Supernets and snHubs: A Foundation for Public Utility Computing

Germano Caronni, Tim Curry
Pete St. Pierre, Glenn Scott

Supernets and snHubs: A Foundation for Public Utility Computing

Germano Caronni, Tim Curry, Pete St. Pierre,
and Glenn Scott

SMLI TR-2004-129 March 2004

Abstract:

The notion of procuring computer services from a utility, much the way we get water and electricity and phone service, is not new. The idea at the center of the public utility trend in computer services is to allow firms to focus less on administering and supporting their information technology and more on running their business. Supernets and their implementation as hardware devices (snHubs) are our approach to make networks part of the public utility computing (PUC) infrastructure. The infrastructure is a key to integrating and enabling such "remote access" constituencies as B2B, out-sourcing vendors, and workers who telecommute in a safe and scalable manner. We have designed, developed, and deployed a prototype whose viability is now being demonstrated by a small deployment throughout Sun Microsystems.

M/S MTV29-01
2600 Casey Avenue
Mountain View, CA 94043

email addresses:
germano.caronni@sun.com
tim.curry@sun.com
pete.stpierre@sun.com
glenn.scott@sun.com

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@sun.com>.All technical reports are available online on our Website, http://research.sun.com/techrep/.

Supernets and snHubs: A Foundation for Public Utility Computing

Germano Caronni, Tim Curry
Pete St. Pierre, Glenn Scott
Sun Microsystems Laboratories
Seclabs@Sun.COM

20th April 2004

1 Vision of Public Utility Computing

The notion of procuring computer services from a utility, much the way we get water and electricity and phone service, is not new. The idea at the center of the public utility trend in computer services is to allow firms to focus less on administering and supporting their information technology and more on running their business. Supernets and their implementation as a hardware device (snHubs) are our approach to make networks part of the public utility computing (PUC) infrastructure. The infrastructure is a key to integrating and enabling such "remote access" constituencies as B2B, out-sourcing vendors, and workers who telecommute in a safe and scalable manner. We have designed, developed, and deployed a prototype whose viability is now being demonstrated by a small deployment throughout Sun Microsystems.

1.1 Background

Computer networks have transformed the way corporations do business. These networks permeate all aspects of business, from the design and engineering departments through legal, finance, and executive management. Because these networks are central to daily operations they must be simultaneously secure and highly available.

In an effort to reduce overhead costs, companies are leveraging these networks to enable employees to telecommute. Since access to information is important in any job, these networks must not only be secure, but they must also be accessible from any remote location. The ability to gain secure access to these networks also provides the means for enhanced collaboration between geographically diverse groups.

One solution being deployed to provide secure access to corporate information is the Internet Portal. The portal approach utilizes standard web server technologies such as HTTP and SSL to provide a browser-based interface to corporate data. HTTPS, along with custom written Java applets can provide a minimal level of access to information. Unfortunately, this solution suffers two major limitations.

The first constraint is the requirement that remote workers learn a new set of tools. While some corporate applications have migrated to native web-based interfaces, there are many legacy applications that must

be rewritten, and employees subsequently educated, to operate in a portal environment. Secondly, these portals become both potential bottlenecks during high traffic periods as well as single points of failure. The deployment of multiple portals can ease the potential impact of outages, but still suffer from a lack of automated fail-over and load balancing solutions, with users often needing to configure each portal through which they connect.

An alternative to the portal-based approach is the Virtual Private Network (VPN). VPNs have the advantage of giving a remote user an environment that more closely approximates being directly on a corporate LAN by encapsulating all traffic and sending it, encrypted, through a VPN gateway. The VPN solution does, however, still exhibit the same bottleneck and single point of failure issues associated with portal-based solutions. Additionally, all traffic between two nodes must pass through the VPN gateway, even in situations where both nodes are outside of the corporate network.

Because of these remote access constraints, to achieve the efficiencies information systems provide, many businesses are required to build and maintain an expensive enterprise network infrastructure and wide area networks. This effort may require expertise in areas unrelated to the company's core business. The resulting solutions are often highly specialized, complex and difficult to maintain, despite the fact that many companies are facing the same set of issues. In these situations, the ability to out-source the design, implementation and management of such an infrastructure is highly desirable.

Previous attempts to achieve secure company networks have produced a variety of ad hoc solutions. Many of these solutions center on the installation and maintenance of monolithic firewalls. While firewalls address many of the issues associated with securing access to company networks, firewall-based solutions have inherent problems. Firewalls attempt to enforce security policies on communication traffic entering or leaving a network policy domain. Most significantly, firewalls cannot provide a fine enough granularity of traffic control. Firewall implementations are designed to filter based on host and port information. In a truly dynamic, distributed world, access needs to be provided based in user identity, not machine identity.

The goal of Public Utility Computing is to provide the technical foundation for out-sourcing a network infrastructure. The result is a network that can provide access to information in a secure, distributed manner, independent of the physical topology. In addition, support costs are minimized through centralized definition and administration of access policies.

1.2 What is Public Utility Computing?

Public Utility Computing describes a paradigm that enables an organization to out-source all aspects of it's computing infrastructure. Ideally, corporations should not need to manage a large, highly specialized computing infrastructure to accomplish simple business functions. Just as they currently contract for electrical, telephony and even financial services, network infrastructure is rapidly becoming a commodity item.

The three basic infrastructure services are communication, storage, and computation. Of these services, secure communication is the key that enables distributed storage and compute solutions to be built. With secure communications in place, storage and compute services can then be used to build other network services. Most basic services such as E-mail, database manipulation, spreadsheet calculation, code compilation and web surfing can be described in terms of a set of stored data and the computations performed on the data. By building a model for secure data storage and secure computation, connected by secure communication, secure end-to-end services can be implemented.

In addition to the security within these three components, a computing infrastructure must be able to provide an environment where communication between components is secure. In order to accommodate remote collaboration and mobility of users and applications, this security cannot be dependent on physical proximity of components. Data encryption between components becomes a crucial component of the Public Utility Computing model.

Finally, a Public Utility Computing infrastructure must be manageable. In this model, the relationships between users and services may be dynamic. The infrastructure must provide a means for managing these relationships without requiring massive effort on the part of end users. A Public Utility Computing based infrastructure must allow an organization to dynamically grow and contract as the business climate changes. In this way, the network scales and adapts to the needs of the organization.

This architecture allows a service provider to securely deliver services to multiple organizations using a single set of computing resources. By securing not only the communications between components, but also securing the contents within a component (such as a storage farm), a single resource may be shared across multiple unrelated organizations. The resource is shared, though the individual contents are protected.

The basic components of Public Utility Computing can be summarized as:

1. Communication
2. Storage
3. Computation
4. Management

At this point, it becomes clear that the foundation of Public Utility Computing is the secure, dynamic communication between members of an organization. We use the term *Supernet* to describe this secure communication between a set of administratively associated users.

2 Supernet Requirements

As explained above, a Supernet is all about secure and scalable communications in an easily manageable framework. Several aspects need to be considered, and will be discussed in the following subsections.

2.1 Security

First and foremost, communications between members of the Supernet must be secure. In the PUC model, many organizations may utilize a shared resource for communication. Therefore, the contents of communications between members of a single organization must be kept private. It must not be possible to eavesdrop on these communications. Likewise, unauthorized parties must not be allowed to join a private network.

In effect, the members of a Supernet should appear to be a single, private network. The fact that this virtual network may span multiple, physically disparate networks is irrelevant. Due to the fact that membership of the Supernet may change quickly, special care needs to be taken to manage the relevant security information efficiently.

2.2 Usability

As with any form of secure communication, it is necessary for a user or system to initially join a Supernet. Aside from initiating a new session, a user should not need to be concerned with whether an application is running within a Supernet context. The behavior of system components should operate in a consistent manner at all times.

The most common problem in deploying new technologies is their impact on legacy systems. For the Supernet model to be successfully deployed, individual applications should operate without modification. The networking interfaces to a Supernet enabled host have thus been made consistent with those of the existing host operating system.

A Supernet should have the appearance of a normal, local area network. Members of the network community are not required to use special naming conventions to communicate with resources in the Supernet. Users' views of the Supernet are always the same, regardless of their physical location in the network.

2.3 Scalability

If Supernets are to be viable as replacement for corporate intranets, they should be highly scalable. Because Supernets are intended to function in IP networks which are inherently scalable, Supernets should be capable of scaling across hundreds, if not several thousands of nodes.

Care must be taken in the design of individual Supernet components. In particular, features of the Supernet architecture, such as key management and admission control, should take care not to introduce bottlenecks or single points of failure.

2.4 Mobility

Members of a Supernet are able to join the virtual network from any location. The Supernet architecture must support transparent access from any location, while providing a consistent view of the network. The logical topology of the network does not need to mirror the physical location of its member components.

The Supernet architecture must also support the ability to connect a single physical system to multiple Supernets. At least in the full Supernet model, if not in the snHub prototype, a system may simultaneously exist in multiple Supernets. As a consequence, the Supernet architecture must assure that processes communicate solely within the logical networks to which they have been authenticated. A system assures that only authenticated processes have access to traffic in the Supernet.

2.5 Administration

A common issue among security solutions is the administrative burden they impose on system operators and administrators. A Supernet should be implemented in a manner that access controls and key/certificate administration is minimized. In general, the more components that can be centrally administered, the lower the total cost of ownership.

The most important administrative consideration is that security not impose a burden on the end-user. Secu-

rity solutions that require significant effort by end-users often result in these users attempting to circumvent the security measures, often exposing other vulnerabilities and lowering the overall effectiveness of the solution.

3 Supernet Concepts and Components

Supernets are built through the introduction of a new layer of abstraction. This Supernet layer sits on top of the existing IP networking layer. It includes its own addressing structure and security services that protect all data transmitted by the network layer. This approach has been used to implement infrastructure services such as virtual private networks, mobility and other tunneling based solutions.

With Supernets, unlike VPN solutions, multiple tunnels exist simultaneously. Where VPNs require all traffic to pass through a single pipe and VPN gateway, Supernets allow the establishment of multiple pipes that directly connect members of the Supernet.

For a more extended discussion as to *why* Supernets have been designed like this, please see [CKSS00].

In our Supernet implementation, we use the terms *nodes* and *channels* to refer to the these abstractions that are layered over an existing machine and its physical network connections.

3.1 Nodes and Channels

A node is any entity that is identified by a network address in a given Supernet. A node may be a process, process trees, or any group of processes that share the same user identity within a single host. A node is an entity defined by an IP address within a Supernet. Nodes may participate in multiple channels at any given time, but may only be members of one Supernet.

Using this model, it becomes clear that for multiple nodes to exist on a single system, multiple addresses may be assigned to a single physical interface. Because the IP addresses being assigned are tunneled over existing IP infrastructure, applications running on a Supernet node can function without modification. The Supernet abstraction does not, however, require the use of IP addresses.

A channel is a communication abstraction that defines the association between Supernet members through a shared encryption key. Communications packets within a Supernet are always associated with only a single channel. As such, all datagrams transmitted within a channel can only be created or interpreted by nodes that are members of that channel.

Each channel is protected from all other channels using cryptographic mechanisms. The use of cryptography allows the creation of multiple trust domains within a single Supernet. This model allows a single Supernet to be created for a large organization, while allowing a finer granularity of resource control within the organization. A channel appears, much like a classic Ethernet, as a shared medium. In this model, every node appears to be a single hop from every other node, regardless of physical network topology.

A Supernet is constructed from one or more channels. A channel can be viewed as a mechanism for delivering a secure service. Nodes in a Supernet have access to all the services that are provided by the channels to which they are given access. It is envisioned that each service may be provisioned on a per-node basis. Membership in a Supernet does not guarantee access to all channels.

In summary, a Supernet is a collection of nodes under a single administrative domain. These nodes share a given network address space as assigned by the Supernet administrator. Channels within a Supernet enable nodes and services to be further segregated within the Supernet.

3.2 Supernet Configuration

The Supernet architecture provides for explicit authentication, admission control and audit functionality. Nodes may only become members of a Supernet in a strictly controlled fashion, thus preventing unauthorized access. Both authentication and admission control are governed by the *Supernet Admission Service (SAS)*. In addition to authorization and admission control, the SAS serves as an auditing mechanism. There may in principle be multiple SAS servers that need to be tightly synchronized as to what participants are admissible into a Supernet.

Every Supernet must have a configuration in which the basic set of Supernet services are configured. All of these operational parameters are configuration parameters of the SAS. This data is used by the SAS to control the following aspects of the Supernet.

3.2.1 Supernet Definition

A Supernet definition consists of a Supernet name, a set of channel names, and address assignment information. The Supernet name definition is a simple mapping of a string based value to the internal Supernet ID. Likewise, channel definitions provide for the mapping between internal channel identifiers to be mapped to string values. In addition to this mapping, virtual address resolution information is defined.

The Supernet definition involves the identification of addressing information. Supernet addressing information includes:

- a range of DHCP addresses
- addresses that are statically bound to specific nodes
- the virtual address for the DNS server

3.2.2 Communication Channel Definition

Every channel is assigned an identification number that must be unique within its Supernet. This ID is defined to map to a string value. Each channel must also have have crypto algorithms, key manager and address resolution information defined.

3.2.3 Key Management

Every channel must have a key manager associated with it. The Key Manager information is assigned on a per-channel basis. An address and port number pair are used to define the location of the Key Manager. The details of Key Management operations are described in section 3.4.

3.2.4 Virtual Address Resolution

Virtual Address Resolution is handled through a daemon that implements the Virtual Address Resolution Protocol (VARP). For a Supernet, the VARP service is defined by the following values:

- DNS resolvable name or IP address
- an IP port number
- a Time-to-Live value for VARP address mappings
- the virtual address of the VARP daemon
- the secure channel to use for VARP requests.

Section 3.3 discusses the VARP protocol in greater detail. For further background on VARP, also see [SL99].

3.3 VARP

A Supernet channel is a communication abstraction that fixes an association between Supernet members through a shared key. Communication packets that live within Supernets are always associated with exactly one channel and they are encapsulated in datagrams that can be interpreted and transmitted on the global IP (Internet Protocol) network.

Each node in a Supernet is assigned a unique IP address. Since these IP addresses are only legal within the Supernet context, a mapping must occur between the IP address and channel ID pair and the actual IP address of the physical interface on the target machine. The mapping from virtual IP address to the underlying physical network is accomplished through the *Virtual Address Resolution Protocol (VARP)*.

The VARP service maintains the mapping of tuples (Supernet ID, Virtual Address) to their associated actual IP Address. Any node that is part of a Supernet is authorized to request mappings of the form (Supernet ID, Virtual Address) from the VARP server for a given Supernet. In order to communicate with the VARP Server, each VARP client needs to know the VARP server's virtual address, real IP address, and the VARP channel ID. During the admission control process, the VARP client is provided with all the required information to communicate with the VARP server.

The VARP server is a distributed service. A Supernet may contain multiple VARP servers, which communicate to synchronize VARP mappings. A VARP client may communicate with any VARP server, minimizing the chance of a Supernet node becoming isolated because virtual address resolution fails.

3.4 Group Key Management

As previously discussed, all network communications between nodes on a channel must be encrypted and authenticated. One of the most difficult aspects of deploying encryption technologies is that of key management. For this reason, the Supernet provides a key management system based on *Versakey* for nodes in the Supernet. Versakey provides highly efficient and scalable key management to large communicating

groups with frequent membership changes. More background information about Versakey can be found in [CWSP98].

The purpose of key management is to provide the *Network Security Layer (NSL)* of authorized client machines with the currently valid encryption and authentication keys. All machines from which a node has joined a given channel will hold the same key to encrypt traffic for that channel. The NSL itself owns two keys: packet keys which are used to encrypt and authenticate the actual data packets, and a master key, which is used to encrypt the packet key.

The packet key may be different on each machine, and may change at any time. To make this possible, each encrypted packet on the wire also holds the encrypted packet key.

The master key for the NSL corresponds to the *Traffic Encryption Key (TEK)* in the key management components. There are additional *Key Encryption Keys (KEKs)* that are used to allow for the efficient change of group membership for each channel. For each Supernet channel there exists one server that manages a tree of keys, and the leaf nodes of this tree correspond to actual clients. The key at the root of the tree is the TEK which needs to be changed whenever a member is removed from the group or joins the group. The keys at the leaf nodes are shared only between the server and the respective client. Intermediate keys in the tree are shared by smaller and smaller sub-groups of clients and the server.

As implemented within a Supernet, key management is comprised of three components, namely the Key Management Server, Client, and Daemon. Those will now be explored in more detail.

3.4.1 *Key Management Server* (KMS)

There is one KMS per channel in a Supernet. Most often, it will be found on the same machine as the Supernet Admission Server (SAS). The KMS communicates with the SAS, where it accepts and processes add and remove requests. The KMS also holds connections to all Key Management Clients (KMC). After an initial admission request from the KMC to the KMS, all communication is unidirectional. As such, there is little possibility of the KMS becoming a bottleneck, unless many clients try to join at exactly the same time. Additionally, the failure of a KMS in the Supernet may cause other KMS servers to become overloaded. It is possible to replicate the KMS server via its external state database.

For each state change, the KMS writes a static file containing the Supernet combo and a unique serial number. In this way, the SAS is able to restart the KMS and have it recover its former state should the KMS fail or the machine need to be re-booted. As a consequence, this state file exposes keying material.

Whenever the SAS requests the removal of a client, the KMS performs a series of internal key changes, and then sends out a message to all clients. This message is sent out separately to each of the clients if they are connected to the KMS via TCP, or is sent as one multicast UDP datagram if they are “connected” via UDP. The message is not explicitly authenticated, although authentication stubs are provided in the code and the communication protocol that is being used. As such, the message could be forged by any group member, thereby disrupting group communication. After having sent the message, the KMS confirms the removal of the client to the SAS. All clients will re-key, and the KMC of the removed client will terminate.

Whenever the SAS requests the addition of a client, the KMS again performs a series of internal key changes, growing the tree of keys if necessary, and sends out a message to all existing clients. Finally, it returns a magic string to the SAS. This magic string contains the contact address by which the new KMC can reach the KMS, and it also contains an initial leaf KEK that the KMS has assigned to the client. The protection of

this initial leaf key is important, as its knowledge would allow an arbitrary party to impersonate that client when communicating with the KMS, and be able to capture all traffic in the group as long as the client is a member of the group.

3.4.2 *Key Management Client* (KMC)

On each machine where a node joins a Supernet, the attachment will spawn a KMC for each channel that the node joins. One client holds keying material for one single channel and virtual address only, so it is strictly limited to serve as keying agent for one particular Supernet node. When launched, the KMC receives a magic string from the KMS via its command line or via a pipe, and immediately establishes a TCP connection to the local *Key Management Daemon (KMD)* and the remote KMS. The remote KMS receives an admission request from the KMC, and responds with a branch of keys, encrypted with the initial KEK. In this way, the KMC receives a current TEK. The TEK is forwarded to the KMD, together with the Supernet and channel identifiers, for which it is to be used.

Whenever the KMC receives a key change message from the KMS, it will forward the resulting TEK to the KMD. Should contact between the KMS or KMD be lost, the KMC will enter a loop, and try to re-contact its two peers once per minute. If it succeeds in re-contacting the KMS, it will restart with an admission request, using the keying material initially provided. Once it enters the recovery loop, the KMC will only terminate if explicitly killed or if instructed by the KMS to exit. If it is able to reach a KMS, the re-admission request may fail, for example, because the particular Supernet session is no longer active.

3.4.3 *Key Management Daemon* (KMD)

The KMD is started upon system boot or initialization. There is one KMD per system that wants to process Supernet traffic, similar to VARP. Upon start-up, the KMD immediately connects to the NSL and flushes any previous state information. All KMC's running on a system connect to a single KMD, and feed it key changes for a particular channel.

Whenever the NSL receives a packet that has been encrypted/authenticated, it checks its local cache of keys to see if a packet key is available for the Supernet and channel in use. If there is none, it issues a request to the KMD, which checks its local database. If the Supernet and channel in question are unknown, for example, there is no matching KMC registered with the KMD, the request will be ignored and the pending packet will eventually be discarded. If the Supernet and channel are known and the channel has a valid TEK assigned to it, the KMD processes the request. This request is either for a new packet key, in which case the KMD generates the new random packet key, and also provides the encrypted packet key, encrypted in the current TEK, or the request is to decrypt an encrypted packet key that has been received from a remote system.

If the KMD terminates and is restarted, the currently running sessions should recover within a short while, when the KMC's that were connected to the old KMD re-connect to the new KMD, and provide it with valid keying material. This functionality provides reasonable fault recovery on the client side.

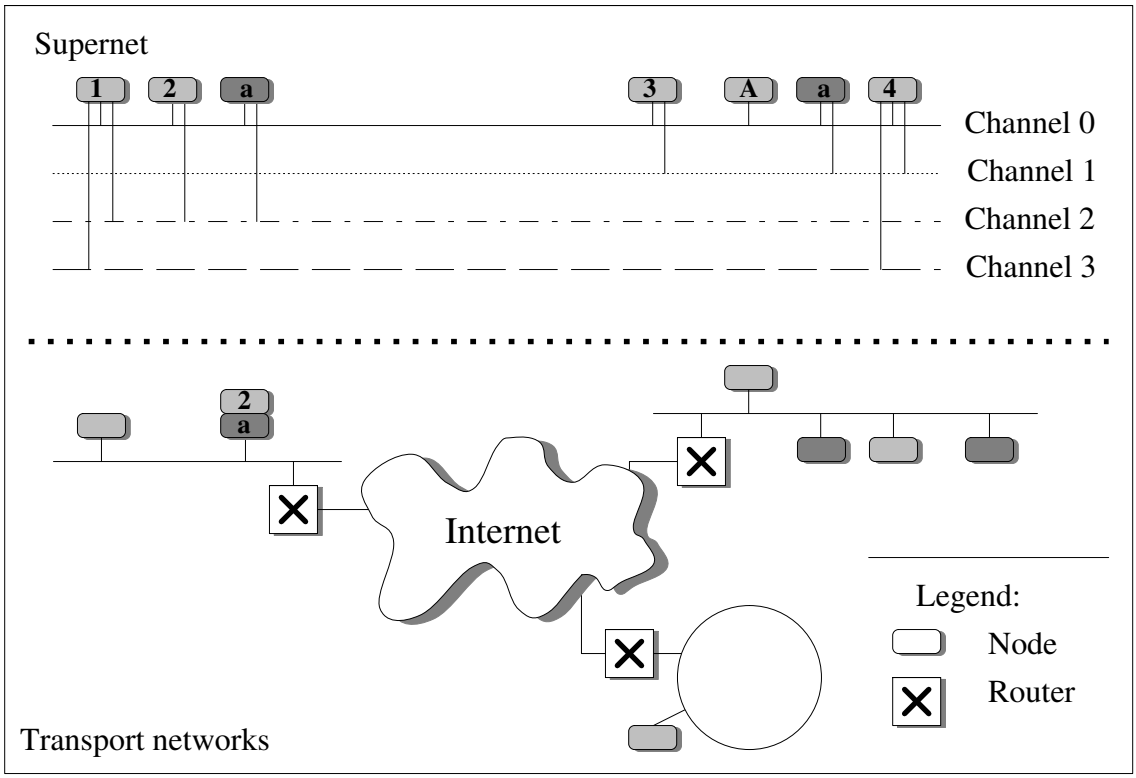

Figure 1: The components of a Supernet.

4 Sample of a Supernet

The sample Supernet illustrated in Figure 1 has four channels:

1. The administrative communication channel Channel 0 (C_0), represented by the line to which all nodes are connected,
2. Channel 1 (C_1), which enables secure communication between [3] and [4],
3. Channel 2 (C_2), which enables secure communication between [1] and [2],
4. Channel 3 (C_3), which enables secure communication between [1] and [4].

Nodes [1] and [4] can communicate with each other on two channels: the administrative channel C_0, and on C_3. Nodes are bound to channels, and it is through this binding that entities are isolated from each other. Nodes [1] and [4] cannot communicate on C_2 because [4] is not a member of channel C_2.

The entity marked [A] in Figure 1 is the abstract entity responsible for enforcing the security policy of the Supernet (admission control, authentication, and authorization), implementing the name service (DNS style), the VARP service, and key management services. Obviously, these services can be implemented in a distributed fashion.

The entities marked [a] in Figure 1 are the channel administrators of each channel. These entities enforce the Supernet policy of their respective channels. For example, deleting a user from a Supernet requires deleting it from all channels of which it is a member.

Applications, high-level services, and users do not need to be aware that they are using this virtual computing environment. This means that to a reasonable extent, the function and behavior of networks, storage, and computing continue to work unchanged in the Supernet environment.

4.1 Joining a Supernet

Joining a Supernet consists of authenticating the joining entity (done by [A] in Figure 1). Once authenticated, the entity is given authorization in the form of capabilities to use the Supernet consistent with its policies. For example, if a node is permitted to access a file service resident on channel C, it is given a capability to talk to the channel administrator for C and to join C. Joining nodes may be directly controlled by a human (such as a login process), or a service that seeks connection in order to make eventual use of the Supernet possible by a human being. These types may require different types of authentication, so the Supernet administrator [A] must permit many different types suitable for each use.

4.2 Leaving a Supernet

A node may leave a Supernet in three fashions: The node may voluntarily part from the Supernet, it may time out, or it may be thrown out by the Supernet administrator [A]. All three cases are handled similarly, it is just a different agent that initiates the *leave* action.

When a leave action takes place, the key manager and VARP server are informed first. The key manager initiates a key change for the remaining group, resulting in the leaving member being unable to understand any traffic in the channels that he is leaving. This operation usually takes less than a second. The VARP servers take the address mapping for this Supernet ID and virtual node address, and delete it from their database. This will cause it to eventually time out from all client caches, and disappear from the Supernet.

Finally, the Supernet Administration Server (SASD) notes the fact that this member is logged out, logs some accounting information and potentially recycles its virtual address.

5 Implementation Issues

The current Supernet implementation is based on a Linux variant, such as Red Hat, or Familiar/Intrinsyc. Some of it exists as user-land processes, and loadable modules, while other parts of Supernets require modification of core kernel data structures, such as the proc structure. Let us explore this in more detail:

5.1 User vs. Kernel space

In general, all administrative functions such as admission control, address assignment and key management have been implemented as user land processes.

Function	Server	Client
Admission Control	sasd	snattach
Key Management	kms	kmc &kmd

Admission control is handled by a client/server application. A user or service joins a Supernet by issuing the snattach command, with the appropriate arguments. The snattach program contacts the *Supernet Admission Server Daemon* (sasd) on the host specified in the command line. At this point, sasd passes the correct operational parameters, including Supernet id, channel id and addressing information to the snattach client. The snattach program sets up the correct environment and exec()s the program, often an xterm, in the secure environment.

Key management is an ongoing function in a Supernet. While the actual encryption and decryption of traffic occurs in the networking portion of the kernel, the key database utilized by the encryption layer is maintained by a set of user land processes. In the Supernet implementation described here, the key manager client (kmc) communicates with the key manager daemon (kmd) and key manager server (kms) as described in section 3.4.

Virtual address resolution (VARP) has been implemented as a kernel loadable module (varpd). When a datagram is to be sent, it traverses down the IP stack. At this point, the datagram contains an address in the virtual address space, which is generally not routable on the physical network. Before constructing the final datagram, the encapsulated datagram is encrypted in the kernel, the virtual address translated through the varp module, and the resulting packet is routed to its physical destination using the same IP stack through which it first traversed. This is necessary in order to be independent of the mechanisms used within the stack for routing table lookups and interface delivery.

The core of the process management and internal packet routing has been accomplished through direct modification of the Linux kernel. Specifically, the TASK_STRUCT has been modified to add a pointer to a Supernet context. This Supernet context contains the Supernet ID, channel set and virtual IP address information for the node.

Maintenance of this data structure also required modification of the FORK() and EXEC() system calls. These changes allow the kernel to maintain proper reference counts to the Supernet context information, as well as to propagate Supernet memberships from parent to child processes.

Finally, modifications have been made to the kernel socket handling code and SKBUFF data structure. Since a node represents an endpoint on a secure channel, a channel-socket association must be created and maintained on each socket creation call. This is handled by maintaining the Supernet id, channel id and virtual address information within the socket structure each time a socket is created in a Supernet context. Storing this tuple in the socket structure and using it for demultiplexing datagrams provides us with a transparent virtualization of the transport layer, from the perspective of applications running within a Supernet context.

6 Experiences with Supernets

The following characteristics have been observed in the initial, small scale deployments of a Supernet. They have been confirmed in our current prototype, involving up to 40 clients interacting with the Sun Microsystems company network.

6.1 Protocol Tunneling

The appearance of a single virtual network on an Internet-wide scale is not a new one. In an effort to stretch out the lifespan of the IPv4 address space, traditional gateway-based VPNs have been implementing this type of functionality for years. Supernets have proven to have a significant advantage over gateway-based VPN solutions. A key consequence of the Supernet abstraction is an independent addressing scheme for entities on the Supernet. This is an additional layer of indirection in naming and routing to real entities on the underlying communications network. This indirection has, among other things, the effect of preserving the end-to-end characteristics that are lost when traversing VPN gateways.

6.2 Multiple Nodes Per Physical Host

Supernets are the basic building block in our vision of out-sourcing. Because out-sourced resources such as computation and storage may be managed independently, the same physical host that provides more than one resource may be assigned to multiple Supernets. In such a case, separating physical addresses from Supernet addresses is very desirable because the same physical host may be assigned different addresses in each such Supernet.

6.3 Simplified Address Management

A key practical problem faced by network administrators today is that of address renumbering and address aggregation resulting from relocation. This usually results from the fact that IP addresses tend to belong to the ISP and not to the corporation. When a company switches ISPs, internal addresses must be renumbered to match their new ISPs address assignments. The high renumbering costs essentially force companies to stay with one ISP.

By separating Supernet addresses from the addresses used by the underlying communication infrastructure (for routing, for example), we can eliminate this problem for corporate and ISP network managers. Renumbering can happen as often as needed at the underlying physical network, without disrupting network services. This is made possible via an automatic address mapping registration in the VARP service with flexible validity expiration periods.

6.4 Addressing Information Remains Private

Layering Supernet addresses over the addresses on the underlying communication infrastructure, we can hide the Supernet addresses from all outsiders. Using a channel-based security architecture, access to services on channels is trivially enforced through key management. Key revocation is enforced through a forced group key change as described in [CWSP98].

6.5 Naming Implications

We have identified two initial areas of exploration with respect to naming within a Supernet. These areas are host names and file systems.

Host naming, commonly handled through DNS, becomes a complicated issue in a Supernet. A process running with a Supernet context is most likely subject to a different naming context than the physical host. As such, name server information must be communicated during the snattach process. Without this communication, the transparency of communicating with other Supernet nodes by host name cannot be achieved.

The second set of naming issues revolves around that of local file system views. The view of a file system from within a Supernet context impacts the set of remote files available in a given process. For example, a machine hosting processes that belong to two different Supernets may have separate sets of NFS file systems mounted. To be secure, each file system is only visible within the Supernet that requested the mount.

Additionally, the view of the local file system impacts the configuration information accessible within a Supernet context. In general, a host only has a single instance of a configuration file, for example, /etc/resolv.conf, /etc/passwd, /etc/printcap or /etc/mail/sendmail.cf. Unfortunately, this complicates life in a Supernet context. Since nodes within a Supernet will have different ideas of what user, printer, mail and other configurations will look like, provisions must be made for providing Supernet specific views.

7 snHubs: A Supernet Hardware Implementation

For the latest prototype in the realm of Supernets, we have chosen to provide a dedicated hardware platform, which can be inserted between the client and the public Internet. This device, which we call a *snHub*, provides all necessary functionality to make the client machine part of one Supernet.

The main benefits of this form of implementation are client platform independence, and a well-known and controlled configuration. The snHub acts as a very small ISP to the client, either providing it with a view into a Supernet, or with a view to the public Internet.

The snHub features an LCD display, a mouse wheel, and a smartcard reader/writer. In terms of interfaces, only two 10/100 Ethernet sockets are of importance. Internally, it runs a Linux kernel on a StrongARM PXA250 processor. Its throughput of protected traffic appears to be adequate for the use in leased-line or cable modem environments.

8 Looking Ahead: Future PUC Components

Supernets, as described here, are just the first component in building an environment for Public Utility Computing. For this model to become pervasive, added functionality must be implemented in Secure Storage and Secure Computing, as shortly outlined in the two following subsections.

8.1 Secure Storage

In a world in which out-sourced data will be stored on untrusted servers, there need to be mechanisms that provide an assurance of the integrity and privacy of the stored data. This should be possible with minimal requirements for trusting the provider. This means that out-sourced data may need to be:

- encrypted, so the provider does not have to be trusted with it,
- *block-headered*, so that modification (including deletion) and non-repudiation of the data by the provider is at least detected, and
- rapidly re-encryptable when the organization's membership changes. When a member of an organization departs, he should no longer be able to view the organization's data even if he has access to the raw data through other paths within the provider's infrastructure.

8.2 Secure Computing

Finally, the ability to securely purchase CPU cycles is of interest to many organizations. While the days of batch processing have seemed to slip into history, many situations arise where the ability to farm out distributed processing tasks is highly desirable. The SETI Project is a good example of these distributed processing tasks. In this model, any corporation could contract out spare computing resources, thus generating revenue by selling under-utilized resources.

While security may not be of concern to a public project such as SETI, companies such as Pixar, which use large computing farms to generate entire animated movies, could greatly expand their processing capacity without the need for massive capital outlays. To make this component a reality, a secure memory model is critical. Private companies must be assured that confidential information is kept secret throughout all phases of processing: from design server to compute farm to long-term storage.

9 Bibliography

References

[CKSS00] Germano Caronni, Sandeep Kumar, Christoph Schuba, and Glenn Scott. Supernetworking: The next generation of secure enterprise networking. In *Annual Computer Security Applications Conference*. Applied Computer Security Associates in cooperation with ACM, December 2000.

[CWSP98] Germano Caronni, Marcel Waldvogel, Dan Sun, and Bernhard Plattner. Efficient security for large and dynamic multicast groups. In *Proceedings of the IEEE 7th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)*, June 1998.

[SL99] Christoph Schuba and Yuefeng Liu. Virtual Address Resolution in Supernets: VARP, October 1999.

About the Authors

Germano Caronni (IEEE/ACM/ISOC) received his M.Sc. in Computer Science in 1993, and a Ph.D on QoS-based Dynamic Security in 1999, both from ETH Zürich. He was one of the first to invent a process to watermark images, participated in the IETF (IPSEC), led the independent implementation effort for SKIP (secure TCP/ IP), and its integration into an adaptive firewall. In 1997, he won the RC5/48 challenge of RSA DSI. Since late 1997, Mr. Caronni has been with Sun Microsystems, where he has introduced a novel solution to secure multicasting, worked on authentication frameworks, participated in the design of an overall security architecture for Sun's products, and co-invented the concept of Public Utility Computing. He is currently a member of the Security Research Group in Sun Microsystems Laboratories, and principal investigator on secure storage solutions.

Pete St. Pierre is a Staff Engineer in the CTO/Network Technology Office. He has worked on a variety of projects at Sun Microsystems dating back to 1990 and the SunNet Manager and Cooperative Consoles products. After a brief consulting engagement, he returned to Sun in 1996 and has worked on research and advanced development projects including distributed security policy management, network service location (SLP) and platforms for mobile computing. Mr. St. Pierre is currently working on network architectures for the DARPA High Productivity Computing System (HPCS) project. He is a graduate of the University of Massachusetts and holds a B.S. in Computer Science with a background in Economics.

Glenn Scott is Senior Engineering Manager and Principle Investigator of the Security Research Group, which he created at Sun Microsystem Laboratories. Currently, He is focused on secure virtual enterprise networks and computing in totally public environments. Peripheral research is in hardware and software devices providing "portable" security for small devices and hardware interfaces. Prior to Sun Microsystems Laboratories, Mr. Scott was the Director of Engineering and the Chief Technologist for Internet Commerce and Security business unit of SunSoft, which defined Security and Electronic Commerce product and technologies for Sun Microsystems, Inc. Previously, he was a Senior Staff Engineer and charter member of the Internet Commerce Group, a strategic business unit formed within Sun Microsystems Laboratories focusing on technology and product development for doing business over the Internet. Before joining Sun in 1990, Mr. Scott was with System Development Corporation's Santa Monica Research Center as a researcher in formal methods of software development, automatic theorem provers, secure operating systems, and trusted systems. Mr. Scott is a graduate of Chapman University, with an academic background in Computer Science, Electrical Engineering, and Applied Linguistics.