Secure and Manageable Virtual Private Networks for End-users

Kenichi Kourai† Toshio Hirotsu† Koji Sato† Osamu Akashi† Kensuke Fukuda†
Toshiharu Sugawara† Shigeru Chiba‡

†NTT Network Innovation Laboratories
NTT Corporation
3–9–11 Midori-cho, Musashino
Tokyo 180–8585, JAPAN

‡Tokyo Institute of Technology
2–12–1 Ookayama, Meguro-ku
Tokyo 152–8552, JAPAN

{kourai,hirotsu,koji,akashi,fukuda,sugawara}@t.onlab.ntt.co.jp
chiba@is.titech.ac.jp

Abstract

This paper presents personal networks, *which integrate a VPN and the per-VPN execution environments of the hosts included in the VPN. The key point is that each execution environment called a* portspace *is bound to only one VPN, i.e., single-homed. Using this feature of portspaces, personal networks address several problems at multi-homed hosts that use multiple VPNs. Information flow is separated by personal networks so that it is not mixed at multi-homed hosts. IP addressing in a personal network is independent of the other personal networks, even the base network, and therefore does not conflict with those of other networks at multi-homed hosts. In addition, personal networks provide facilities for easy bootstrapping so that the end-users can construct such isolated networks easily. Inheritance of portspaces supports the creation of new portspaces based on existing portspaces. Self-construction of personal networks enables end-users to construct personal networks without help from the base network.*

Keywords: information flow, multi-homing, VPN, overlay network, network construction, execution environment

1. Introduction

Virtual private networks (VPNs) are becoming indispensable for people who exchange private information via the Internet. VPNs protect private information from leaking to people who have no permission to access that information. VPNs are subsets of the base network, which is the existing network under VPNs, and they can have free network topology. Traditionally, VPNs have been constructed among network sites that belong to the same organization but are distributed geographically. Such a site-to-site VPN is managed by network administrators and provides users a view as if the VPN were the only network.

However, VPNs can be used for various purposes, in particular, by end-users. To enable secure remote accesses, the end-user can construct VPNs from her home to her corporation or other organizations that she belongs to. Using such VPNs, she can securely receive mail from the mail server inside her corporate network. Simultaneously, she may use online shopping sites and send private information such as her credit card number using SSL [4]. Moreover, VPNs can be constructed as protected networks that deliver multimedia contents such as music and movies. By enclosing these contents inside VPNs, content providers are released from worry that these contents may be illegally copied in open networks. End-users can select content providers and join VPNs to buy their contents. Also, VPNs are suitable for peer-to-peer networks that need their privacy protected.

As end-users use various VPNs, each host must deal with multiple networks, i.e., VPNs and the base network. Such multi-homed hosts encounter several problems. One problem is that information flow among these networks is mixed at the hosts. Unintended information flow occurs due to routing at the network layer and accidental or intentional forwarding at the application layer. This causes leakage of private information from some networks to other networks. Another problem is that IP addressing between networks may conflict when the same private addresses are used in some networks. To avoid this problem, end-users cannot simultaneously use networks whose IP addressing conflicts.

The *personal network,* which we propose in this paper, integrates a VPN and the per-VPN execution environments of the hosts included in the VPN. The per-VPN execution environment is called a *portspace* and is single-homed, i.e., bound to only one VPN. This is the key point in solving problems caused by multi-homed hosts. Using single-homed portspaces, personal networks can separate end-users' networking activities. Information flow is limited to the inside of a personal network, not only at the network layer, but also at the application layer. IP addressing is closed within a personal network so that it does not conflict with other networks. As a result, portspaces in a host can use the IP address assigned to the host in the base network.

For end-users, it is also important to make it easy to bootstrap personal networks since personal networks are completely isolated and must be therefore constructed from scratch. To do this, personal networks provide the following two facilities: *inheritance of portspaces* and *self-construction of personal networks*. Portspaces can be created based on one of the existing portspaces and they can inherit network services and a file system provided by the parent portspace. By the self-construction facility, the end-user can construct a personal network within itself, not from the base network. *Portspace translation* enables portspaces to establish VPN connections between them without help from the base network. Also, network construction is controlled based on the Chinese Wall model [2] so that portspaces with conflicting private information are not included in the same personal network.

The rest of this paper is organized as follows. Section 2 describes the problems of hosts multi-homed by multiple VPNs. Section 3 presents our personal networks and explains their components and how to construct these. The details of implementation are described in Section 4. Section 5 has the results of our experiments to determine the overheads of personal networks. Section 6 discusses related work and Section 7 concludes the paper.

2. Motivation

When end-users use their own VPNs freely, end-hosts become multi-homed. That is, end-hosts have multiple networks that end-users can use simultaneously. Even if an end-host has only one VPN, the host also uses a traditional LAN, so that the host becomes dual-homed. Until now, multi-homing of end-hosts has been often used for network redundancy or load balancing. When network failure occurs in one network, another network is activated. Multiple networks are used for load balancing so that the traffic in each network is equal. At multi-homed hosts formed by VPNs, multi-homing is used for security. VPNs are constructed to dedicate particular private information and are chosen by end-users according to the private information they want to use. While it is natural that different end-users must use different VPNs, one end-user must also use different VPNs according to the nature of her activities. For example, she would use different VPNs between when she reads business mail and when she reads private mail.

When multi-homed hosts are formed by VPNs that end-users construct, these hosts encounter several problems. First, information flow among networks, including VPNs and a LAN, is uncontrollable. If there are network routes between these networks, private information inside one VPN can be carried to another network, resulting in leaking information. This is caused if an end-host runs as a router in VPNs because it determines network routes according to a single routing table including routes in various networks. Unfortunately, even if such network routes do not exist, unintended information flow can occur via file systems or applications. For example, mail received from some VPN can be forwarded to other networks intentionally or accidentally by a mail client application. It is difficult to prevent illegal information flow among networks at the application layer. It is unknown from which network information that has been processed in applications comes.

Second, IP addressing may conflict among networks when some networks include hosts with private IP addresses. Private IP addresses are freely used within closed networks. A VPN may assign private IP addresses to the member hosts when they join the VPN or, otherwise, it may include hosts that inherently have private IP addresses. Also, a LAN may use private IP addresses. If each network is completely independent, IP addressing does not conflict even if the same IP addresses are used. However, the existence of multi-homed hosts breaks this independence by combining multiple networks at these hosts. If there are hosts with the same IP address in different networks, a multi-homed host that is connected to these networks cannot determine the unique route to the IP address. This problem seems to be solved by assigning unique IP addresses to hosts in all VPNs but this approach is not realistic. A free IP address space is not large enough, in particular, in IPv4 when end-users create multiple VPNs as they wish.

3. Personal networks

To address the problems on multi-homed hosts formed by VPNs, we propose *personal networks*. A personal network encloses an aspect of the end-user's networking activities and prevents interference with outside networks. Personal networks have the following features:

- separation of networking activities, and
- easy bootstrapping.

The second feature is not a solution to multi-homing problems but is important for end-users to construct isolated networks on demand.

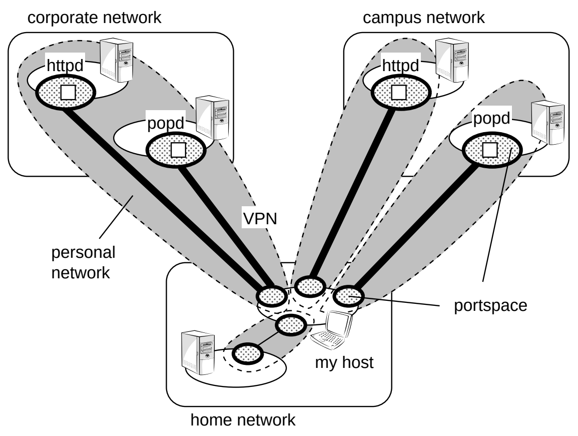

Figure 1. Example of personal networks.

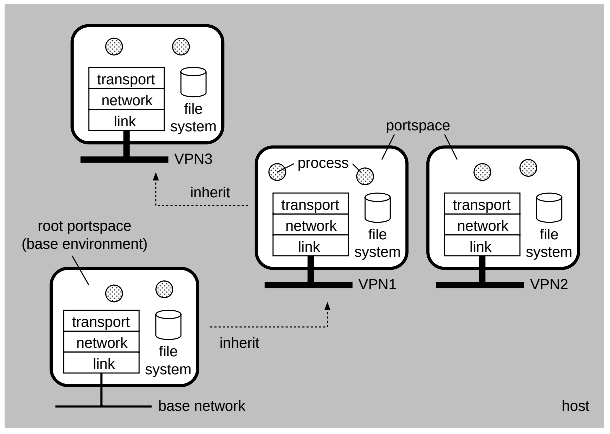

Figure 2. Portspaces and the hierarchy.

3.1. Overview

A personal network integrates a VPN and the per-VPN execution environments, called *portspaces*, of the hosts included in the VPN. While a VPN is a virtualized network, a portspace is considered as a virtualized host. Each portspace has independent namespaces for a network, a file system, and processes and is bound to only one VPN, i.e., single-homed. Figure 1 is an example of personal networks that the end-user constructs between a home network, a corporate network, and a campus network. The end-user can select one of these personal networks according to her activities.

Single-homed portspaces are the key point in solving problems inherent in multi-homed hosts. A personal network can limit information flow at the network layer by separating network routing from the other personal networks and the base network. Separation of network routing is achieved by providing an independent routing table at each portspace. A personal network can also limit information flow at the application layer. Since each portspace separates a file system and processes from the other portspaces and belongs to only one personal network, information flow via file systems or processes does not occur among portspaces that belong to different personal networks. Moreover, IP addressing in a personal network is closed and independent of other networks, even the base network. Each personal network can use even the same IP addresses with the base network for member portspaces, so that end-users can easily configure and access new personal networks.

Personal networks provide the facilities for easy bootstrapping of components, i.e., portspaces and VPNs. Easy bootstrapping is important for end-users to construct personal networks by themselves since the setup of the components is hard task. One facility for easy bootstrapping is *inheritance of portspaces*. Since a newly created portspace basically provides an empty space isolated from other portspaces for security, end-users have to set up the network, file system, and server processes from scratch. Inheriting network services and file systems of portspaces allows end-users to create new portspaces based on existing ones to reduce the difficulty of the task. In other words, they can customize existing portspaces by creating new portspaces. For example, they can replace some network services and change network configurations.

The other facility for easy bootstrapping is *self-construction of personal networks*. Self-construction means that a personal network can be constructed within itself without help from the base network. Self-construction prevents the base network from interfering with personal networks on top of it. When a portspace joins a personal network, the portspace itself needs to establish VPN connections with other portspaces in the personal network but it has no network links used for VPN negotiation until it joins the personal network. To establish VPN connection between such portspaces, portspaces borrow the network links of the parent portspaces in the inheritance hierarchy. In addition, this self-construction facility controls the membership of personal networks so that private information does not flow between organizations.

3.2. Portspace

Portspaces are isolated execution environments and have a certain hierarchy (Figure 2). The base environment, which is the original execution environment of a host, is called a *root portspace* in the hierarchy and is considered a pseudo portspace. We use the term, root portspace only in the context of the hierarchy.

3.2.1. Structure. A portspace consists of a network space, a file space, and a process space. These spaces are independent of those of other portspaces and the base environment. Applications run inside a portspace, using a network protocol stack and a file system that the portspace provides. Since portspaces are transparent to applications, existing applications do not need any modifications. Programmers can also write applications aware of portspaces for special purposes.

A network space provided by a portspace includes the following network elements:

- **Network interfaces:** A portspace provides logical interfaces to a VPN. The logical interfaces are mapped to physical interfaces in the base environment via the VPN. Multiple logical interfaces are created if a VPN is composed of a set of point-to-point network links. A loopback interface to send packets locally in a portspace is also provided.

- **IP address:** The end-user can assign a favorite IP address to a portspace. The IP address does not conflict with those of other portspaces and the base environment in a host because each portspace belongs to a different personal network and the personal networks cannot communicate with one another. The most useful assignment is to use the same IP address with the base environment. This assignment improves user accessibility because end-users and applications do not need to be aware of the difference of their IP address. When the use of the same IP address is not allowed, a portspace can use an IP address manually or automatically assigned within the personal network that the portspace belongs to. Ways to assign unique IP addresses in a personal network are beyond of the scope of this paper.

- **Routing table:** A portspace provides an independent routing table. This routing table contains network routes used only in the personal network that the portspace belongs to. The independence of routing avoids the abuse of unintended routes and prevents information flow among networks. Also, the isolated routing table avoids the conflicts of routes when the same IP address is used in different networks. In addition, dividing a traditional large routing table keeps a routing table for each VPN small, resulting in speeding up lookups of routes.

- **Protocol control blocks:** A portspace provides independent protocol control blocks (PCBs) for protocols over IP, such as TCP and UDP. PCBs bind sockets to arbitrary network ports to run new services in the portspace. Using this facility, end-users can run a different network service at the same port number with the base environment. Starting new services affects neither the other portspaces nor the base environment.

- **VPN configuration:** A portspace allows the end-user to configure a VPN bound to it. Configurable parameters are destination IP addresses of VPN connections, key expiration times, the encryption algorithm, the authentication algorithm, and so on. Note that a VPN identifier is allocated by the system, not by each portspace, because that value is used to dispatch packets to appropriate portspaces.

A portspace provides a separated file system. The end-user can describe network configurations such as DNS for a personal network in configuration files. These configuration files are read by processes running in the portspace and are reflected in these processes themselves or throughout the portspace. Since this file system is allowed to be accessed only from processes within the corresponding portspace, information flow via this file system does not occur among portspaces within the same host.

A portspace limits interaction between processes. A process can only send messages to and receive messages from processes running in the same portspace, using interprocess communication, shared memory, and signals.

3.2.2. Communication. Two portspaces at different hosts communicate using a VPN connection between them. VPN connections are established among portspaces and a set of VPN connections forms a VPN. When a portspace sends a packet, it first finds a VPN connection that can transfer the packet to a target portspace within its network space. If such a VPN connection is found, the packet is sent using the connection. When the target host receives the packet, it finds a portspace to deliver the packet according to the VPN connection from which it received the packet. If an appropriate portspace is found, the packet is dispatched to the portspace.

A personal network routes packets according to its network topology because VPN connections are not established between every two portspaces. To relay packets to target portspaces, all portspaces behave like routers and forward packets to other portspaces according to the independent routing tables. For example, a portspace can transfer packets to an Intranet web server, which can be accessed only via an application gateway, using this routing facility. Packets are first sent from the portspace to the gateway and are then forwarded to the target web server. A routing table in each portspace is distributed using existing routing protocols over IP such as RIP [6]. RIP is known not to scale in large networks but may be suitable for a personal network, which tends to be small because it is often used by one end-user.

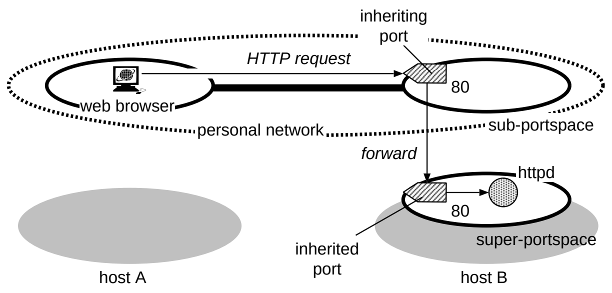

Figure 3. Example of inheritance in network services.

3.2.3. Inheritance. Portspaces can be created based on an existing portspace. The existing portspace is called a *super-portspace* and the newly created portspaces from the super-portspace are called *sub-portspaces*. In this hierarchy, a common parent of all portspaces is called a root portspace. Using the inheritance facility, sub-portspaces can borrow network services and the file system provided in the super-portspace. In addition, sub-portspaces can override or hide some services and some files. Thus, end-users can customize existing portspaces and create new portspaces easily.

Sub-portspaces can use a network service of the super-portspace by accessing a port whose number is the same as the port that the network service is bound to in the super-portspace. For example, if a web server process is running at TCP port 80 in a super-portspace, the sub-portspaces inheriting the super-portspace can also provide the same service at the same port 80. Inheritance of network services is also referred to as inheritance of network ports or inheritance of server processes since network services are closely related to network ports and server processes. As outlined in Figure 3, when the host A in a personal network sends an HTTP request to the inheriting port 80 using the VPN, the portspace that received that request forwards it to the super-portspace. After the web server process in the super-portspace handles the request, the reply is sent from the sub-portspace using the VPN that the request was sent with.

End-users can run other servers at inheriting ports in sub-portspaces to override the network services of the super-portspace. If web servers are running at port 80 both in a super-portspace and a sub-portspace inheriting the super-portspace, requests to that port in the sub-portspace are not forwarded to the super-portspace. It is handled by the web server of the sub-portspace. As a useful example, end-users can run servers that provide the same functions with servers of the super-portspace but whose versions or configurations are different. To avoid unexpected inheritance of network services, end-users can hide network services of the super-portspace when or after they create sub-portspaces. The hidden network services are not visible to the sub-portspaces. Using overriding and hiding network services, end-users can inherit only necessary network services.

The super-portspace can expose its file system to the sub-portspaces to inherit file systems. The end-user in a sub-portspace can read files from the file system of the super-portspace. Modified files and newly created files are only visible to the sub-portspace. Deleted files are hidden from the sub-portspace. In particular, inheriting the file system of the root portspace, which has fundamental executables and libraries such as ones provided in `/usr/bin` and `/usr/lib`, helps end-users construct file systems in new portspaces.

Inheritance of network services may, as a result, allow a portspace to communicate with the super-portspace and the sibling portspaces derived from the same super-portspace. We will discuss security from the view point of constructing personal networks in Section 3.3.3.

3.3. Self-construction of personal networks

Personal networks are constructed by connecting portspaces in a peer-to-peer manner. It is easy to set up personal networks from the base network, but this may allow the base network to interfere in the policy of how to construct personal networks. For example, network administrators may attach portspaces for monitoring all activities in personal networks. In addition, this setup needs a mechanism to elevate the portspaces constructed in the base network so that they run in personal networks. This may violate the integrity of personal networks because malicious users can also intrude into these using this mechanism. To avoid these security problems, personal networks are constructed without help from the base network.

3.3.1. Portspace translation. *Portspace translation* is a mechanism to enable portspaces with no network links to communicate with one another. Using this mechanism, communication between two sub-portspaces with no network links is translated to communication between their super-portspaces with network links. To associate these two communications with each other, portspace translation rewrites the IP addresses and the port numbers of packet headers based on the translation table. This is similar to NAPT [17] if we consider sub-portspaces as end-hosts and super-portspaces as NAPT boxes. The major difference is that portspace translation not only considers IP addresses and port numbers but a portspace identifier.

Figure 4 shows an example of communication between sub-portspaces with no network links. When a UDP packet is sent from port 500 in a local sub-portspace to port 500 in a remote sub-portspace, the source and destination port numbers in the packet header are rewritten to 1025 and

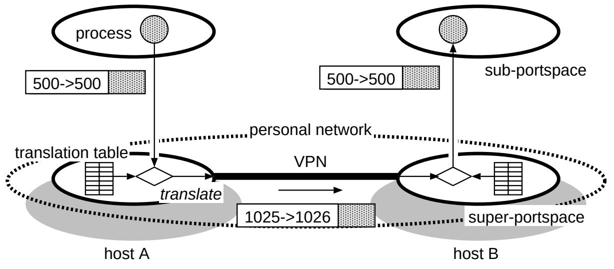

Figure 4. Example of portspace translation.

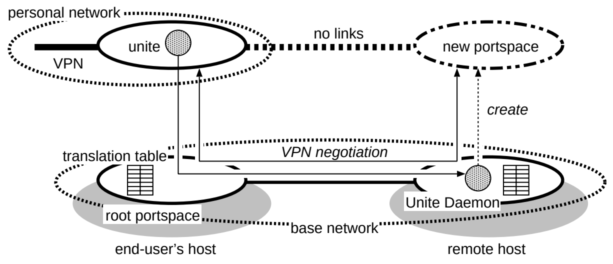

Figure 5. Unite operation.

1026, for example. Then, the packet is sent using a VPN of a local super-portspace. When a remote super-portspace receives the packet, it rewrites the packet header so that the source and destination port numbers are 500 and 500 again. Finally, the packet is forwarded to an appropriate sub-portspace. The use of portspace translation is limited to portspaces with no network links since this mechanism can violate the integrity of personal networks if portspaces that belong to different personal networks can communicate with one another.

3.3.2. Operations for self-construction. Personal networks have two operations for self-construction: *unite* and *reunite*. These operations connect two portspaces using portspace translation to construct a personal network. To enable these operations, every portspace has the *Unite Daemon* running in it.

Using the `unite` command, the end-user can attach remote hosts to the personal network that she resides in. Figure 5 shows unite operation. In the first stage, `unite` interacts with the Unite Daemon running in the root portspace at a target remote host and makes it create a new portspace. This communication is done using portspace translation only at the end-user's host because `unite` runs in the personal network separated from the remote host. In the second stage, `unite` connects the newly created portspace to the portspace where `unite` is executed. To establish VPN connection between the two portspaces, they exchange the keys for it. This communication is done using portspace translation at both hosts. They also add a new route to their routing tables.

Using the `reunite` command, the end-user can attach the portspace that she resides in to a remote personal network. Reunite operation is basically the same as unite operation. The difference is that the personal network authenticates the end-user to ensure that her portspace has the permission to participate in the personal network. The authentication is done by the operating system of the remote host at the first stage of negotiation. After that, the Unite Daemon in the target portspace at the remote host is called to set up a VPN.

3.3.3. Enforcing the Chinese Wall model. Depending on how personal networks are constructed, inheriting network services causes unintended information flow among personal networks and the base network. When a portspace inherits the network services of the super-portspace, the portspace uses server processes running in the super-portspace. This means that the server processes are shared among the super-portspace and all the sub-portspaces inheriting those services. As a result, information flow occurs via the server processes among the networks that these portspaces belong to. For example, when a web server is inherited, data submitted with CGI programs by end-users in one network is saved in the local file system by the server. If end-users in the other networks browse the data through dynamically created web pages, it is passed from one network to other networks.

One way to address this problem is that all networks that include the inherited server processes belong to the same network domain. We define a network domain as a range where private information is allowed to circulate. If all networks are constructed in such a way, information flow among these networks does not violate the integrity of any network. In a wide sense, these networks are loosely coupled and form a large pseudo personal network. Such network construction is realistic. Consider an inherited web server in a corporate network. The end-users can access the private information of the web server within the corporate network. They are also allowed to access the private information from their personal networks that include the inherited web server. Their personal networks naturally belong to the same network domain with the corporate network.

To enforce such network construction, personal networks manage the membership of portspaces based on the Chinese Wall model [2], where entities are only allowed access to information that does not conflict with what they already possess. We define the conflict of private information as a state where some private information belongs to network domains different from the others. Based on this

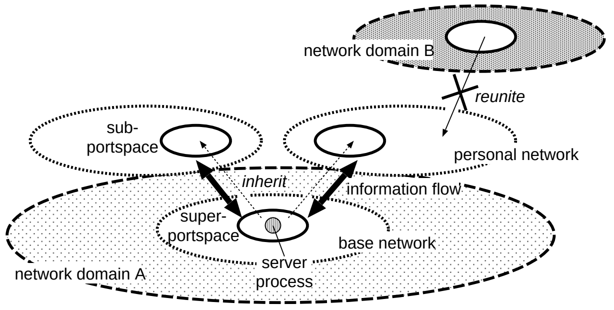

Figure 6. Construction restrictions based on the Chinese Wall model.

definition, a portspace is allowed to join a personal network as far as private information that the portspace possesses does not conflict with what the personal network possesses. Figure 6 shows an example that the reunite operation fails between different network domains.

4. Implementation

We have implemented the Persona system based on FreeBSD 4.7. Persona consists of an operating system providing portspaces and VPNs and middleware for membership management. In the current implementation, IP security (IPsec) [10] is used to form VPNs.

4.1. Portspace

A portspace is created by issuing a system call from a process. The process and the descendant processes belong to the created portspace. When all processes within the portspace are terminated, the portspace is destroyed.

4.1.1. Network management. We have modified the operating system kernel so that network processing is done with appropriate data sets such as a routing table and IPsec databases, according to portspaces. To dispatch network processing, a portspace identifier is given to sockets and network buffers (mbufs). The portspace identifier of a socket is set by a process that creates the socket while that of an mbuf is set depending on what data the mbuf contains.

IPsec tunnel mode is used for communication between portspaces. When a process in a portspace sends a packet, Persona looks up an entry in a routing table of the portspace. If there is a matching entry, Persona encapsulates the packet and passes it to the base environment. Likewise, if there is a matching entry in the routing table of the base environment, the packet is carried to a target host with the destination IP address according to both the routing in the personal network and that in the base network. When the target host including the destination portspace receives the packet, Persona dispatches it to an appropriate portspace using the security parameter index (SPI) contained in the IPsec header. The value is considered as a VPN identifier and is unique between every two hosts.

Inheritance of network services is implemented by modifying the protocol control block (PCB) lookup function, which is called to find a PCB entry corresponding to a socket that a packet is delivered to. Our PCB lookup function first attempts to find a PCB entry at the portspace that received a packet. If no entry is found there, lookup continues at the super-portspace. If no entry is also found at the root portspace, lookup fails. Thus, a request packet sent to an inheriting port is delivered to the socket of the server process running in the super-portspace. The socket preserves the identifier of the original portspace that the request packet is sent to. Based on the identifier, the socket returns a reply packet using the IPsec connection that the request packet was sent with.

4.1.2. File system management. Portspaces construct an independent file system based on the chroot mechanism in Unix. Chroot changes a subdirectory in the base environment to the entire directory in a portspace. To achieve independence even from the base environment, we have modified this mechanism so that the base environment cannot access the subdirectories used by portspaces. However, inherited file systems are implemented using the union file system in FreeBSD. The union file system enables a subdirectory to be mounted above the existing file system in such a way that both directory trees remain visible. The processes read files from the lower or upper file system and they write files to the upper file system. Since mount operation is applied to the whole system in traditional Unix, we have modified it so that a mounted file system is only visible to a specific portspace.

4.2. Chinese Wall

To express private information that portspaces possess, we use labels corresponding to the network domains that the private information belongs to. In the current implementation, the label is described by subnetwork address and mask in the base network. We assume that private information is allowed to be accessed within a specific subnetwork, which is assigned to an organization such as a corporation, a department, and a group. The root portspace is labeled according to the subnetwork. If a portspace inherits a super-portspace, the label is the same with that of the super-portspace. If a portspace does not inherit any super-portspaces, the label is null. A null label means a network domain with no private information. Spoofing of a label is

Table 1. Round-trip latency (μsec) and throughput ($Mbps$) in three network constructions.

	TCP		UDP	
	latency	throughput	latency	throughput
Base network+IPsec	132.40	91.13	126.63	94.00
Personal network	134.43	91.07	128.12	93.85
Personal network+inheritance	134.75	91.04	128.34	93.80

detected by comparing the IP address of the host where a portspace is created with the subnetwork described in its label. If the IP address is not included in the subnetwork, we can decide that the label is spoofed. For more rigid check, the X.509 certification framework [3] can be used.

To enforce the Chinese Wall model to the construction of a personal network, our system compares the labels of two portspaces when they are connected by unite or reunite operation. Comparing the two portspaces is sufficient to check the integrity of the complete personal network because our algorithm guarantees that all portspaces in the personal network have the same label. The join succeeds if one has a null label or if both have the same label, otherwise it fails. If one portspace has a null label, it is rewritten by the label of the other portspace so that these two portspaces have the same label. If both have null labels, these labels remain null.

5. Experiments

Personal networks incur overheads of IPsec and portspaces, compared with the base network. Since the overheads of IPsec change largely depending on the encryption strength that end-users need, we focused on the overheads of portspaces, which we introduced in this paper. We used two PCs each of which had a single 1.4 GHz Pentium III-S processor with 512 MB memory and an Intel Pro/100+ network interface card. These PCs were connected via a 100baseT Ethernet switch. We configured the IPsec protocol so that only the ESP protocol [9] with the NULL encryption algorithm, which does not encrypt or decrypt data, was used.

We first measured the throughput and round-trip latency of both TCP/IP and UDP/IP using the `netperf` benchmark program [7]. `Netperf` was run in three kinds of network constructions. These were (1) a base network with IPsec, (2) a personal network, and (3) a personal network with portspace inheritance. In the third construction, we inherited `netserver`, which is the remote-side program for `netperf`, at the server-side portspace. The per-packet data size sent in measuring latency was 1 byte.

Table 1 shows the round-trip latency and throughput. Using portspaces increases latency increase by 1.5% at maximum while throughput declines by 0.1%. This overhead is caused by looking up routing tables twice, checking the portspace translation table, and dispatching packets to an appropriate portspace. Using inheritance increases latency by 0.2% at maximum. CPU utilization in measuring TCP throughput was 16.1% in a base network with IPsec while that was 20.3% in a personal network despite the use of inheritance.

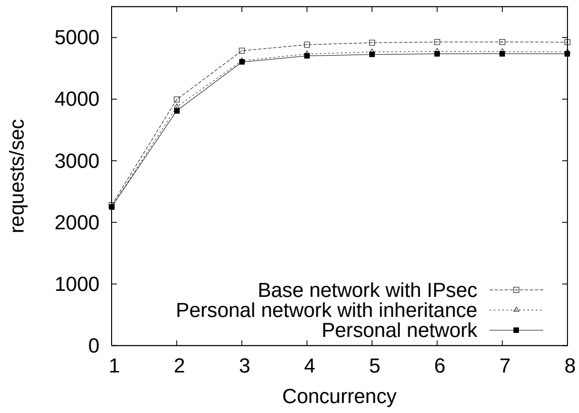

Figure 7. Changing performance of `thttpd` in three network constructions.

We then measured the performance of the `thttpd` web server [14] using the Apache benchmark program (`ab`) [1] in the three constructions. To establish the relation between the server's CPU utilization and server performance, we conducted this experiment as we changed the concurrency of requests from `ab`. The concurrency was adjusted by the number of processes that sent requests concurrently. `Ab` requested an HTML file of 0 byte so that the overheads for network processing were maximized.

Figure 7 shows changes in `thttpd` performance. Comparing a personal network with a base network with IPsec, when we set the concurrency of requests to 1, its degradation in performance was 1.1%. At this time, the server's CPU utilization was 51.7%. However, as concurrency increased, the CPU load became higher and degradation in performance increased. When we set the concurrency to 3, the server's CPU utilization reached 100% and performance decreased by 3.9%. This result suggests that the overheads of portspaces become apparent at overloaded hosts. Meanwhile, the performance of a personal network with inheritance was a little better than that of a personal network without inheritance. This is caused by the fact that `thttpd` running in the super-portspace does not access the union file system but directly access the Unix file system. Access to the union file system incurs overheads because the system needs to access two file systems, i.e., the union file system and the Unix file system. Based on our measurements, writes in the union file system were about 10% slower while the overheads of reads were negligible.

6. Related work

6.1. Virtual networks

Most virtual networks ignore the fact that hosts are shared in multiple virtual networks but some do consider problems in multi-homed hosts. Virtual Internets [20] introduce independent environments with virtual network interfaces into a host. The environments are connected to specific virtual networks via an internal router inside the host. The internal router controls the connections between environments and virtual networks so that an environment can switch multiple virtual networks to support fault-tolerant and persistent connections. Our personal network also introduces execution environments called portspaces. The major difference is that portspaces are used to insure security of information flow. To separate information flow, a portspace is always bound to only one virtual network. In addition, a portspace virtualizes not only network interfaces but the other network elements and the file system.

Router partitioning [12, 16] incorporates routing with VPNs at routers. To support routers that forward packets from multiple VPNs, one routing table per VPN is provided. Per-VPN routing tables allow the IP addresses of virtual network interfaces to overlap between VPNs. VNS [12] distinguishes VPNs by a VPN identifier inserted into every packet. Scandariato et al. adds a VPN identifier to each network interface for VPNs and it switches routing tables according to the identifier of an incoming network interface [16]. Our personal networks also achieve routing virtualization at intermediate hosts that work as routers. Moreover, end-hosts have multiple routing tables because we assume that end-hosts can manage multiple VPNs. The major difference between end-hosts and routers is end-users' applications. In routers, end-users' applications are not running and so information flow at the application layer does not occur.

A DVPN [15] is aware of operating system processes at each host. A process is created within one DVPN and has a DVPN identifier. Based on the DVPN identifier, a process gets its own virtual host name and virtual IP address through the modified `gethostname` and `gethostbyname` functions. The virtual IP addresses are globally managed by the available address service. DVPNs are similar to personal networks in that a process is only aware of other processes in the same VPN. In DVPNs, however, a process is also aware of the other processes belonging to different DVPNs in the same host. In personal networks, on the other hand, processes belonging to different networks are isolated by portspaces.

To make it easy to construct virtual networks called overlay networks, the X-Bone [19], which is used by Virtual Internets, provides high-level interfaces such as a web-based GUI and a program-controlled API. Using parameters received through these interfaces, the Overlay Manager automatically discovers available hosts and routers and configures them. While the X-Bone constructs overlay networks using the Overlay Manager, personal networks are constructed in a peer-to-peer manner. Therefore, end-users can join private hosts that cannot gain access directly from the Overlay Manager to personal networks.

6.2. Virtual hosts

There are many techniques to create virtual hosts. The chroot system call confines files and directories that processes can access to a subdirectory of the whole file system. Jail [8] extends chroot to virtualize a network space and a process space to some degree. An independent IP address and host name are visible to the processes inside a jail environment. A process cannot access other processes outside a jail environment. Clonable network stacks [24] provide independent network stacks from the network interface layer to the application layer and independent file systems based on chroot. This is the same with portspaces except for inheritance of portspaces and integration with VPNs. Zap [13] introduces a pod abstraction, which provides a virtualized view of the operating system to a group of processes. Different from our portspaces, pods translate names in existing namespaces and do not provide new namespaces. The virtual hosts created using these techniques are lightweight and are isolated from other environments more or less. However, it is unsuitable for end-users to create these virtual hosts dynamically since they need to set up their file systems and/or networks from scratch.

Virtual operating systems such as User Mode Linux [5] and virtual machines (VMs) such as VMware [22] run different operating systems, called guest operating systems, on top of the host operating system. End-users can configure the guest operating systems as completely independent hosts. Some of these systems enable new disk images for virtual hosts to be created from the existing image. Additionally, in VMware, end-users can take a snapshot of a running VM, including the disk image and the memory image, and thereby can create a new VM from the snapshot. However, the new VM conflicts with the original VM in terms of an IP address. Moreover, the performance overheads of guest operating systems are high although this has recently been reduced [18, 23].

These techniques for virtual hosts are different from portspaces in two respects. First, all of them are closed within one host and do not cooperate with other virtual hosts via networks. Since such virtual hosts assume the use of the base network, they have to have globally unique IP addresses to provide network services to the Internet. On the other hand, portspaces are integrated with a specific VPN and can thereby communicate with each other even if portspaces do not have globally unique IP addresses. Second, traditional virtual hosts have been managed from the base environment, which can access their file systems because these are constructed on top of the base file system. In personal networks, no users, not even administrators, in the base environment can access the file systems of portspaces.

7. Conclusion

In this paper, we proposed a personal network, which integrates a VPN and portspaces using the VPN. A portspace is bound to only one VPN and is a single-homed execution environment. By confining information flow to a personal network, separation of information flow is achieved. Moreover, inheritance of portspaces and self-construction of personal networks enable end-users to construct personal networks more easily. Although our current implementation of personal networks uses IPsec for VPNs, our method can be implemented using other VPN technologies such as L2TP [21] and VLAN [11].

One future direction is to deal with private information that belongs to multiple network domains simultaneously. In the current design, a personal network is allowed to include private information that belongs to only one network domain to insure security of information flow. However, not all private information is secret. We are seeking a way of loosening the independence of personal networks in a secure manner. Another direction is to introduce QoS to personal networks. We plan to combine network QoS such as bandwidth and host QoS such as CPU.

References

[1] Apache HTTP Server Project. Apache HTTP Server Benchmarking Tool. `http://www.apache.org/`.

[2] D. Brewer and M. Nash. The Chinese Wall Security Policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[3] CCITT. Recommendation X.509: The Directory – Authentication Framework, 1988.

[4] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, 1999.

[5] J. Dike. A User-mode Port of the Linux Kernel. In *Proceedings of the 4th Annual Linux Showcase & Conference*, 2000.

[6] C. Hedrick. Routing Information Protocol. RFC 1058, 1988.

[7] R. Jones. Netperf Benchmark. `http://www.netperf.org/`.

[8] P. Kamp and R. Watson. Jails: Confining the Omnipotent Root. In *Proceedings of the 2nd International SANE Conference*, 2000.

[9] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, 1998.

[10] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[11] LAN MAN Standards Commitee of the IEEE Computer Society. Virtual Bridged Local Area Networks (802.1Q), 1998.

[12] L. Lim, J. Gao, T. Eugene Ng, P. Chandra, P. Steenkiste, and H. Zhang. Customizable Virtual Private Network Service with QoS. *Computer Networks*, 36(2–3):137–151, 2001.

[13] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 361–376, 2002.

[14] J. Poskanzer. Tiny/turbo/throttling HTTP Server. `http://www.acme.com/software/thttpd/`.

[15] O. Rodeh, K. Birman, M. Hayden, and D. Dolev. Dynamic Virtual Private Networks. Technical Report TR98–1695, Cornell University, Computer Science, 1998.

[16] R. Scandarioato and F. Risso. Advanced VPN Support on FreeBSD Systems. In *Proceedings of BSDCon Europe*, 2002.

[17] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663, 1999.

[18] J. Sugerman, G. Venkitachalam, and B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the USENIX 2001 Annual Technical Conference*, 2001.

[19] J. Touch and S. Hotz. The X-Bone. In *Proceedings of the 3rd Global Internet Mini-Conference at Globecom'98*, pages 59–68, 1998.

[20] J. Touch, Y. Wang, and L. Eggert. Virtual Internets. Technical Report ISI–TR–2002–558, Information Sciences Institute, 2002.

[21] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer Two Tunneling Protocol "L2TP". RFC 2661, 1999.

[22] VMware, Inc. VMware. `http://www.vmware.org/`.

[23] A. Whitaker, M. Shaw, and S. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002.

[24] M. Zec. Implementing a Clonable Network Stack in the FreeBSD Kernel. In *Proceedings of the USENIX 2003 Annual Technical Conference*, pages 137–150, 2003.