Secure Quality of Service Handling: SQoSH

D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, Steve Muir, and Jonathan M. Smith, University of Pennsylvania

This work was supported by DARPA under Contract # N66001-96-C-852, with additional support from the Intel Corporation.

ABSTRACT

Proposals for programmable network infrastructures, such as active networks and open signaling, provide programmers with access to network resources and data structures. The motivation for providing these interfaces is accelerated introduction of new services, but exposure of the interfaces introduces many new security risks. The risks can be reduced or eliminated via appropriate restrictions on the exported interfaces. In this article we describe some of the security issues raised by active networks. We then describe our secure active network environment architecture. SANE was designed as a security infrastructure for active networks, and was implemented in the SwitchWare architecture. SANE restricts the actions loaded modules (including "capsules") can perform by restricting the resources that can be named; this is further extended to remote invocation by means of cryptographic credentials. SANE can be extended to support restricted control of quality of service in a programmable network element. The Piglet lightweight device kernel provides a "Virtual Clock" type of scheduling discipline for network traffic, and exports several tuning knobs with which the clock can be adjusted. The ALIEN active loader provides safe access to these knobs to modules that operate on the network element. Thus, the proposed SQoSH architecture is able to provide safe, secure access to network resources, while allowing these resources to be managed by end users needing customized networking services. A desirable consequence of SQoSH's integration of access control and resource control is that a large class of denial-of-service attacks, unaddressed solely with access control and cryptographic protocols, can now be prevented.

INTRODUCTION

This article describes one approach to providing quality of service (QoS) guarantees in a network using a secure active network infrastructure (Secure Active Network Environment, SANE) and a novel low-overhead multiprocessor operating system (Piglet). We begin by discussing the notion of security, provide a constructive definition useful for asserting general security properties, and discuss the mapping of this definition onto computer networks, and outline the challenges of securing programmable network infrastructures, which the remainder of the article addresses.

SECURITY

It is attractive to think of security as a clearcut property of a system, as in the definition of odd and even numbers. Unfortunately, security is not as easily abstracted, since it is fundamentally a context-dependent property of an engineered system. For example, some applications can tolerate near-infinite delays, while others have strict real-time requirements. Thus, a system design which guarantees reliable and eventual action may be considered secure in the first case but inadequate (e.g., against "denial-of-service" or DoS attacks) in the latter (e.g., a military command and control system susceptible to DoS attacks).

In general, a secure system is one that meets or exceeds an application-specified set of security policy *requirements*. So, for example, in message delivery, the high-level requirements may be that the correct information gets to the right person, in the right place, at the right time. The details of "right" are determined by the application's needs; for example, timely message delivery is crucial for battlefield or stock-trading tasks.

SECURITY FOR ACTIVE NETWORKS

Active networks are proposed for packet-switched networks which are programmable, perhaps on a per-user or even per-packet basis. The more aggressive proposals share the property that "programs" are loaded into network elements on the fly, providing rapid dynamic reconfiguration of the network infrastructure.

Security for active networking is a major challenge, as well as a widespread and legitimate cause for concern. One view of information security can be characterized as getting the right information to the right person at the right place and time. This is the positive statement of a security policy; other security policies might assert what *cannot* occur. The flexibility of an active networking infrastructure has the potential effect of hugely expanding the threat model for attacks on the network infrastructure. For example, DoS attacks can now be made against a variety of resources, such as CPU cycles, output link bandwidth, and storage, since these are exposed either wholly or in part to loaded programs.

Typical reasons to defer consideration of security, aside from simple difficulty, are the negative consequences of making a system more secure on flexibility, usability, and performance. Since programming-language-based approaches to active networking offer advantages in terms of flexibility and usability, and performance optimizations for these environments are ongoing, providing security to such an environment would offer an attractive design point among the various trade-offs.

THE THREAT MODEL FOR QOS PROVISION IN AN ACTIVE NETWORK

An active network infrastructure is very different from the current Internet. In the latter, the only resource consumed by a packet at a router is the memory needed to temporarily store it and the CPU cycles necessary to find the correct route. This overhead is generally quite small compared to the cost of executing an active packet, and thus strict resource control in the routers was considered noncritical. While this approach has worked well, it is fairly easy to mount DoS attacks due to the simple resource model. Attacks to the infrastructure itself are possible, and result in major network connectivity loss. Finally, it is very hard to provide enforceable QoS guarantees.

In the context of QoS, a secure system is one that is secure against two types of threats, which we denote *admission* failures and *policing* failures.

An admission violation is one where an unauthorized reallocation of bandwidth/resources occurs. For example, a Resource Reservation Protocol (RSVP)-capable router might be asked to reallocate bandwidth away from one flow to one that has not paid for/has no right to the bandwidth. The policing mechanism is working correctly; it is applying a QoS specification admitted by the network infrastructure. Unfortunately, the specification was unauthorized. This is as much a threat in RSVP or other resource reservation systems as it is in an active network; the greater concern in the active network is a direct consequence of the more complex resource model. Thus, an active infrastructure must be able to correctly identify and authorize users (or, more generally, principals in the system). The SANE architecture provides such services [1].

A policing violation is one where the specified QoS is correct, but the system fails to deliver what is requested. For example, a network element incorporating a computer might be subject to DoS attacks based on "receive livelock," or may experience "QoS crosstalk." A second example is aggressive use of bandwidth on a shared output port, which denies bandwidth to a process with QoS requirements. This is a threat to basic IP routers with FIFO queuing disciplines. Piglet [2], briefly described later, provides the necessary mechanism to enforce QoS policing.

SQOSH APPLICATIONS

The proposed Secure QoS Handling (SQoSH) architecture provides a powerful new tool for managing resources in a network. It controls access to managed resources, and integrates this control with the resource management mechanisms provided by the Piglet operating system. While the SANE/Piglet combination represents the first instance of the SQoSH architecture, we believe that compelling applications will motivate deployment of SQoSH and SQoSH-inspired architectures. Examples include:

- Economic algorithms for robust adaptive control: Active QoS is well adapted to this environment for two reasons. First, it is critical that resources sold match those delivered if the marketplace is to work. Second, the recovery strategies for flows that are outbid in an auction may be quite complex (e.g., aggregating several flows, each of which delivers a portion of the request, searching for a different route, delaying until the resources required become available at the desired prices, or combinations of different strategies). We expect that capturing these complex decisions can be done most easily by active packets in a programmable infrastructure.
- Military applications, where hierarchical command responsibility maps to multiple classes of service and security: The SQoSH architecture ensures that control requests are authenticated, autonomous network elements bootstrap into a secure state, and (once admission requests are validated) service will be delivered. For example, a command channel of 2 percent of bandwidth could be preserved at all times. For commercial applications this might be considered wasteful, while military uses might dictate provision of such a no-delay override facility.

The proposed SQoSH architecture provides a powerful new tool for managing resources in a network. It controls access to managed resources, and integrates this control with the resource management mechanisms provided by the Piglet operating system.

THE SQOSH ARCHITECTURE

The goal of SQoSH is to protect against two types of threats to QoS provision, admission and policing. Balancing performance, flexibility, and security considerations suggests that we make common operations (e.g., those used to classify packets) cheap, and make less common operations more expensive if this contributes to reducing the cost of common operations. An example of this approach is to provide heavyweight authentication mechanisms at the level of aggregates of packets such as channels or flows so that these checks need not be done on individual packets.

This suggests an architecture where authentication and other resource management decisions are "front-loaded" to reduce the cost of subsequent decisions. We view this scheme as one where expensive static checks are traded for cheaper dynamic checks. Thus, the SQoSH architecture echoes similar design decisions made in restricting programmability to the control plane and similar, although not equivalent, decisions made in the overall *SwitchWare* architecture.

This division of functions into admission/authentication and policing/provision is the approach we have chosen for SQoSH. Figure 1 illustrates the SQoSH architecture at a high level. The SANE system is the only means of access to resource management interfaces provided by Piglet. Heavyweight cryptographic operations required for granting access to Piglet resources are performed by SANE as a front-

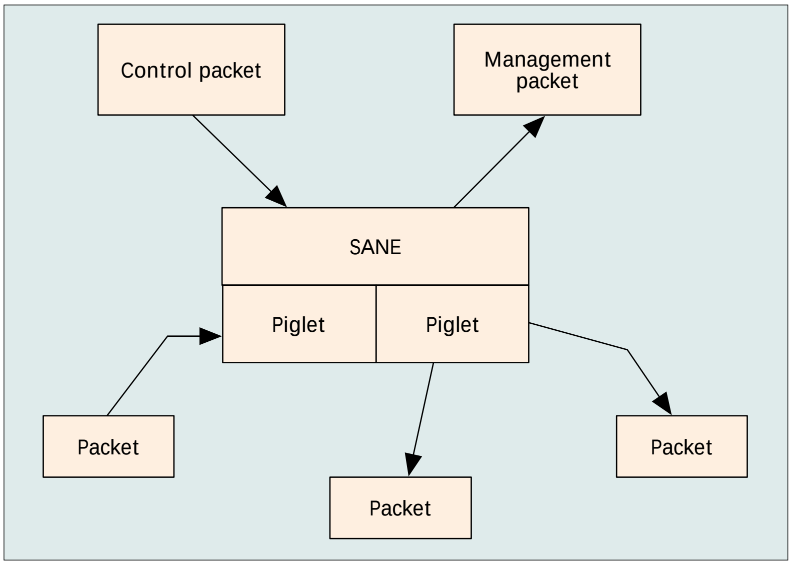


Figure 1. *The SQoSH architecture.*

end. Piglet is thus assured that any resource requests have been authenticated, and thus needs to focus on whether the resources can be allocated to the validated request. Packets destined for SANE are demultiplexed by Piglet, which provides basic packet delivery operations for SANE. The active environment provided by ALIEN allows for considerable flexibility in arbitrating and managing the protected resources.

Since Piglet is intended as an asymmetric multiprocessor operating system, multiple instances of Piglet can manage multiple network line cards. A full-scale SQoSH system would consist of a multiprocessor with a Piglet instance on each device-managing processor (a resizable subset of all the processors in the system). In this way, all device I/O can be managed by Piglet. Since a typical model for scheduling resources is activity triggered by a device interrupt, Piglet can manage interrupts, buffering, status polling, and so on, and protect the host operating system from device-initiated actions.

Security and Safety in SwitchWare

While the SQoSH architecture is portable across many active networking environments, our experimental prototype is constructed in the context of the SwitchWare architecture. SwitchWare is based on the approach of using restricted semantics to contain the behavior of potentially mischievous programs. This has the benefit that enforcing restrictions can be performed once at compile or link time, resulting in lower cost than an OS approach such as memory protection, which requires repeated checks at runtime. These semantic restrictions depend on the integrity of other system components such as the operating system and shared libraries. The semantic restrictions are enforced with a strongly typed language which supports garbage collection and module thinning.

The SwitchWare project is a joint effort of the University of Pennsylvania and Bellcore (now Telcordia). The overall project goal is to accelerate network evolution by turning store-and-forward networks into store-*COMPUTE*-and-forward networks, an approach we originated in the Protocol Boosters [3]. The goal is to build a network infrastructure which balances flexibility, usability, security, and performance. The current infrastructure provides a model where modules can be loaded into the node on the fly by the ALIEN active loader [4].

The basis of the ALIEN approach is restricting a general mode of computation. Caml provides the general model, and ALIEN provides the restrictions. The loader is sufficiently powerful that the extreme case of "capsules" can be supported by treating each packet as a module, although a more typical use of the facility is to add a service used by streams of inactive packets.

More precisely, Caml provides a model of computation equivalent to that of a Turing machine. By itself, this computation model is secure since it involves no shared resources. In practice, since we are running on a real machine, we have DoS attacks that arise because our CPU and memory resources are finite. Additionally, the actual Caml environment also includes a runtime system that, among other features, provides access to operating system primitives, which, in turn, provide access to shared resources. Furthermore, under this runtime, memory is a shared resource. The role of ALIEN is to control the access to these shared resources and thereby ensure that a loaded program (called a *switchlet*) does not exceed its resource limits (ALIEN is not responsible for determining those limits).

The Secure Active Network Environment

An illustration of SANE in the context of the overall SwitchWare network element architecture is shown in Fig. 2.

SANE builds on ALIEN in order to provide security services for an active network. These services include:

- Secure bootstrapping and component recovery using the AEGIS architecture [5].
- Cryptographic primitives provided as ALIEN libraries.
- Packet encryption and authentication, for both data packets and active capsules.
- A key establishment protocol (KEP), which allows nodes or users to establish secret keys and exchange certificates. This protocol is used for:
  –Secure bootstrap component recovery in AEGIS [6]
  –Session-key establishment and principal authentication and authorization. The principals can authenticate each other and exchange authorization credentials. We make use of KeyNote [7] credentials to specify the resource usage and access control policies ALIEN enforces.
  –Secure neighbor discovery once the node boots. The node may also establish trust relationships with its neighbors; these may then be used by mobile-agent types of applications, or to secure other critical infra-

structure information (e.g., routing updates).
- Administrative domains allow a set of network elements under the same administrative control to restrict security requirements when communicating with each other. Border elements act as present-day firewalls and may require extensive authentication before allowing access to the internal network. Furthermore, they can impose restrictions on active packets by encapsulating them in active "wrappers."
- Naming services allow for secure module identification.

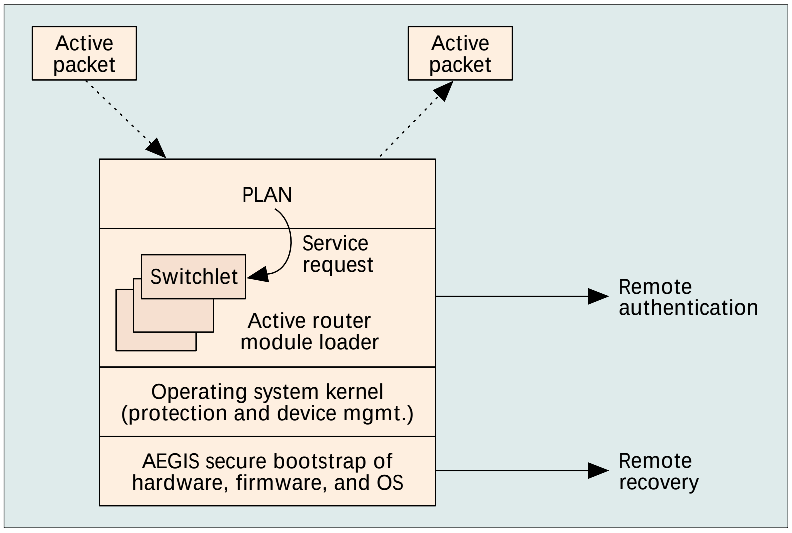
SANE SUMMARY

The SANE elements are combined into a system using the following design principles:
- Dynamic checks, performed while the active node is operating, should be as fast as possible, since they are done many times.
- Static checks, performed before the active node enters the operating state, can be expensive, since they are done infrequently (typically once).
- System performance can be improved by trade-offs that decrease the cost of the dynamic checks at the expense of more costly static checks, or, ideally, by using static checks to eliminate the need for any dynamic checking at all, analogous to "once at compile time" versus "many at runtime."

RESOURCE CONTROL

Resource control on the active switch is imposed by the runtime system, as specified by the credentials exchanged during key establishment. Piglet controls resource usage based on the credentials acquired by SANE. The protected resources include access to standard and loaded modules, CPU cycles, memory allocated, number of packets, latency and bandwidth requirements, and so on. For the Piglet experiments we report later that we have limited resource management to network bandwidth. There is a great deal of further research necessary to determine which are the right resources to manage and how to resolve conflicting resource requests.

Since a tenet of our approach is controlled module loading, SANE must manage loading modules in a secure fashion if it is to be useful in an active network. It must control which modules are loaded, and by whom. SANE associates cryptographic credentials with modules. SANE can either require a certificate for loading a particular module, or allow universal loading of the module. Examples where such universal loading may be useful include low-cost operations like `ping`, as well as the security operations used for bootstrapping the security relationship with remote switches. There are two classes of certificates that can be presented by a user packet requesting access to a resource via a module. An *administrative* certificate allows loading of any or all modules into the system; it is intended for management and emergencies as might arise, and can be thought of as analogous to a "master key" granted by the switch administrator. More commonly, certificates are used to permit loading of selected modules. For system resources (e.g., memory or bandwidth), the certificates can also specify the allowed usage patterns (e.g., "no more than 4 Mb/s"). As a result, this scheme allows fine-grained control of switch resources.


Active packet
Active packet
PLAN
Service request
Switchlet
Active router module loader
Remote authentication
Operating system kernel (protection and device mgmt.)
AEGIS secure bootstrap of hardware, firmware, and OS
Remote recovery
■ **Figure 2.** *SANE's relation to SwitchWare.*

PIGLET

While the SANE architecture presents a secure resource management interface to switchlets, the underlying system must be capable of providing the appropriate capabilities to implement that interface. In the SQoSH architecture those capabilities are provided by Piglet [8], a multiprocessor operating system designed to provide applications with low overhead direct access to network resources.

PIGLET: STRUCTURE AND ARCHITECTURE

The Piglet architecture is based around the concept of an *active kernel* — one or more system CPUs are dedicated to continually running the *lightweight device kernel* (LDK). The LDK consists of only those parts of the operating system that directly interact with physical devices; like other vertically structured operating systems, it implements only the minimal set of functions necessary to support direct user-level access to those devices, namely protection, translation, and multiplexing.

This has two primary benefits: first, since it runs continuously the LDK communicates with devices by polling, rather than using interrupts, thus eliminating the possibility of interrupt-based livelock, potentially reducing kernel response time and greatly simplifying the kernel implementation; second, applications invoke kernel services using shared memory communication, a much lower-overhead mechanism than the conventional system-call trap. We believe that in systems where I/O makes up a large fraction of the workload, such as an active network element, these two benefits combined more than compensate for the cost of dedicating one processor to running the LDK.

Reducing System-Call Overhead — To evaluate how successfully Piglet manages to reduce the invocation overhead for system services, we

```
frameset_t *
piglet_create_frameset (int tx_bufs,
                        int rx_bufs,
                        unsigned options);

int
piglet_set_filter (frameset_t *flow,
                   filter_t *filter);

int
piglet_set_vclock (frameset_t *flow,
                   unsigned period,
                   unsigned limit);
```

Figure 3. *Flow management functions in Piglet.*

	Modal RTT (µs)	
Payload (bytes)	**Linux**	**Piglet**
64	199	161
256	253	215
1024	468	413

Table 1. *Modal ping round-trip times.*

performed a simple experiment. The standard `ping` program was modified to use the Piglet shared-memory interface to send *echo* packets. We then measured the round-trip time for various sizes of packet, and compared these to the times measured for the unmodified program in the same test environment. Since the only difference between the two tests was the mechanism used to send packets, any differences in round-trip time can be attributed to the overhead of those mechanisms.

Table 1 shows the modal round-trip time measured for various sizes of packets; a more detailed histogram and analysis of these results is presented in [8].

These figures show that the difference in round-trip time, or equivalently the difference in the overhead of sending a packet, varies from 38 µs to 55 µs, there being some dependence on payload size due to Piglet performing less data copying than Linux. We believe that such a reduction, due to the Piglet architecture, could be significant in a system such as a SQoSH node, where a large proportion of the workload is in I/O operations.

Resource Management in Piglet

Piglet's network resource management functions are based on an abstract data structure known as a *frameset*. A frameset consists of independent transmit and receive queues into which *frames*, each corresponding to a network packet, can be placed. Services can be associated with framesets in order to manipulate and process frames — since services form an integral part of the Piglet OS, they have full access to the host system and can thus perform almost any conceivable function.

For the purposes of network element applications each frameset is logically associated with a network *flow*. The exact details of a flow specification are bound only by the packet-filtering mechanism employed by Piglet to classify received network packets into flows (and hence demultiplex to the appropriate frameset). A flow can be defined as broadly as "all packets received on this network interface" or as specifically as "all packets sent by host 158.130.6.140 to TCP port 5005 on host 158.130.4.4."

Figure 3 shows prototypes for the principal functions used to manage framesets. `piglet_create_frameset` is used by an application to create a frameset, specifying the transmit and receive buffer sizes and an option field indicating, among other things, which services should be used to process this frameset. Once the frameset has been created, `piglet_set_filter` is called to associate a flow specification with the frameset.

An example of a service provided by Piglet is the Virtual Clock algorithm, a mechanism for scheduling a network link across multiple flows and controlling the rate at which each flow sends data. Virtual Clock is parameterized by the clock period and maximum amount of data to be sent in that period; tuning these two parameters allows an application to specify not only its bandwidth requirement but also the degree of burstiness of its traffic. The function `piglet_set_vclock` is used to convey these parameters to Piglet.

SQoSH Resource Multiplexing Using Piglet

The specific problem we address is that of multiplexing a single network interface between a number of uncooperative applications, each of which may have specific requirements. This is exactly the challenge faced in secure multiplexing of resources once a policy has been validated. In this example, the resource we wish to divide among these applications is network bandwidth.

The Experimental Description

The experimental setup for this test consists of two PCs connected to an AsanteFast 100 Mb/s Ethernet hub by 3Com 3c905 network interface cards. The sender is a 200 MHz dual-processor Pentium Pro PC running RedHat Linux 5.0 with the Piglet kernel replacing the standard Linux kernel. The receiver is a 200 MHz uniprocessor Pentium Pro PC running RedHat Linux 4.2 with Linux kernel 2.0.31. Both machines are idle apart from the test applications, and the test network has no other traffic.

The experiment consists of three applications all trying to send a large amount of data from the sender to the receiver. The results are plotted in Fig. 4. Each application has different bandwidth requirements:

- **A** — an unconstrained sender that uses as much bandwidth as is available. This can be viewed as the source of a DoS attack in the context of SQoSH
- **B** — a sender constrained to run at 40 Mb/s[1]
- **C** — a sender constrained to run at 10 Mb/s

The application used to send the data is the standard *ttcp* augmented with an option to set the Virtual Clock parameters (period and time).

[1] *Here and in our experimental results we use the convention that 1 Mb/s = (10^6) b/s.*

These parameters are passed to the Linux TCP/IP stack by `setsockopt()` system calls, where they are then passed to Piglet to create application-specific framesets with those parameters. This is the only modification made to the Linux networking code.

Applications **A**, **B**, and **C** each start and stop sending their data at different times, and the per-application bandwidth is measured every second and plotted in Fig. 2 as the three heavy lines. The three thinner lines show reference bandwidth measurements for each sender with no competing applications over a 30 s period.

- After 1 s, **B** starts sending at 40 Mb/s.[2]
- After 5 s, **A** starts sending as fast as possible. Piglet's guarantee of 40 Mb/s to **B** limits **A** to approximately 40 Mb/s also.
- After 9 s, **C** starts sending at 10 Mb/s, causing **A**'s bandwidth to decrease by approximately 10 Mb/s.
- After ≈15 s, **B** stops sending; **A**'s bandwidth thus increases to approximately 10 Mb/s below the absolute limit.
- After ≈20 s, **A** stops sending.
- After ≈29 s, **C** stops sending.

Experimental Results

We see from Fig. 2 that Piglet's queue scheduling mechanism provides controlled multiplexing of the shared network resource, despite the fact that the applications are not cooperating to share the resource; neither they nor the host O.S. (Linux) are aware of the constraints imposed on them.

The applications which have specific bandwidth requirements receive exactly that amount of bandwidth, even when an application with no specified constraint is competing for the same resource. This ability to add resource management to a standard host operating system is one of the key strengths of Piglet, and enables its easy integration into SQoSH.

Related Work

QoS Provision and Management

QoS provision and management has a wide-ranging literature. A lot of the early work was stimulated by the promise of asynchronous transfer mode (ATM) networks. The demand for these services was stimulated by multimedia traffic. The relevant promise was the control of multiplexing behavior in both endpoints and network elements, with the idea that ATM hardware-supported queuing disciplines, such as Fair Queuing or its many variants, could be used to allocate bandwidth resources, and for the most part provide delay bounds. While such hardware support remains attractive, the signaling software has proved sufficiently unwieldy that the potential for managed bandwidth remains largely unrealized.

The attraction of integrated services did serve, however, to revitalize and stimulate research into integrated services in the IP Internet community. This research program resulted in the RSVP proposal for signaling resource reservations to network elements by endpoints.

Neither ATM signaling protocols nor RSVP provide the integrated admission control and policing of SQoSH. It is presumed that administrative entities are trusted in either system, while policing is delegated; to hardware in the ATM setting, and to some lower layer through the Internet subnet-specific layer in the RSVP case. Some extensions for securing signaling are discussed by Schuba [9]. An additional limitation of these systems is that their policing is limited to bandwidth management.

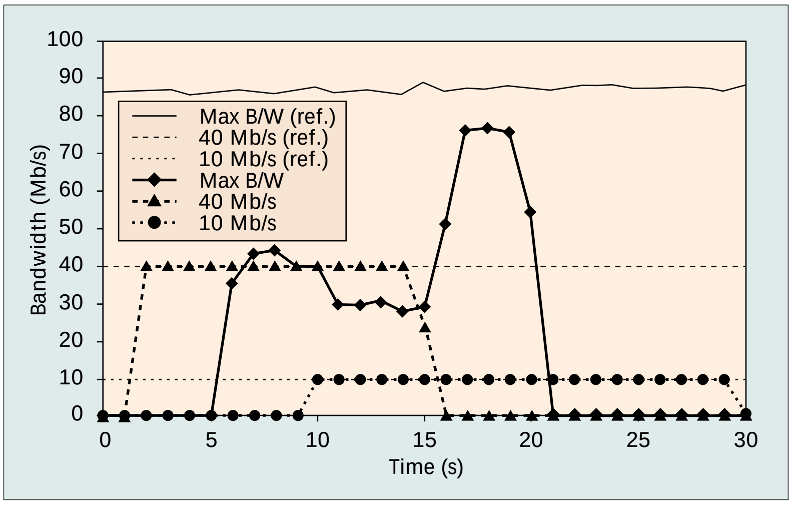


Figure 4. *Per-channel bandwidth as a function of time.*

Secure Resource Control in Active and Programmable Networks

SANE has no direct analogs in ongoing work on active networks [10]. While ANTS uses MD5 hashes ("fingerprints") to name on-demand loaded modules, the hashes provide unique names rather than security. The ANTS execution environment depends on the Java programming language for protection, a dependency shared by many active network prototypes. Unfortunately, as Wallach *et al.* [11] note, Java's security is suspect. The remote authentication and namespace security aspects of SANE address issues ignored in these systems, and could be applied even in cases where Java is used (e.g., to provide integrity checking of the JVM or layers beneath it, as well as on-demand loaded modules).

PLAN [12] is a part of the SwitchWare project at the University of Pennsylvania. The PLAN project is investigating the trade-offs brought about by using a different language for active packets than is used for active extensions. PLAN is designed so that pure PLAN programs will not be able to violate the security policy. Because this limits the operations that can be performed, PLAN programs can call *services* which can be either active extensions or facilities built into the system, perhaps requiring authentication and authorization before allowing access to the resources they protect.

The Safetynet Project [13] at the University of Sussex has also designed a new language for active networking. They have explicitly enumerated what they feel are the important requirements for an active networking language and then set about designing a language to meet those requirements. In particular, they differ from PLAN in that they hope to use the type system to allow safe accu-

[2] ttcp actually tries to send as fast as possible, but Piglet constrains packet transmission to 40 Mb/s.

The SQoSH architecture provides controlled access to allocations of system resources in an Active Network element. It is more generally applicable to any resource allocation or policing scheme where remote allocation and deallocation of resources is required.

mulation of state. They appear to be trying to avoid having any service layer at all.

An architecture which extended a protection model from the local domain to a distributed environment was provided by Sansom *et al.* [14], where protection was enforced locally with memory-protection enforced capabilities. The capabilities were extended to remote nodes via cryptographic means. SANE provides more general mechanisms and could thus be specialized to such an application (moving memory-protected objects about the network) but more importantly guarantees local integrity before extending itself into the network.

Security of active networks is a broad evolving area. We suggest Moore [15] as a source of additional information in this area.

SQoSH and Other Environments

The Cambridge University Nemesis operating system has considerable potential for supporting SQoSH functionality since its single-layer multiplexing model can readily be adapted to the SQoSH policing requirements. The RCANE system [16] follows the ALIEN architecture but utilizes the capabilities of Nemesis to provide resource management and control in a manner equivalent to Piglet's role in the SQoSH architecture.

Another system with great potential is the Arizona Scout/Escort operating system with its support for end-to-end resource allocations, called *paths*. Paths, in spirit, are the right idea for end-to-end allocation in an active network. The security infrastructure is not as complete as SANE, with its secure initialization, public-key infrastructure, ALIEN active loader, and remote module authorization certificates. We believe that, like Nemesis, Scout could readily be adapted to support SQoSH.

Conclusions and Future Work

The SQoSH architecture provides controlled access to allocations of system resources in an active network element. It is more generally applicable to any resource allocation or policing scheme where remote allocation and deallocation of resources are required. We believe that this new architecture is well suited to providing secured resource allocation in an integrated services internetwork.

An example of the general architecture can be constructed using SANE as a protective mechanism for resource allocations available from the Piglet operating system. We have demonstrated Piglet partitioning bandwidth using a Virtual-Clock-like queue scheduling discipline. The measurements showed the effectiveness of Piglet on this task.

Since SANE operations are in the SQoSH "control plane," they are performed infrequently relative to the policing functions; thus, the cost of its cryptographic mechanisms has a minor effect on overall performance. An additional benefit of SANE's use of a public-key infrastructure is the presence of this infrastructure for preserving privacy and integrity of media streams if required.

We believe that SQoSH represents a practical advance in automating and securing the administration of remote network elements of any type. We present the threat model SQoSH addresses, and show that these attacks can be thwarted. Typical architectures and implementations provide no protection against such attacks, except perhaps via inflexibility. In any environment where resources and resource allocations have value, SQoSH ensures that the resources are allocated as intended.

References

[1] D. S. Alexander *et al.*, "A Secure Active Network Environment Architecture: Realization in SwitchWare," *IEEE Network*, Special Issue on Active and Programmable Networks, vol. 12, no. 3, 1998, pp. 37–45.
[2] S. J. Muir and J. M. Smith, "Functional Divisions in the Piglet Multiprocessor Operating System," *SIGOPS Euro. Wksp.*, Sept. 1998.
[3] D. C. Feldmeier *et al.*, "Protocol Boosters," *IEEE JSAC*, Special Issue on Protocol Architectures for the 21st Century, Apr. 1998, pp. 437–43.
[4] D. S. Alexander, "ALIEN: A Generalized Computing Model of Active Networks," Ph.D. thesis, Univ. of PA, Sept. 1998.
[5] W. A. Arbaugh, D. J. Farber, and J. M. Smith, "A Secure and Reliable Bootstrap Architecture," *Proc. 1997 IEEE Symp. Security and Privacy*, May 1997, pp. 65–71.
[6] W. A. Arbaugh *et al.*, "Automated Recovery in a Secure Bootstrap Process," *Proc. Network and Dist. Sys. Security Symp.*, Internet Society, Mar. 1998, pp. 155–67.
[7] M. Blaze *et al.*, "The Keynote Trust Management System Version 2," Internet RFC 2704, Sept. 1999.
[8] S. J. Muir and J. M. Smith, "Piglet: A Low-Intrusion Vertical Operating System," Tech. rep. MS-CIS-00-04, Univ. of PA, Jan. 2000.
[9] C. Schuba *et al.*, Analysis of a Denial of Service Attack on tcp," *IEEE Sec. and Privacy Conf.*, May 1997, pp. 208–23.
[10] D. L. Tennenhouse *et al.*, "A Survey of Active Network Research," *IEEE Commun. Mag.*, Jan. 1997, pp. 80–86.
[11] D. S. Wallach *et al.*, "Flexible Security Architecture for Java," *Proc. 16th ACM Symp. Op. Sys. Principles*, Oct. 1997.
[12] M. Hicks *et al.*, "PLAN: A Programming Language for Active Networks," Tech. rep. MS-CIS-98-25, Dept. of Comp. and Info. Sc., Univ. of PA, Feb. 1998.
[13] I. Wakeman *et al.*, "Designing a Programming Language for Active Networks," submitted to Hipparch Special Issue of *Network and ISDN Sys.*, June 1998; http://www.cogs.susx.ac.uk/projects/safetyne/papers/isdn.ps.gz
[14] R. D. Sansom, D. R Julin, and R. F. Rashid, "Extending a Capability Based System into a Network Environment," *Proc. 1986 ACM SIGCOMM*, Aug. 1986.
[15] J. Moore, "Mobile Code Security Techniques," Tech. rep. MS-CIS-98-28, Univ. of PA, May 1998.
[16] P. Menage, "RCANE: A resource Controlled Framework for Active Network Services," *1st Int'l. Working Conf. Active Networks*, *Lecture Notes in Comp. Sci.*, no. 1653, Springer-Verlag, July 1999, pp. 25–36.

Biographies

D. Scott Alexander is a member of technical staff at Lucent, Bell Labs. His work is in the area of active networking. He earned his Ph.D. and M.SE. in computer science from the University of Pennsylvania. He earned a B.A. in computer science from Rice University.

William A. Arbaugh received a Ph.D. from the University of Pennsylvania in 1999. He currently consults for business and government on a wide range of security issues. His research interests include communications, and embedded and operating system security. He has served as a senior computer scientist with the Research Group of the U.S. Department of Defense, and as a senior software engineer and tactical communications platoon leader with the U.S. Army. He earned an M.S. in computer science from Columbia University and a B.S. from the United States Military Academy.

Angelos D. Keromytis [M] (angelos@dsl.cis.upenn.edu) is a Ph.D. candidate at the University of Pennsylvania. His research interests include communications and operating system security, firewalls, and cryptographic protocols design and analysis. He earned an M.S. in computer science from the University of Pennsylvania, and a B.S. from the University of Crete, Greece. He is a member of ACM, USENIX, and IACR.

STEVE MUIR is a Ph.D. candidate at the University of Pennsylvania. His research interests include operating systems, networking, and dynamically reconfigurable systems. He earned a B.A. and an M.Eng. in electrical and information sciences from the University of Cambridge.

JONATHAN M. SMITH [SM] (jms@central.cis.upenn.edu) is a professor in the Penn CIS Department. He was previously at Bell Telephone Laboratories and Bellcore, where he focused on UNIX internals, tools, and distributed computing technology. He was also a member of a technology transfer team on computer security. At Penn, his current research interest is programmable network infrastructures: "Protocol Boosters" provide a methodology for using such infrastructures, and "SwitchWare" is an idealized programmable infrastructure. He is a member of ACM and Sigma Xi, and has consulted extensively for industry and government. He has patented technology for key-agile encryptors using asynchronous (ATM) networks and ultra high-speed ATM encryptors.

This ability to add resource management to a standard host O.S. is one of the key strengths of Piglet, and enables its easy integration into SQoSH.