monitoring. A graphical user interface enables service providers to manage, configure and deploy VPNs remotely. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Virtual private networks; Network quality of service; Programmable networks

1. Introduction

The Internet is gradually evolving into an infrastructure for network-based services. Virtual private network (VPN) service will be one of the important Internet services. A VPN service allows a customer to build a virtual wide-area network on top of a shared wide-area network infrastructure, such as the Internet, without setting up any dedicated physical connections. There is strong economic incentive for the VPN service because of the opportunity to share a common expensive physical network infrastructure amongst multiple VPNs. The ubiquity of the Internet makes it an ideal infrastructure for providing the VPN service. Fig. 1 illustrates the situation where two different VPN topologies are created on top of the same underlying shared network infrastructure.

Various forms of private networking services have been available to enterprises for years. Initially, private networks were built using dedicated leased lines, but the cost of building a large private network using dedicated hardware is prohibitive to all but the largest corporations. Then, with the introduction of low-cost, packet-switched virtual circuit-based services such as Frame Relay and X.25, virtual private networking became possible. Unfortunately, the availability and functionality of these services is very limited. For an Internet-

* Corresponding author. Tel.: +1-412-268-3261; fax: +1-412-268-5576.
E-mail address: prs@cs.cmu.edu (P. Steenkiste).

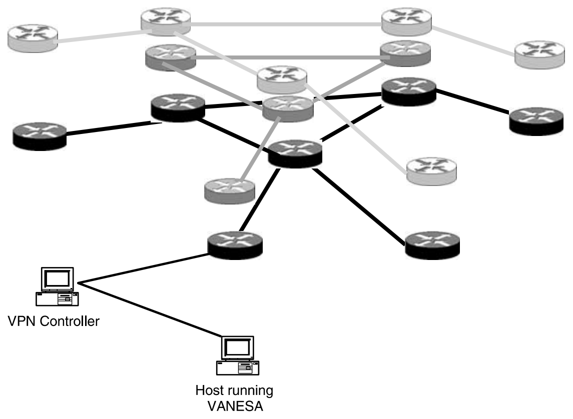


Fig. 1. Two VPNs built on top of one shared physical network in VNS. VANESA is a graphical VPN management tool. The VPN controller is responsible for carrying out commands from VANESA.

based VPN service to be a viable alternative, it must have properties comparable to that of a dedicated physical network. The service must provide mechanisms to enforce quality of service (QoS) and confidentiality of data must be guaranteed as the data travels over the common infrastructure. In addition, the service must offer each VPN with the autonomy to customize resource management.

Most Internet-based commercial VPN solutions today construct virtual links using either site-to-site IP tunnels or site-to-site MPLS paths. The configuration of the VPN topology is therefore highly restricted. The services supported are often limited to best-effort site-to-site connectivity and secure communication between sites. If QoS is offered, it is usually provided by over-provisioning network resources so that QoS service-level agreements are unlikely to be violated. Recently, some efforts such as in [5,12] use QoS strategies that require VPN traffic to be regulated at ingress nodes. The downside is that the opportunity for statistical sharing of unused resources is reduced. Another important limitation of these approaches is the lack of customizability. For example, a customer cannot control the routing of VPN traffic for load balancing or QoS routing, nor can a customer specify resource management policies in the VPN.

In this paper, we propose and implement Virtual Network Service (VNS), a value-added network service for deploying VPNs in a managed wide-area IP network. VNS is built on top of the IP layer to ensure interoperability across various layer 2 technologies (e.g., ATM, MPLS). A VPN is constructed from virtual links. A virtual link is a link abstraction connecting any two physical nodes that are in the VPN's topology. Communication over the VPN is secure, and each virtual link is allocated a guaranteed bandwidth. Moreover, unused bandwidth is shared statistically between VPNs for additional performance gains.

The key advantage of VNS is that it deploys VPNs that have a level of performance and degrees of freedom in management that are comparable to physical private networks. For instance, instead of being restricted to only site-to-site virtual links, a customer has full control of the VPN topology, and how the VPN topology maps onto the underlying network. This has two advantages. First, the topology can be engineered such that applications that are sensitive to the network topology (such as multicast applications) can achieve the best performance. Second, by carefully choosing the topology, statistical sharing of bandwidth within the VPN can be optimized. In addition to customizing the topology, each VPN can also select its own control protocols. For example, it can use a customized routing protocol that supports load balancing, policy-based routing or QoS routing. VNS also provides guaranteed QoS on each virtual link in a VPN. Moreover, because link bandwidth is virtualized using hierarchical packet scheduling, each VPN can even have its own signaling protocol (e.g., RSVP) to customize resource sharing policies in the VPN or to provide per-flow QoS to real-time applications.

The main technique underlying VNS is the *virtualization* of the control and data planes in routers. Virtualization of the control plane enables each VPN to have the autonomy to execute custom routing and signaling protocols while sharing a common physical infrastructure. Our approach to provisioning customizable control planes leverages a programmable router architecture that provides an open programmable interface [29].

In the data plane, packet forwarding and link bandwidth are virtualized per VPN. Virtualization of the forwarding mechanism enables isolation and routing of traffic according to virtual topologies. Virtualization of the link bandwidth provides each VPN with virtual links of guaranteed capacity, and the autonomy to specify its own bandwidth sharing policy. Earlier work in VPN services such as in [7,15,24,31] did not consider statistical sharing of under-utilized resources. In this work, the additional performance benefit of statistical multiplexing is achieved without compromising any bandwidth guarantees by using the fair service curve (H-FSC) [27] hierarchical packet scheduler.

Architecturally, VNS is based on the Darwin [8] router design, which is programmable and capable of virtualizing the link bandwidth. The Beagle [9] signaling protocol is used for resource allocation and control plane customization. In order to virtualize routing and forwarding, we extend the Darwin router design to allow each VPN to have its own routing protocol and forwarding table. Secure communication is achieved through IPSEC [18]. The virtual network system administrator (VANESA), a Java-based VPN management tool, provides a user interface that hides the complexity of the signaling from the user. VNS is targeted towards deployment on the CAIRN research network [1].

The rest of this paper is organized as follows. In Section 2, we examine the overall system design of VNS. In Section 3, we explain the key concept of virtualization by describing the mechanisms used to enforce virtualization of bandwidth, control plane protocols, and the forwarding mechanism. We then survey related work in Section 4 and summarize our work in Section 5.

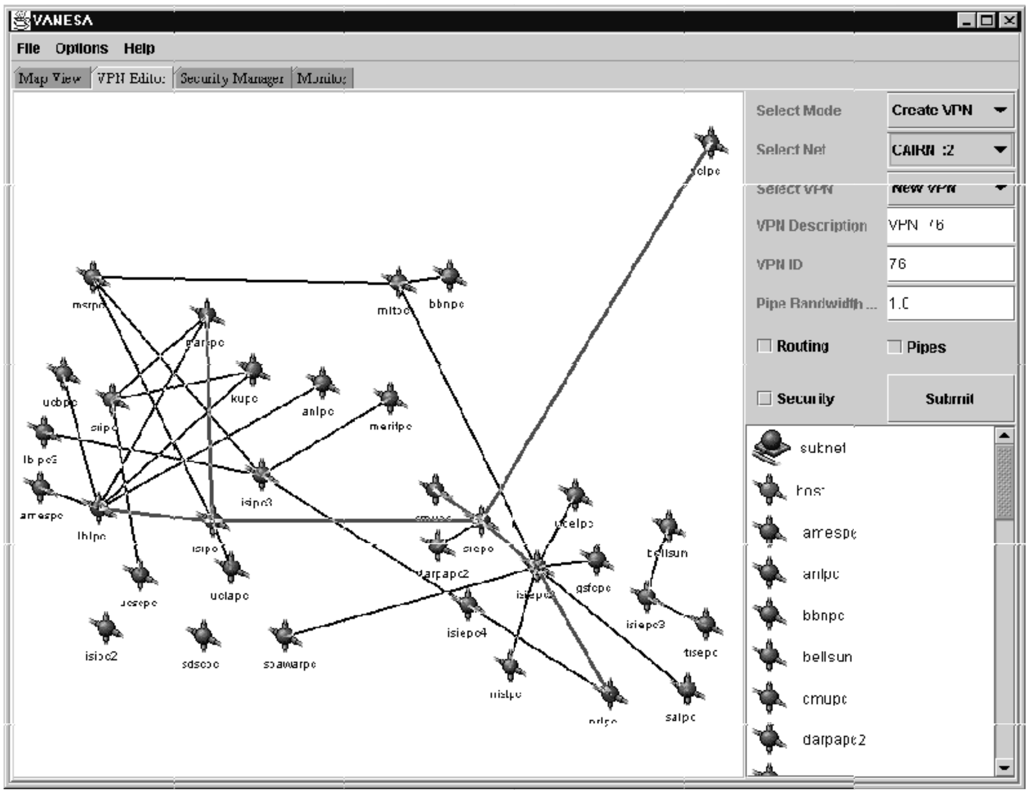
2. VNS system overview

In this section, we describe the major components of VNS and how they interoperate. A more detailed desciption of the techniques used in virtualizing routers is presented in Section 3.

2.1. Components

The main VNS components are:

1. *VANESA*: VANESA is a Java-based centralized graphical user interface for configuring and managing VPNs. Fig. 2 is a screen capture of VANESA. The idea here is similar to the concept of a software toolkit for deploying virtual networks as described in [13] by Ferrari and Delgrossi. VANESA provides a simple interface for the network administrator to configure VPN properties such as the virtual topology, bandwidth requirements of virtual links, parameters for security configuration and VPN *membership* information. Members of a VPN are described by the member end hosts' IP addresses and/or the member subnets' network prefixes. In addition, VANESA can also be used to specify custom routing and signaling protocols that are to be deployed within a VPN.
2. *VPN controller*: The VPN controller is a process that runs on a host or router that has direct access to the network where VNS is deployed. The job of the VPN controller is to act as a proxy for control messages between VANESA and routers in the WAN where VNS is deployed. This enables VANESA to be executed remotely from anywhere in the Internet. Furthermore, the complexity of the signaling required to set up the VPN is handled by the VPN controller and decoupled from the user interface. This setup is depicted in Fig. 1.
3. *Virtualizable VNS routers*: VNS routers are Darwin-based routers built on commodity PC hardware running a variant of FreeBSD Unix. Usually, a minimal PC router performs packet forwarding based on a single forwarding table and a routing daemon that does route computation. Darwin routers have enhancements such as a signaling protocol module, a sophisticated packet scheduler, packet classifier, and a programmable interface for deploying value-added services. Leveraging these existing features of Darwin, we extended the Darwin router design for VNS. Control plane and data plane resources on a VNS router are virtualized to support the unique needs of each VPN.
   In the data plane, each VPN is allocated its own resources such as link bandwidth and a forwarding table. In the control plane, a VNS router has mechanisms that enforce isolated



Select Mode: Create VPN
Select Net: CAIRN :2
Select VPN: New VPN
VPN Description: VPN 76
VPN ID: 76
Pipe Bandwidth ...: 1.0
☐ Routing ☐ Pipes
☐ Security
Submit

subnet
host
arrespc
anlpc
bbnpc
bellsun
cmupc
darpapc2

Fig. 2. A screen-shot of VANESA.

execution of custom-VPN routing and signaling protocols. Fig. 3 illustrates the virtualization of a router.

Next, we describe service provisioning in VNS by explaining the interactions between the components of the system during the design, setup, and operation of a VPN.

2.2. VPN design

We will describe the design of a VPN using the example VPN shown in Fig. 4. Each VPN's virtual topology is constructed from virtual links, illustrated as dotted lines for VPN #1 and as the lightly shaded lines for VPN #2 in Fig. 4. A router that is part of a VPN's topology is called a virtual router. For instance, VPN #2's virtual routers are A, B, D and E.

A VPN provides connectivity for end hosts or subnets identified as *members* of the VPN. In our example, VPN #2's members are subnet 10.1.1/24 attached at router A and subnet 10.2.1/24 attached at router E. The router that is the access point to the network for a VPN member is called an *edge router*. All other interior routers in the network that are part of a VPN but are not directly connected to VPN members are called *core routers*.

In order to provide QoS to virtual links and support per-VPN forwarding, virtual routers need to maintain VPN-specific information for QoS enforcement and per-VPN forwarding. In addition, edge routers must maintain VPN membership information, IPSEC security parameters, and the encapsulating IP headers to use for each VPN.

2.3. VPN setup

During the setup phase, the network administrator specifies a VPN's properties through VA-

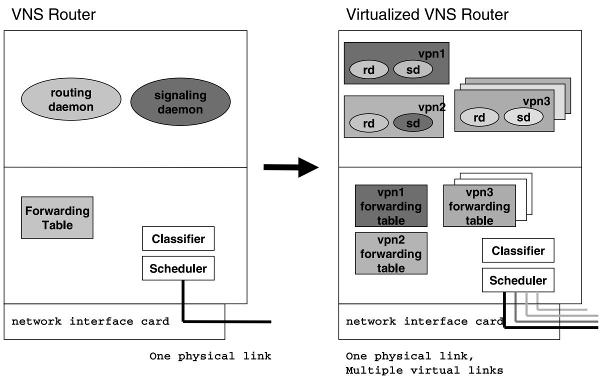


Fig. 3. A virtualized VNS router with three instances of virtual control planes and customized forwarding tables.

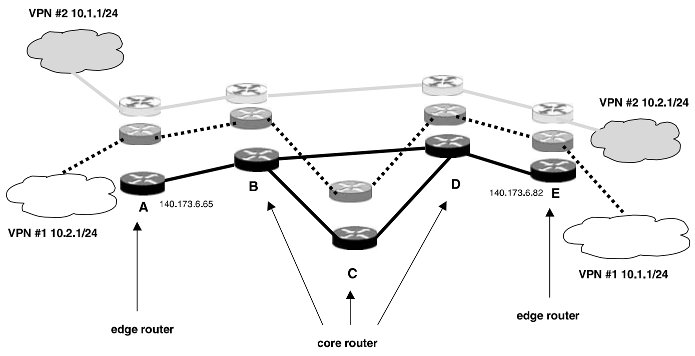


Fig. 4. Basic concepts illustrated with two VPNs.

NESA's graphical interface. These properties include the VPN's virtual topology, bandwidth requirements of the virtual links in the topology, members, local routing policies for virtual routers, security information and encapsulating IP headers for tunneling VPN traffic. After specifying the VPN description, the network administrator submits the request for setting up this VPN by clicking on the "Submit" button on VANESA's interface. Subsequently, VANESA sends appropriate setup messages to the VPN controller based on the request. There are several types of setup messages. Each is related to a request to configure one of the VPN properties. For instance, in a minimal VPN setup that has no security configuration, VANESA will be used to set up virtual links with bandwidth guarantees, dispatch membership information and configure local routing policies of routers in the virtual topology. VANESA would therefore send three setup messages to the VPN controller since each of these configuration steps corresponds to a specific type of setup request.

Upon receiving the VPN setup messages, the VPN controller initiates requests to routers in the virtual topology through the Beagle signaling protocol [9]. While it would also be possible to set up resource reservations with flow-based signaling protocols such as RSVP [4], we chose Beagle because it provides support for the allocation of resources for mesh structures such as VPN topologies. All VPN connection management tasks are handled by the Beagle daemon on the VPN controller and the Beagle daemons on the routers that are part of the virtual topology. In Fig. 5, we show this setup procedure for one of the routers that is part of the VPN.

For virtual link resource reservations, the Beagle daemon on every router of a VPN configures

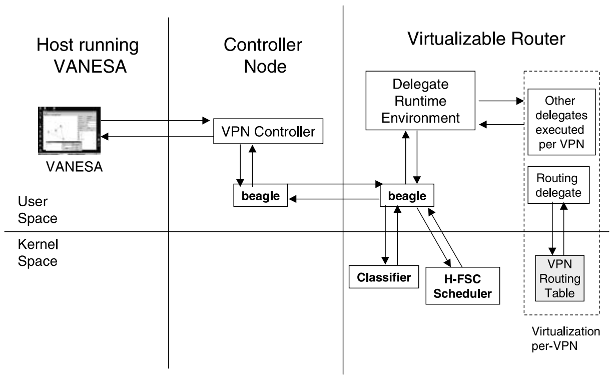


Fig. 5. Control path in VNS.

the local classifiers and schedulers of the appropriate network interfaces to reserve resources. Beagle is also used to deploy VPN-specific routing and signaling protocol modules on the routers of a VPN. The customization of control protocols is discussed in Section 3.2.

During the setup of a VPN, Beagle also performs two configuration steps that are specific to edge routers. The first step is to provide edge routers with VPN membership information and the globally unique VPN identifier (VPN-ID) that was chosen by VANESA; this information is needed so edge routers can inject packets appropriately into the VPN. The second VPN-specific step is to establish security associations between the edge routers; the security associations are used to provide authentication and encryption of the data that travels over the VPN. Both operations are described in more detail below.

2.4. VPN operation

The operation of a VPN is based on IP-in-IP tunneling, but support is provided to maintain privacy of the data and to allow per-VPN customization of packet handling inside the core of the network. We discuss the main tasks performed during the operation of a VPN in more detail in this section (Fig. 6).

As in a private physical network, we believe the basic security service a VPN should have is the confidentiality of data and principals when VPN packet flows in the core of the network. This is provided in VNS by establishing ESP [17] tunnels between the ingress and egress edge routers. This means that for any VNS data stream, cryptographic packet processing is performed at edge routers only. It can be argued that this is less secure than an alternative model that requires

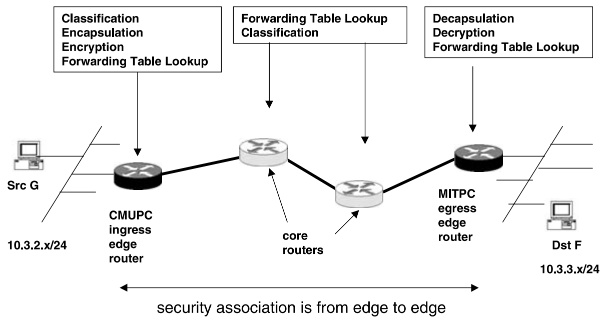


Fig. 6. Datapath through a VNS-enabled network.

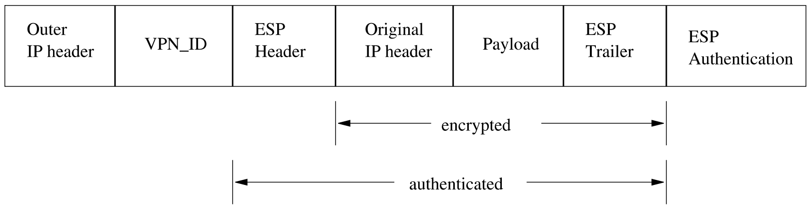


Fig. 7. VNS packet format.

re-keying at every link. Our choice in keeping the security model simple is motivated by a performance trade-off, i.e., we reduce the overhead on the core routers.

Using the membership information provided to them by Beagle, an ingress edge router can correctly identify packets that belong to a VPN. It then injects the packet in the appropriate IP-in-IP tunnel and tags the packet with the globally unique VPN-ID of the VPN. The VPN-ID is necessary because once a packet enters a VPN tunnel, the original packet is encrypted, so core routers can no longer use the header fields to identify what VPN the packet belongs to. To differentiate between packets so as to enable per-VPN forwarding and resource management, the VPN-ID is added to the encapsulating header at the ingress edge router as an `IPOPT_SATID` IP option. This approach does not support inter-VPN communication, though an easy extension to enable this would be to supplement a pair of VPN-IDs identifying the source VPN and destination VPN, respectively.

By relegating the task of tagging packets with a VPN-ID to the edge routers, we allow any end host to become a VPN member without requiring any changes. Implicitly, this limits the freedom of hosts to directly control what VPNs they participate in, since the information of what traffic uses what VPN has to be stored on the edge routers using a signaling protocol. End-hosts can be given more control by making them VNS-aware so they can insert a VPN-ID into the packets they send. This way, the end host can control more easily which specific VPN-ID they want to use for specific applications.

VPN membership is maintained at each network interface of an edge router in the form of `<VPN-ID, member src IP, member dst IP, ingress IP, egress IP>` tuples. The member source address in the tuple identifies a VPN member that is reachable through that network interface. Using the source and destination addresses of a packet, the packet is classified to be part of a VPN if it matches the `<member src IP, member dst IP>` portion of a tuple in the membership list. The packet is then encrypted by IPSEC and prepended with the corresponding VPN-ID, and at last the packet is encapsulated with the ingress and egress routers' IP addresses found in the tuple. Fig. 7 illustrates the resulting packet format. We can provide more fine grain control over what traffic enters a VPN by using additional fields (e.g., source and destination port numbers) in the filter that is used to classify packets.

When a core router receives a packet, it uses the VPN-ID to identify the VPN that the packet belongs to. It can then service the packet in a way that is appropriate for that VPN. Packet forwarding and packet scheduling (QoS) can be customized on a per-VPN basis, as is discussed in more detail in Section 3. This allows packets to be scheduled based on the policies of the VPN and forwarded according to the VPN's topology. At the egress edge router, the packet is decrypted and decapsulated. The inner packet is then examined and forwarded to the locally attached VPN destination.

We have also modified the `route`, `traceroute` and `netstat` commands for the VNS environment such that we can create the initial routing table setup and verify VPN routes.

3. Virtualization

In this section, we describe in detail how we virtualize VNS routers.

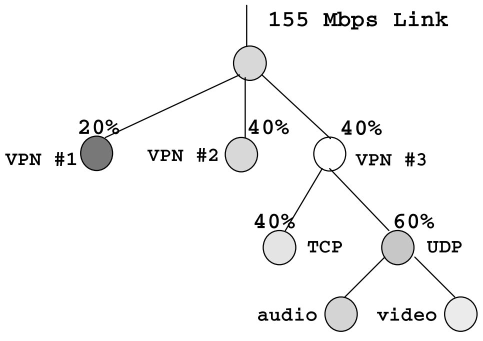


Fig. 8. Hierarchical resource tree of link bandwidth.

3.1. Virtualization of link bandwidth

Enforcement of bandwidth guarantees to virtual links is performed using a packet classifier and a hierarchical packet scheduler. For any router, we represent the division of the bandwidth of a link at the router as a hierarchical resource tree. In the context of VNS, each VPN virtual link created over a physical link is represented by a node [1] in the first tier of nodes underneath the root node in the hierarchical resource tree. A certain amount of bandwidth is reserved for each node at the VPN set up time. The effect of this is that each virtual link will have a guaranteed capacity. Fig. 8 is an example of what a resource tree at a physical link might look like with three VPNs. In this example, VPN #3 reserved 40% of the link bandwidth, which ensures that the virtual link of VPN #3 has a capacity of about 62 Mb per second. The hierarchical scheduler allows a VPN to further divide its bandwidth across the traffic classes it carries by creating a subtree. For instance, VPN #3 allocates 40% of its bandwidth to its TCP traffic in our example.

VNS uses the H-FSC [27] packet scheduler developed in the context of Darwin. An advantage of using H-FSC as opposed to other class-based scheduling discipline such as H-PFQ [2] and CBQ [14] is H-FSC's flexibility in defining and enforcing QoS on a multi-tier hierarchy. Unlike H-PFQ and CBQ, H-FSC is capable of decoupling the allocation of delay and bandwidth resources and characterizing the provided service precisely. As a result, real-time traffic can enjoy a low delay without over-reserving resources. This allows the router to have greater flexibility in resource allocation and increases resource utilization. We extended the packet classifier from the Darwin implementation to support VPN-ID-based classification. [2]

Another important property of the H-FSC scheduler is that it allows sibling nodes in the resource tree to borrow bandwidth from each other when possible. This means that if a flow inside a VPN does not use all the bandwidth that is allocated to it, other flows within the same VPN will first have the opportunity to use that bandwidth. If a VPN does not fully utilizing its capacity on a virtual link, the extra bandwidth will be shared by traffic belonging to other coexisting VPNs. This additional performance gain from statistical multiplexing demonstrates that VPNs in VNS can actually do *better* than a physical private network with fixed capacity.

3.2. Virtualization of the control plane protocols

The control plane of a commodity PC router running the Unix operating system typically consists of user-level daemons that implement various control protocols. For example, a routing daemon creates and maintains the routing table on a router, which governs the packet forwarding behavior, by exchanging routing protocol messages with peer routing daemons on other routers in the network. In a traditional (physical) network, network administrators can deploy a different routing protocol by installing new routing daemons on the routers within the network. Similarly, we would like the administrators of VPNs to be able to choose and deploy their own control plane protocols and network management policies within their VPN. To meet this requirement, the control

[1] Generally, a node corresponds to one or multiple *flows*. A flow is defined using a *flow_spec* which includes fields from IP and transport layer headers and an optional application ID.

[2] In the case of encrypted traffic, an additional flow identifier must be added to the packet header at the ingress router in order to differentiate between flows inside the VPN. This feature is not implemented in the current VNS prototype.

plane of the network that supports VPN services needs to be virtualized. In other words, the control plane can be sub-divided into multiple VPN control planes, each running a VPN-specific set of control daemons.

3.2.1. Darwin programmability support

To control the behavior of the router, a control protocol daemon needs to interact with modules in the data plane, e.g., a routing daemon must be able to update the routing table in the kernel, and a signaling daemon must be able to change the states of the classifier and scheduler. However, a traditional router is shipped as a "closed box" with a set of standard vendor protocols. It is difficult if not impossible for users to install any customized control protocols. In this project, we take on a programmable network approach to support control plane virtualization. In a programmable network, the control plane functionality of the routers can be extended dynamically by installing customized control protocols on the router. These protocol can modify the forwarding behavior of the data plane in a controlled fashion through a programming interface.

VNS leverages the programmability of the Darwin system [8] to dynamically deploy VPN-specific control protocols. In Darwin, mobile code segments, called *delegates*, can be transferred to the router and instantiated in the delegate runtime environment (DRE) using the Beagle signaling protocol. Delegates can implement control plane protocols, customized control policies or customized services. They run at user level within the DRE and change the router's behavior by controlling data plane modules, such as the classifier, routing table and the scheduler through Darwin's programming interface, the router control interface (RCI) [16]. Delegates can only modify the forwarding behavior of the traffic flows that are explicitly assigned to them.

3.2.2. Routing virtualization

We demonstrate control plane virtualization by showing that VPN-specific routing protocols can be deployed using delegates. During VPN setup, delegates implementing a selected routing protocol are installed on all the virtual routers of the VPN. The coordinated actions of these routing delegates will create VPN-specific routing tables according to the VPN's topology. This means that a virtual router will have multiple routing delegates running, each responsible for the traffic of a separate VPN.

To demonstrate the concept of routing virtualization, we use RIP-2 [19] as an intra-VPN routing protocol. For each VPN, a separate RIP-2 routing daemon will be started by Beagle. We modified the existing CAIRN routing daemon, *mrtd* [26], to support multiple RIP clouds over a single physical network. The RIP-2 specification requires all RIP messages to be exchanged at the multicast address 224.0.0.9 and port 520. In order to support multiple RIP clouds, we extend the RIP protocol to support the exchange of RIP messages at an assignable port number. The idea here is to allow a VPN to select an unused port number at the RIP multicast address and have VPN routing daemons use that port number for RIP messages. This way, we ensure isolation of VPN-specific RIP messages and prevent VPNs from *leaking* routes into each others' domain. In our implementation, port 520 remains as the port used by RIP-2 for default routing, and for each VPN deployed, VANESA assigns a unique and well-known port number to the VPN. All RIP-2 messages pertinent to this VPN will then be exchanged via this port.

Another possible approach would be to assign each VPN with a specific multicast address for RIP-2 protocol messages. This address would be chosen from the administratively scoped range (239.192/14) [21] and the only requirement is that the multicast address must be uniquely mapped to a specific VPN. This approach has the advantage that a router will only receive VPN-specific RIP-2 messages if the router is a virtual router in the VPN, but it requires that multicast is available.

The VNS approach of executing independent per-VPN routing daemons on a router offers customers the flexibility of deploying VPN-specific routing protocols. However, it has the disadvantage that it will not scale well to large numbers of VPNs. Each routing daemon will consume resources such as CPU cycles and memory, which may degrade the router's performance when it supports a large number of VPNs. When multiple

VPNs use the same routing protocol, we can reduce the number of routing daemons by deploying a single routing delegate that sends and receives all the routing messages belonging to the VPNs using the same routing protocol. The delegate then demultiplexes the messages internally to compute routes for each VPN separately.

Besides multiple routing daemons, routing virtualization also requires multiple routing tables in the data plane. We made extensions to the FreeBSD Unix forwarding mechanism so that packets belonging to different VPNs are forwarded by looking up the next hop in a VPN-specific forwarding table. We discuss the details of this extension to the forwarding mechanism later in this section.

3.2.3. *QoS management within a VPN*

A virtualized router control plane allows a VPN to deploy other VPN-specific control plane protocols. As an example, we discuss how a VPN can deploy its own signaling protocol to perform VPN-specific resource management.

As discussed earlier, each virtual router employs a hierarchical packet scheduler, i.e., the bandwidth of each link is shared in a hierarchical fashion. As shown in Fig. 8, the first level in the resource tree corresponds to the bandwidth sharing across the VPNs running on the physical link. To further exploit the merit of the hierachical scheduler, the owner of a VPN link, i.e., a node in the first tier of the resource tree, can set up more sophisticated bandwidth sharing policies for applications running within its VPN, as is illustrated for VPN #3 in Fig. 8.

To manage the resource reservations within a virtual network, a VPN may need to deploy its own signaling protocol. This can be done by instantiating per-VPN signaling deamons (e.g., Beagle, RSVP), similar to what VNS does for routing daemons. Sigaling messages must be tagged with a VPN-ID, the same way as other VPN traffic, and they will be forwarded according to the VPN topology, i.e., use the VPN forwarding table managed by the routing delegate of that VPN. The actions of the signaling daemon will be restricted to the resources of a specific VPN, i.e., the daemon will only be able to modify the resource allocations within a specific subtree of the resource tree.

3.3. *Virtualization of forwarding mechanism*

In this section, we will discuss a specific virtualization technique for forwarding packets according to virtual topologies. Conceptually, this means that we may have to forward packets destined for the same destination (egress router) different. However, FreeBSD Unix only supports single path routing [20]. This is an inherent limitation of the forwarding table radix-tree-based lookup algorithm and data structures [25].

Our solution for route isolation in the forwarding mechanism is to simply require the system to maintain a separate forwarding table for each VPN. Every forwarding table is populated with routes computed based on the VPN's virtual topology. Whenever a packet arrives at a router and needs to be forwarded, the forwarding mechanism classifies the packet. If the packet is classified to a VPN, it will be forwarded based on a route lookup using that VPN's forwarding table. Moreover, our system's routing architecture must correctly demultiplex routing messages that are exchanged between the user space and the kernel space. In the remainder of this section, we present the extensions that we made to the FreeBSD Unix routing system.

3.3.1. *Packet forwarding in FreeBSD Unix*

In a FreeBSD Unix router, the user-level routing daemon and the kernel communicate using messages [32]. The core information carried in these messages are addresses of destinations and gateways. These addresses are stored as one or more `sockaddr` structures in the payload of these messages. Fig. 9 is a simplified illustration of the forwarding mechanism in FreeBSD Unix.

Forwarding and routing are organized on the basis of different address families. Separate routing tables are used for different address families, and routing daemons inform the kernel what family of addresses they are responsible for. To make this system work correctly, routing messages must be demultiplexed to the appropriate routing daemon and forwarding table updates have to be applied to the right table. Also when there are local changes in routes or route policies, the kernel's routing subsystem must be able to dispatch these changes to the correct routing daemon.

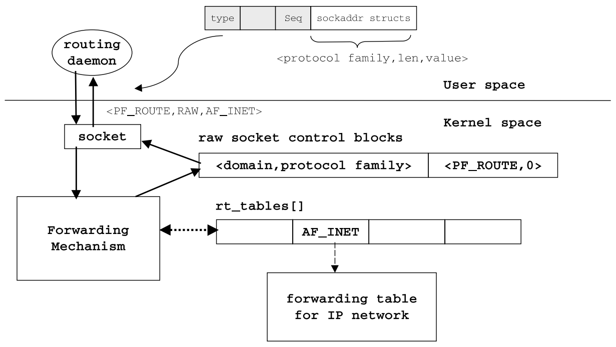


Fig. 9. Forwarding mechanism in FreeBSD Unix.

To demultiplex to the correct forwarding table, a pointer to the forwarding table is obtained by using the `sa_family` field of addresses as an index into the `rt_tables[ ]` array. Similarly, to dispatch routing messages to routing daemons, the forwarding mechanism searches through the control block list in the kernel in order to find a control block which would give a back pointer to the routing daemon. The search strategy is an exhaustive search that returns any control block that has its `<domain, protocol>` values match the key `<PF_ROUTE, protocol family of address>`.

It is clear that the above routing architecture cannot support the multiple forwarding table solution needed for per-VPN packet forwarding and routing. All addresses in the VPNs are IP addresses and will therefore have the protocol family field set to `AF_INET`. As a result, all VPNs will share the same IP forwarding table and any routing update will be dispatched to all VPN routing daemons. Any forwarding table updates will "leak" to other VPNs.

3.3.2. Routing and packet forwarding in VNS

We provide per-VPN packet forwarding by supporting demultiplexing to different forwarding tables based on the VPN-ID, as is illustrated in Fig. 10. This requires that we virtualize the various kernel data structures involved in routing and packet forwarding:

1. *Create an array* `vpn_rt_tables[ ]` *for VPN forwarding tables*: At compile time the kernel allocates two arrays; `vpn_rt_tables[ ]` for VPN forwarding tables and `rt_tables[ ]` for forwarding tables of all other protocol families. Each entry in `vpn_rt_tables[ ]` contains a pointer to a forwarding table and an unsigned integer that stores the VPN-ID for the associated forwarding table.
2. *Initialize routing daemon with VPN-ID*: When a routing daemon is instantiated, it is given the

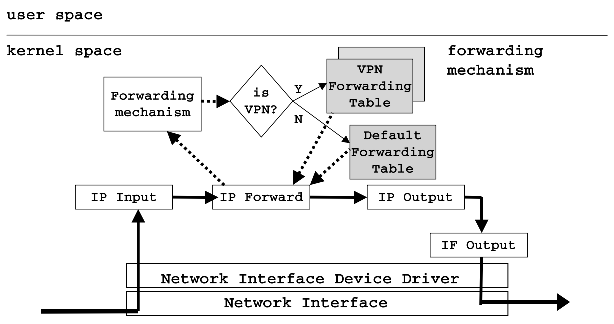


Fig. 10. Virtualization of forwarding mechanism in a VNS router's kernel.

VPN-ID of the VPN it is responsible for. This VPN-ID is used by the routing daemon to interact with the kernel.

3. *Label routing sockets with a VPN-ID*: We augmented the kernel socket structure with an additional unsigned integer field named `vpn_id`. After a VPN routing daemon has created a routing socket, it will make an additional `ioctl( )` system call to set the `vpn_id` field of the socket structure in the kernel to the VPN-ID of the VPN.
4. *Label the raw socket control blocks with a VPN-ID*: We modified the raw socket control block structure by adding a field named `vpn_id`. As in step 3, `vpn_id` is set to the VPN-ID of the VPN associated with the routing daemon.

All the functions responsible for processing routing messages entering the kernel from user space have access to the kernel socket structure of the process that generated the routing messages. As a result, we can use the `vpn_id` field in the kernel socket to associate the routing messages with the correct VPN. For example, using the `vpn_id` field as the index to the `vpn_rt_tables` array, we can easily obtain the pointer to the appropriate VPN forwarding table.

In the other direction, when routing messages need to be dispatched to the routing daemon, we cannot easily associate these routing messages with a kernel socket. In the IP domain, the forwarding mechanism uses `<PF_ROUTE, AF_INET>` as the search key to find a match from the list of raw socket control blocks. We extended the search to use the tuple `<PF_ROUTE, VPN-ID>`. Within the forwarding mechanism, we overloaded the functionality of the `sa_family` field in `sockaddr` to encode the VPN-ID in the following way. If its value falls outside the set of well-known protocol families, then we know that `sa_family` must be a VPN-ID. Consequently, in the search for the corresponding raw socket control block, the `vpn_id` field in the control block structure will be used for comparison.

Finally, an extra step is added into the packet forwarding mechanism. As shown in Fig. 10, a route lookup is performed in the `IP_FORWARD` module after the `IP_INPUT` module determines that a packet has yet to reach its final destination. In VNS, the extra step involved in this lookup consists of a classification step to determine if the packet belongs to a VPN. The classification step checks for the availability of the `IPOPT_SATID` option in the packet header. If the option exists, the packet is assumed to be a VPN packet and the option value is used as the VPN-ID. The destination address of the packet is then packed into a `sockaddr` structure and tagged with the VPN-ID. This `sockaddr` structure is then passed to the forwarding mechanism for a route lookup.

Our virtualization of the forwarding mechanism is straightforward and results in no changes to the tree-based forwarding table lookup algorithm and its associated data structures. An alternative would be revamp the forwarding table lookup algorithm and data structures, as is done in the Detour project [10] or to use MPLS. A more detailed discussion of Detour and MPLS in comparison to VNS is provided in Section 4.

4. Related work

One of the distinguishing features of VNS is that it can provide VPN services with customizable intra-VPN QoS support. To the best of our knowledge, other approaches such as the X-Bone [30], Genesis [6] and Supranet [13] are more focused on providing an overall service architecture and have not fully developed techniques for enabling per-VPN QoS. Earlier research in VPNs, such as in [7,15,24] focused on VPN services on broadband ATM networks, i.e., they developed methods for managing and mapping VPNs on virtual circuits. While some measure of QoS is attainable through dedicated virtual circuits, these ATM-based solutions typically do not allow bandwidth sharing across VPNs, since VPN flows are mapped directly to virtual circuits. This leads to a lower utilization of the bandwidth resources. Other QoS strategies that regulate traffic exclusively at the ingress router, such as in [5,12], also cannot capitalize on statistical multiplexing gains as easily.

Our approach to QoS is based on an IP layer mechanism that provides bandwidth guarantees to VPNs and has the added benefit of statistical multiplexing gained through the use of the H-FSC packet scheduler. When flows are inactive, their unused bandwidth can be utilized by other active flows.

A programmable network router architecture facilitates the virtualization of a router's control plane. Projects such as Tempest [31], Genesis [6], and Virtual Active Network (VAN) [28] represent recent efforts in using concepts of programmable networks for deploying virtual networks. Their approaches are conceptually similar to ours. Architecturally, Tempest is an ATM-based solution that uses logical entities called switchlets for isolating multiple control architectures. Genesis on the other hand, has an architecture for spawning virtual networks through the operating system services of the Genesis kernel. VAN uses a functional language [11] to specify virtual networks and virtual networks generated from VAN are deployed as application layer tunnels using UDP encapsulation. In VNS, we leverage Darwin's programmable router architecture, which provides programmability of the routers through the use of delegates and an open programming interface called RCI.

Virtualization of packet forwarding can be implemented in several different ways. To our knowledge, VNS and Detour are the only two projects that implement virtual forwarding by modifying the behavior of the forwarding mechanism in a router's kernel. VNS virtualizes the forwarding mechanism by maintaining multiple forwarding tables to isolate VPN routes. In Detour, the forwarding mechanism looks up routes in a flow database and tunnels packets using IP-in-IP encapsulation every time the packet traverses from one virtual node to another virtual node. In our approach, encapsulation occurs only once at the edge of the network and no tunneling is needed in the core of the network. Furthermore, because routing algorithms are not considered as part of the Detour framework, the flow database used for route lookups in Detour are manually configured with routes. The VNS approach allows for automatic construction of a VPN forwarding table by using a VPN-specific routing protocol.

MPLS-based VPN solutions such as in [22,23] have also been proposed. For QoS, these approaches rely on traffic engineering and regulating traffic at the ingress router using service models such as DiffServ [3]. For the purpose of labeling packets, MPLS-based solutions require the insertion of a *shim* layer between layer 2 and layer 3 protocols or overloading of existing layer 2 protocol fields. The networks where such a service is deployed must therefore be MPLS aware. In contrast, the VNS approach is an IP layer solution and is independent of the underlying link layer.

5. Summary

In this paper, we presented the design and a prototype implementation of VNS, a VPN service that is customizable and supports VPN-level QoS. We had three design goals. First, we wanted VPN support at the IP level for interoperability across multiple network technologies. Second, we wanted VPNs to be very similar to physical networks by providing the flexibility to use a variety of QoS models inside the VPNs. In fact, we want VPNs to be *better* than physical networks in the sense that heavily used VPNs can share the unused capacity of lightly loaded VPNs through statistical multiplexing. Finally, users should be able to customize the management and control functions of their VPN.

Our proposed VNS design uses IP tunnels and IP security as the basic VPN infrastructure. To support the VPN isolation and customization that is needed to meet the above goals we use three complementary mechanisms. A H-FSC scheduler provides bandwidth isolation between VPNs and allows each VPN to independently manage the bandwidth that is assigned to it. Customization of control plane functionality is provided by using a programmable router platform that supports the execution of third party control plane protocols. These customized control protocols can control the data path functions for the traffic they are responsible for through a RCI. Finally, we virtualized critical functions in the data plane. The H-FSC scheduler already supports virtualization of resource allocation (scheduling) and in our prototype we also demonstrate the virtualization of packet forwarding.

We implemented a VNS prototype based on this design using the Darwin network as a foundation. Darwin is a programmable network that uses the H-FSC scheduler and it also provides a signaling protocol that makes bandwidth reservations and installs customized control protocols. Our prototype is rich enough to demonstrate bandwidth isolation, isolation of bandwidth management, and customization of routing and packet forwarding. We plan to expand our prototype to further evaluate the possibilities of our approach, e.g., by providing support for customized signaling protocols and hierarchical VPNs.

References

[1] Collaborative Advanced Inter Agency Research Network. http://www.cairn.net.

[2] J.C.R. Bennett, H. Zhang, Hierarchical packet fair queueing algorithms, IEEE/ACM Transactions on Networking 5 (5) (1997) 675–689 (also in SIGCOMM'96).

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An Architecture for Differentiated Services, Request for Comments (Informational) 2475, Internet Engineering Task Force, December 1998.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, Resource Reservation Protocol RSVP – Version 1 Functional Specification, Request for Comments (Standards Track) 2205, Internet Engineering Task Force, September 1997.

[5] T. Braun, M. Gunter, I. Khalil, An architecture for managing QoS-enabled VPNs over the Internet, in: 24th IEEE Annual Conference on Local Computer Networks (LCN 99), Lowell, Boston, MA, October 1999.

[6] A.T. Campbell, M.E. Kounvanis, D.A. Villela, J. Vicente, K. Miki, H.G. De Meer, K.S. Kalaichelvan, Spawning networks, IEEE Network Magazine, July/August 1999.

[7] M.C. Chan, A.A. Lazar, R. Stadler, Customer management and control of broadband VPN services, in: Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management, May 1997.

[8] P. Chandra, A. Fisher, C. Kosak, T.S.E. Ng, P. Steenkiste, E.Takahashi, H. Zhang, Darwin: resource management for value-added customizable network service, in: Sixth IEEE International Conference on Network Protocols (ICNP'98), Austin, TX, October 1998.

[9] P. Chandra, A. Fisher, P. Steenkiste, A signaling protocol for structured resource allocation, in: Proceedings of the IEEE Infocomm '99, New York, March 1999.

[10] A. Collins, The Detour framework for packet rerouting, Ph.D. Qualifying Examination, Department of Computer Science and Engineering, University of Washington, November 1998.

[11] S. DaSilva, D. Florissi, Y. Yemini, NetScript: a language-based approach to active networks, Technical Report, Computer Science Department, Columbia University, New York, January 1998.

[12] N.G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K.K. Ramakrishnan, J.E. van der Merwe, A flexible model for resource management in virtual private networks, in: Proceedings of the ACM SIGCOMM 1999, September 1999, pp. 95–108.

[13] D. Ferrari, L. Delgrossi, Supranets, Technical Report CTR-96-001, Center for Research on the Applications of Telematics to Organizations and Society (CRATOS), Universita Cattolica, Piacenza, Italy, September 1996.

[14] S. Floyd, V. Jacobson, Link-sharing and resource management models for packet networks, IEEE/ACM Transactions on Networking 3 (4) (1995) 365–386.

[15] S. Fotedar, M. Gerla, P. Crocetti, L. Fratta, ATM virtual private networks, Communications of the ACM 38 (2) (1995).

[16] J. Gao, P. Steenkiste, E. Takahashi, A. Fisher, A programmable router architecture supporting control plane extensibility, IEEE Communications Magazine, March 2000.

[17] S. Kent, R. Atkinson, IP Encapsulation Security Payload (ESP), Request for Comments (Standards Track) 2406, Internet Engineering Task Force, November 1998.

[18] S. Kent, R. Atkinson, Security Architecture for the Internet Protocol, Request for Comments (Standards Track) 2401, Internet Engineering Task Force, November 1998.

[19] G. Malkin, RIP Version 2, Request for Comments (Standards Track) 2453, Internet Engineering Task Force, November 1998.

[20] M.K. McKusick, K. Bostic, M.J. Karels, J.S. Quarterman, in: The Design and Implementation of the 4.4BSD Unix Operating System, Addison-Wesley, Reading, MA, 1996.

[21] D. Meyer, Administratively Scoped IP Multicast, Request for Comments (Best Current Practice) 2365, Internet Engineering Task Force, July 1998.

[22] K. Muthukrishnan, A. Malis, Core MPLS IP VPN Architecture, Internet draft draft-muthukrishnan-mpls-corevpn-arch-03.txt, Work in Progress, expires December 2000.

[23] E. Rosen, Y. Rekhter, BGP/MPLS VPNs, Request for Comments (Informational) 2547, Internet Engineering Task Force, March 1999.

[24] J.M. Schneider, T. Preuss, P.S. Nielsen, Management of virtual private networks for integrated broadband communication, in: Proceedings of the ACM SIGCOMM 1993, September 1993, pp. 224–237.

[25] K. Sklower, A tree-based packet routing table for Berkeley UNIX, in: Proceedings of the Usenix Winter Conference, Dallas, TX, January 1991.

[26] Software available for download at http://www.mrtd.net.

[27] I. Stoica, H. Zhang, T.S.E. Ng, A hierarchical fair service curve algorithm for link sharing, real-time and priority service, in: Proceedings of the ACM SIGCOMM, September 1997.

[28] G. Su, Virtual Active Network: A White Paper. http://www.cs.columbia.edu/~gongsu.

[29] E. Takahashi, P. Steenkiste, J. Gao, A. Fisher, A programming interface for network resource management, in: Proceedings of the 1999 IEEE Open Architectures and Network Programming, New York, March 1999, pp. 34–44.

[30] J. Touch, S. Hotz, The X-Bone, in: The Third Global Internet Mini-conference in Conjunction with Globecom '98, Sydney, Australia, November 1998.

[31] J.E. van der Merwe, S. Rooney, I.M. Leslie, S.A. Crosby, The Tempest – a practical framework for network programmability, IEEE Network 12 (3) (1998) 20–28.

[32] G.R. Wright, W.R. Stevens, TCP/IP Illustrated. Vol. 2. The Implementation, Addison-Wesley, Reading, MA, 1995.

L. Keng Lim received his M.S. in Information Networking (2000) and B.S. (1993) in Math/Computer Science from Carnegie Mellon University. His research interest is in virtualization of network infrastructure for providing QoS to scalable overlay IP networks. He is currently a software engineer working on a new class of optical Internet switching systems that combines optical and IP technologies at Laurel Networks Inc.

Jun Gao received his B.S. degrees in Engineering Physics and Computer Science in 1995 from Tsinghua University, Beijing, China, and an M.S. degree in Nuclear Engineering in 1997 from University of Virginia, and an M.S. degree in Computer Science in 1999 from Carnegie Mellon University. He is currently a Ph.D. candidate of Computer Science at Carnegie Mellon. His research interests include network resource management mechanisms and customizable Internet services.

T.S. Eugene Ng received his B.S. in Computer Engineering from University of Washington in 1995 and M.S. in Computer Science from Carnegie Mellon University in 1998. He is currently a Ph.D. candidate in Computer Science at CMU. His thesis research focuses on developing a 3rd-party network service to enable connectivity across Internet networks of heterogeneous address spaces, and performance optimization techniques in a wide range of 3rd-party network services.

Prashant R. Chandra received his B.E. in Electronics Engineering from Bangalore University in 1991, M.S. in Computer Engineering from West Virginia University in 1994 and Ph.D. in Computer Engineering from Carnegie Mellon University in 2000. He is currently a network architect at Intel Corporation. His research interests are in the areas of programmable networks, signaling protocols and traffic engineering.

Peter Steenkiste is an Associate Professor in the School of Computer Science and the Department of Electrical and Computer Engineering at Carnegie Mellon University. He received the degree of Electrical Engineer from the University of Gent in Belgium in 1982, and M.S. and Ph.D. degrees in Electrical Engineering from Stanford University in 1983 and 1987. His research interests are in the area of network support for electronic services.

Hui Zhang is the Finmeccanica Associate Professor at the School of Computer Science of Carnegie Mellon University. He received a B.S. in Computer Science from Beijing University in 1988, an M.S. in Computer Engineering from Rensselaer Polytechnic Institute in 1989, and Ph.D. in Computer Science from the University of California at Berkeley in 1993. Hui Zhang's research interests are in scalable solutions for QoS and value-added services over the Internet. He received the National Science Foundation CAREER Award in 1996 and the Alfred Sloan Fellowship in 2000.