BBN Technical Memorandum No. 1278

SENCOMM Architecture

Job No. 12369000

January 24, 2001

Prepared for:
Defense Advanced Research Projects Agency
Information Technology Office
3701 North Fairfax Drive
Arlington, VA 22203-1714

Prepared by:
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
SENCOMM Architecture

Alden W. Jackson, James P.G. Sterbenz,
Matthew N. Condell, Joel Levin, David J. Waitzman
[awjacks, jpps, mcondell, levin, djw]@bbn.com
http://www.ir.bbn.com/projects/sencomm/

BBN Technologies, GTE
10 Moulton St, Cambridge, MA 02138-1191*

Version 1.1

April 26, 2000

Contents

1 Introduction ................................. 3
  1.1 Motivation .................................. 3
  1.2 History .................................. 4
  1.3 Document Roadmap ......................... 4

2 Overview ................................ 5
  2.1 Smart Packets ............................. 5
  2.2 General Management Goals ................. 5
  2.3 General Requirements ..................... 6
  2.4 SENCOMM Operation ....................... 7
  2.5 Representative SENCOMM Applications .. 7
    2.5.1 Active Persistent Traceroute ......... 7
    2.5.2 Multicast Monitoring ................. 8
    2.5.3 Resource Discovery ................... 8
    2.5.4 Multicast/Concast .................... 8
    2.5.5 Remote Ping ......................... 8
    2.5.6 Multicast Tree Core Maintenance ... 9
    2.5.7 Event Processing ..................... 9

3 Component Detail .......................... 9

*This work was sponsored by the Defense Advanced Research Projects Agency issued by the AFRL under contract F30602-99-C-0131.
3.1 SENCOMM Management Execution Environment (SMEE) ......................................................... 9
  3.1.1 SMEE for Active Network Management ................................................................................. 10
    3.1.1.1 The IP Router .................................................................................................................. 10
    3.1.1.2 The Active Node .............................................................................................................. 10
  3.1.2 Management of the Active Node .............................................................................................. 12
  3.1.3 EE Requirements ....................................................................................................................... 12
3.2 Smart Probes .................................................................................................................................. 13
  3.2.1 Smart Probe Requirements ........................................................................................................ 13
  3.2.2 SENCOMM Message Encapsulation ......................................................................................... 13
3.3 Loadable Libraries .......................................................................................................................... 15
  3.3.1 Loadable Library Requirements ............................................................................................... 15
  3.3.2 Message Length ......................................................................................................................... 15
  3.3.3 Sharing Libraries ....................................................................................................................... 16
  3.3.4 Dynamic Loading ....................................................................................................................... 16
  3.3.5 Naming Libraries ....................................................................................................................... 16
  3.3.6 Locating Libraries ...................................................................................................................... 16
  3.3.7 Updating Libraries ...................................................................................................................... 17
  3.3.8 Library State ............................................................................................................................. 17
    3.3.8.1 Naming ............................................................................................................................... 18
    3.3.8.2 Security ............................................................................................................................... 18
3.4 Management API ........................................................................................................................... 18
  3.4.1 Management API for EEs ......................................................................................................... 18
    3.4.1.1 EE Management API Functionality ..................................................................................... 19
  3.4.2 Management API for AAs in Managed EEs ............................................................................... 19

4 Network Security Requirements ..................................................................................................... 20
  4.1 Threats .......................................................................................................................................... 20
    4.1.1 Communication Threats ......................................................................................................... 20
    4.1.2 Operating Environment Threats .............................................................................................. 21
  4.2 Goals and Constraints ................................................................................................................... 21
  4.3 Services ......................................................................................................................................... 22

5 Acknowledgments ............................................................................................................................. 22

6 References .......................................................................................................................................... 22
1 Introduction

This document describes the architecture for the Smart Environment for Network Control, Monitoring and Management (SENCOMM) and its components. SENCOMM comprises a Management Execution Environment (SMEE), which coexists with and controls other execution environments (EEs), and runs on top of the NodeOS [Pet00]. Management applications run in the SMEE. An application is mobile executable code that is delivered to active node within an Active Network Encapsulation Protocol (ANEIP) [ABG+97] datagram.

![SENCOMM Architecture](image)

Figure 1: SENCOMM Architecture

Figure 1 presents the overall SENCOMM architecture, which is compatible with the architectural framework for active networks [Ca98]. Key components of SENCOMM include smart probes and loadable libraries which provide the capability to execute network management programs. Smart packets are the means for transporting SENCOMM smart probes, installing loadable libraries, and exchanging control and security messages. This infrastructure provides the underlying core components to build a flexible, scalable, and extensible basis for SENCOMM.

1.1 Motivation

The future Internet will provide numerous services to a variety of devices across huge, heterogeneous, ad hoc, topologically complex and politically diverse autonomous systems. The size, topological complexity and heterogeneity of even today's Internet is overwhelming current network management protocols and toolkits. In the future Internet this complexity and the associated operations burden will increase significantly. While research to create this vision of the Internet is ongoing, the tools
and protocols necessary for network control, monitoring and management have historically lagged behind the rate of development of other network applications. Current tools are difficult to secure, cannot scale to handle a growing amount of information from an increasing number of nodes, cannot be easily extended to investigate dynamic or unplanned events, nor can they be easily specialized to match the increasingly different roles played by network users and operators.

Current monitoring and management tools are limited in several respects [FV99]. Administrators usually handle management tasks manually. The dependence on manual management leads to centralized use of tools on a small number of workstations. Access to management variables is accomplished using SNMP on each node, which is limited to accessing and changing configuration variables in a query/response model, and often cannot perform the complex management functions which are required. Also, management tools and functionality have generally been added to networking software and applications instead of being integrated into their design.

To solve the limitations of current tools and to properly control and monitor future networks, we believe it is necessary to use Active Networks technology [CBZ98]. In this document, we discuss our rationale and describe our solution – The Smart Environment for Network Control, Monitoring, and Management (SENCOMM). The network contains active routers, which support our proposed architecture. End systems only need to run the applications which generate and launch smart packets and smart probes.

1.2 History

In 1996, BBN received DARPA funding for the Smart Packets project (Contract No. N66001-96-C-8517) in a new ITO program, Active Networks. The goal of the project was to add a flexible and rich programming environment to network management and diagnostic packets. This previous project’s architecture consisted of four parts:

1. The format and encapsulation of smart packets into a network data delivery service
2. The specification of a high level language and its tightly-encoded assembly language
3. A virtual machine, providing a context for executing a smart packet’s program
4. A security architecture

SENCOMM takes advantage of the lessons learned in the Smart Packets work and leverages as much of the Smart Packets technology as is practical, in the context of the current active networks community.

1.3 Document Roadmap

Section 2 provides an overview of SENCOMM, a review of its predecessor – Smart Packets, management goals, and several potential applications which motivate SENCOMM development and use. Section 3 describes the main components of SENCOMM in detail – the management Execution Environment, smart probes, and loadable libraries. Finally, Section 4 outlines the security services necessary to support SENCOMM.
2 Overview

SENCOMM's objective is to implement a network control, monitoring and management environment using active networks. By building SENCOMM on active networks technology, we provide a flexible control and monitoring service that is necessary to track networking technologies and new service offerings. This approach has three advantages. First, the information content returned to the management center can be tailored (in real-time) to the current interests of the center, thus reducing the back traffic as well as the amount of data requiring examination. Second, many of the management rules employed at the management center can now be embodied in programs which, when sent to managed nodes, automatically identify and correct problems without requiring further intervention from the management center. Third, the monitoring and control loop is shortened — measurements and control operations are taken during a single packet’s traversal of the network, rather than through a series of set and get operations from a management station.

2.1 Smart Packets

The Smart Packets project, which preceded SENCOMM, was designed to demonstrate that network management is a fruitful target for exploiting active networks technology. Smart Packets programs are written in a tightly-encoded, safe language specifically designed to support network management and avoid dangerous constructs and accesses. A major focus of the project was development and implementation of a pair of languages (Spanner and Sprocket) that were constrained to the needs of network management and had a very small footprint in the packet. A constrained programming language minimizes the potential risk to the router. Targeting the language to network management permitted the description of network concepts and the manipulation of management data and network packets using primitives in the language.

Smart Packets programs and messages were limited to a single packet. The languages were designed with a concise encoding in order for programs to be deliverable in a single packet and succeeded in allowing complex programs to fit. SENCOMM retains the single packet design for smart probe packets and messages, however loadable libraries can span several packets.

The execution environment for Smart Packets was a virtual machine designed to interpret Sprocket programs. The virtual machine interpreter was not dynamically extensible. Any change in the interpreter requires recompilation of the entire virtual machine.

Smart Packets was designed before the emergence of the current DARPA active networks architectural framework [Ca98]. In contrast to the Smart Packets design, the reference architecture of an active node has several execution environments running on top of a single NodeOS. SENCOMM uses the reference architecture and implements its services within an existing execution environment, including dynamic loading of libraries to extend its base functions. While it is possible to implement the Smart Packets virtual machine within an execution environment, the limitations of the Sprocket interpreter would remain.

2.2 General Management Goals

SENCOMM addresses many of the goals and requirements for managing networks. Below is a set of management goals for SENCOMM:

Management of Active Networks: SENCOMM focuses not only on using active networks technology to monitor and diagnose conventional networks, but also on monitoring and managing active networks.
Dynamic Deployment and Adaptation: SENCOMM provides support for dynamic changes in the network management modules and in the software that are being managed.

Applications-Controlled Management: Management active applications must be able to monitor and control all aspects of the management environment provided by SENCOMM.

Automation of Problem Detection and Resolution: SENCOMM provides the tools to detect situations in the network including congestion, faults and anomalous behavior. Active applications can then be written which respond to these situations. Support is required from both the NodeOS and other instrumentation on the node to monitor both hardware and software resources, i.e., the NodeOS API should expose this instrumentation. We will work with the NodeOS and node instrumentation developers to specify the appropriate programming interface.

2.3 General Requirements

The following list describes a set of necessary, but not sufficient, requirements for SENCOMM.

Packet Delivery: SENCOMM packets must not need any other packet delivery service other than what is provided by the underlying internetworking protocol: IP. For network management processes to be robust, packets to and from these processes must not need any special processing or handling.

Heterogeneous Network: SENCOMM does not require all the nodes to be active in the network. While SENCOMM requires that the management probes run on active nodes, non-active nodes will be managed by management probes using conventional means, such as SNMP. Furthermore, the management node, on which the application that issues probes runs, is not required to be active.

Packet Receipt: SENCOMM management probes must be able to request any incoming packet or a copy from any interface, according to an arbitrary filter specification, which operates on the contents of the packet’s headers or payload. The probes must be able to re-insert a packet into the forwarding path before the initial demultiplexing stage, before the normal routing decision for outbound packets, or directly into an output interface queue.

Message Size: It is a conservative design practice that all control messages be small enough to deliver in a single datagram. Given that the control messages in active networks are most likely to be programs, it is even more important that the entire program be delivered in single datagram. While the use of a reliable packet delivery protocol, e.g., TCP, would offer robust delivery of the control program, the delivery is not time-bound. Many of the situations where control and diagnostic programs are likely to be used are exactly the situations where most reliable packet delivery protocols have the poorest performance. It is a goal of SENCOMM that its control and diagnostic programs fit within an Ethernet frame (typical internet MTU), less any protocol headers needed for packet delivery.

MIB Access: SENCOMM requires access to MIB data on the active node.

Persistent Storage: SENCOMM requires that persistent storage be available for bootstrapping and optionally for longterm storage of loadable libraries. Since the Management EE is booted from the storage, the persistent storage must be secure. SENCOMM requires that the storage device be able to insure the integrity and optionally the privacy of the data across reboots. This requirement does not differ from what is needed for the NodeOS at boot. Both secure bootstrapping protocols and secure persistent storage architectures exist in the literature [AAR98, Nag]. SENCOMM can take advantage of these services, if they exist.
**Distributed Time Service:** SENCOMM utilizes a distributed service for accurate time (such as NTP [Mil92]) so that probes on different active nodes can report distributed data or make a configuration action at the same time. For robustness, distributed time service is *not* required for proper operation of the underlying SENCOMM system. A very coarse level of time accuracy is necessary and required for security services, e.g., for expiring security certificates. SENCOMM can take advantage of a fine degree of accuracy than required for security.

**Secure Management:** While SENCOMM will not focus on security, it is essential that it provides secure access to its management functions.

### 2.4 SENCOMM Operation

The general operation of SENCOMM is as follows:

User-written network management and monitoring programs generate smart probes, which are encapsulated in Active Network Encapsulation Protocol (ANEP) frames. The probes are demultiplexed to the local SENCOMM Management EE (SMEE), which injects the smart probe(s) into the network. A probe can be sent in either end-to-end or a hop-by-hop mode. In end-to-end mode, the embedded active program is executed only at the destination. In hop-by-hop mode, every SENCOMM active node passes the smart probe to its SMEE for execution. The number of available intermediate nodes depends on the several parameters, including evaluation environment support on remote nodes, and if a network overlay is used. The probe contains directives to access loadable libraries of functions on the node, registers to receive incoming packets that meet a filter specification, and optionally inject the packet back into the network. Probe packets can be sent either to unicast or multicast addresses. Obviously the security implications with hop-by-hop mode and a multicast address are critical, but the ability to distribute a smart probe to all routers in a multicast tree is very powerful.

### 2.5 Representative SENCOMM Applications

This section presents several scenarios where the use of SENCOMM offers a capability that either does not exist or can not be easily deployed in the current Internet.

#### 2.5.1 Active Persistent Traceroute

Traceroute [Fre96] is a popular tool that displays the path packets take to a network host. It modifies the TTL field in outgoing probe packets to generate ICMP messages from routers along the path to the host. Traceroute also returns the round trip time for each probe packet, allowing for some profiling delay on the path to the host.

An active traceroute reproduces the base functionality of traceroute with code that records or reports the address of the each active hop on the path. Since active traceroute executes at each node, it is able to do much more, such as return a vector of information at each active node. For instance, the vector could include both the incoming and outgoing interface addresses for the probe packet, the round trip delay, and a snapshot of the state of local resources.

An active *persistent* traceroute stays resident in each router after the initial probe packet is forwarded. By staying resident, the probe can monitor local resources and issue reports, either periodically, at the request of the management system, or when a local threshold is crossed. All probes in the network are uniquely identified, allowing the management system to specify which probe to contact.
2.5.2 Multicast Monitoring

Currently, several multicast tools exist to trace and map multicast routing in the Internet [Fre95b, Fre95a]. The tools are based on tracing support built into multicast routers, because the use of ICMP is forbidden for multicast traffic. Traces are run "backwards" from receiver to source. A trace query is sent from the receiver to its multicast router, which builds a trace response message, fills in its data and unicasts back along the path to the source. Each additional multicast router on the reverse path adds to the trace response packet, adding information at each hop. The last hop router (directly connected to the the source) sends the completed trace via unicast to the destination address in the query (which may be different from the multicast tree source).

Active management can impact multicast monitoring in several ways. First, the existing tools can be enhanced by the deployment of a monitoring probe that collects data on the multicast groups to which the node has subscribed, or its administrative scoping state. Active monitoring data is likely to be more accurate as it does not represent a snapshot. Second, the data collected could be used to trigger a standard multicast path query when a threshold is crossed, e.g., the packet drop rate on a link for a group exceeds some value. Third, smart probes can replace the standard multicast path query. A smart probe can be configured to trace not only the unicast path back to the source, but also, from the source multicast router, to map out the entire multicast tree.

2.5.3 Resource Discovery

As the development of services within the network continues, there is a growing need to identify the availability and describe network-based resources. For example, a user application might need for a transcoding service, but not know if or where it is offered. Active management can offer two mechanisms that can assist with this problem. First, a smart probe can create a resource registry in any active node. The registry would accept registry notifications from local hosts and exchange registry information from other registries. For robustness, the registry could be replicated by copying itself (the program, not the data) to other active nodes. Second, the registry would answer resource queries or forward it to other registries, much in the way DNS forwards queries up to root servers.

Another active method to aid resource discovery is to send probes to find the desired resources. Using probes and active routers, the "best" collection of resources can be found according to a set of user-defined and network criteria. Once the resources are found, the probes remain to monitor the performance of the ensemble. In cases of changing network topology or behavior, the probes can initiate a search for another set of resources that meet the criteria.

2.5.4 Multicast/Concast

While multicast is one-to-many (or occasionally many-to-many) packet delivery, concast [Cal99] is many-to-one packet delivery. Many-to-one packet delivery implies that data synthesis, reduction, or fusion techniques are available. Smart probes allow the description and implementation of the packet data fusion engines. A filter, specified with the data fusion engine, directs incoming packets to the correct smart probe. Combining multicast with concast allows us to deploy monitoring and fusion probes in the network via multicast, and fuse the responses back to the source, minimizing data rate and network traffic.

2.5.5 Remote Ping

A smart probe can be sent to a remote node and be told to ping some other destination and sent the data back to the initiator. This is similar the Distributed Management (DISMAN) work in the
The pings themselves could be standard ICMP Echo Request packets, or a particular number of bytes to the TCP discard port, or a timed HTTP GET. This functionality is particularly useful when a user complains of slow traffic to the Network Operations Center (NOC). A NOC operator initiates a ping from the user's end system. Even if the actual end systems the user is using are not configured to allow responding to ICMP echo requests replies (such as because of a tight firewall policy), active routers near the ping source and destination exchange ping packets. In case a user reports an intermittent problem, those active routers could continue their pinging throughout the day (at a low bandwidth) and notify the operator when the delay exceeds a threshold. An advantage that an active router has over a DISMAN implementation is that in response to an intermittent problem, the remote ping probe can take other actions, such as keeping a history, ping'ing with differing packet sizes or types of packets, or starting a traceroute.

2.5.6 Multicast Tree Core Maintenance

Smart probes could be used to maintain Rendezvous Points (RP) in PIM-SM [Eea98] or other kinds of state that must be manually configured within the core of a multicast router infrastructure. The active nature allows a probe to affect all active routers in a path, rather than the management system having to specify each individually. Coupling this with distributed time and all the routers in a path can make a change at nearly the same time.

2.5.7 Event Processing

Events include both SNMP Traps and other asynchronous notification mechanisms, such as Unix-style syslog logging. SNMP makes little use of events due to a belief that it is inappropriate to rely upon delivery of packets sent over an unreliable transmission channel, UDP. As networks scale and the number of types of events that might be generated in the network increases (because of increasing complexity of the managed nodes), event based management will become more important. But the events require filtering as early as possible in the network in order to reduce the number of non-relevant events sent to the management stations.

Smart probes can tackle this problem in the following way: if a smart probe running on a node can receive events reliably from the node it is running upon and has an access to accurate fine-grained timestamps, it can then filter and correlate the events as appropriate for the needs of the management station that sent the smart probe. Reliable access to the events requires secure access to the NodeOS. The probe needs mechanisms to insure that the event message is from the NodeOS and not from some other entity. The smart probe can then either use UDP, TCP, or some other protocol to get the filtered events back to the management station or to an intermediate node in the network for further processing. The use of a reliable protocol, such as TCP, is recommended.

3 Component Detail

As discussed in Section 2, SENCOMM has three major components, a Management Execution Environment (SMEE), Smart Probes (SPs), and Loadable Libraries (LLs). The SMEE is also responsible for the management of other Execution Environments. The architecture and requirements of each of the four components is described in detail below.

3.1 SENCOMM Management Execution Environment (SMEE)

The SMEE has two primary functions:
1. Providing the Execution Environment for smart probes engaged in performing management activities for the active network

2. Management of the active node itself, including other EEs and the NodeOS

Responsibility for the second may lie with, or is at least shared with, the NodeOS. To the extent the SMEE participates in node management, the requirements for this function overlap substantially with the first function.

3.1.1 SMEE for Active Network Management

In addition to all the capabilities provided by the NodeOS to any arbitrary EE, the SMEE requires additional facilities in order to enable smart probes to function, as described next.

3.1.1.1 The IP Router

The SMEE must be able to make the standard MIB-2 information about the basic IP router portion of the active node available to smart probes. Many of the management functions performed by smart probes are concerned with collection or evaluation of, or computation based on values of MIB variables. Any arbitrary MIB object supported by the IP router should be accessible by specification of its OID, including:

- System objects
- Network interface parameters
- Protocol stack parameters
- Routing table information

The SMEE should be able to request a notification of status changes, such as network interface states or changes to the kernel routing table. Modern implementations of routing software, e.g., `gated`, permit the registration for this information.

A smart probe may need to affect the router configuration, so the SMEE requires access to the router configuration. For example, a smart probe intended to mitigate a congestion problem may need to modify the allocation of buffers to an interface.

3.1.1.2 The Active Node

Smart probes will be able to collect and process information about the active networking portion of the active node.

The Active Node Operating System (NodeOS) [Pet00] provides mechanisms for creating and destroying a flow, the abstraction which aggregates computation and communication resources for some purpose. The NodeOS itself is the root flow, and individual EEs are child flows of the NodeOS. The NodeOS allocates computation (threads), memory pools and bandwidth resources (channels) to a flow according its configured needs as requested by an EE, subject to trust established between an EE and the NodeOS by active network security policies. Any EE may request that active packet filters be set up on its input channels at the time of channel creation.
The SMEE will have use of all the standard EE facilities for its own use. Thus it will be able to establish its own memory pools, set up input and output channels for receipt and forwarding of smart probes. In order to perform its management task, the SMEE has additional requirements for interface with the NodeOS in support of smart probes.

Information that the SMEE should be able to provide to a smart probe, and which thus must be provided by the NodeOS, includes:

**Flows:**
- flow ID
- status
- credentials
- thread pool allocation
- memory pool allocation
- channel list
- packet statistics
- utilization
- error statistics

**Channels:**
- channel ID
- channel status
- type (in, out, out)
- flow ID
- protocol InChannels and OutChannels have additional information associated with them
- packet statistics
- interface utilization and error statistics for channels
- packet statistics for filters

**InChannels:**
- channel parameters above, plus demux key and address spec (i.e., filter information)
- thread pool allocation
- memory pool allocation

**OutChannels:**
- Channel parameters above, plus link QoS specification

**OutChannels:**
- union of those for InChannels and OutChannels

**File system:**
- available capacity (file system and OS resources)
- file system activity (reads-per-second, writes-per-second, etc.)
The SMEE requires enhanced packet copying and filtering from the active node. To fulfill its functions of monitoring, analyzing and managing network traffic, the SMEE must be able to obtain copies or intercept arbitrary packets. It must have the capability to reinsert the possibly modified packets ahead of the NodeOS active packet filters, insert them back into the forwarding path, or queue them directly into an output channel.

To accomplish this the SMEE must be able to specify arbitrary filters to be applied to the node's entire packet stream ahead of the standard active packet filtering that assigns incoming packets to EEs or InChannels. These SMEE filters must be able to access any field or pattern in the entire datagram, header or payload. The capabilities described here are not currently available in the NodeOS specification [Pet00].

3.1.2 Management of the Active Node

While not the focus of the SMEE, satisfying the requirements for network management as described above provides most of the abilities necessary for management of the active node itself. In order to enable the SMEE to manage the active node, additional capabilities are required by the SMEE:

NodeOS management requires the ability to:

- Modify status, attributes, and configuration parameters of each EE
- Modify NodeOS configuration to affect:
  - Memory pools (memory allocated to each flow, etc.)
  - Thread pools (CPU utilization)
- Modify channel status, attributes and configuration to affect:
  - Packet filters
  - Memory pools
  - Thread pools
  - Assigned bandwidth or QoS
- Evaluate filter requests for completeness and overlap – if a conflict with SMEE filter occurs, the SMEE filter has priority.

At boot time, the SMEE is automatically initialized; it can then initiate other EEs, obtain certificates, establish credentials, etc.

For management of specific EEs and their Active Applications (AA)s: The SMEE requires use of an inter-EE API. This is discussed in Sections 3.4.1 and 3.4.2.

The active network architectural framework [Cal98] allocates security management functions to the management EE, in particular maintenance of the security policy database of the active node. While the SMEE will support secure access to all network management functions, full implementation of security management for the active node is beyond the current scope of the current SENCOMM project effort.

3.1.3 EE Requirements

The SMEE will be able to monitor flows, channels and active applications within individual EEs to the extent that EEs make this information available, and depending on the availability of inter-EE
communications (see section 3.4.1). Access methods for management of specific EEs can be adapted by development of a loadable library interface module.

3.2 Smart Probes

Smart probes are executable programs that perform management functions. A smart probe is delivered to the active router inside an Active Network Encapsulation Protocol (ANEP) datagram.

3.2.1 Smart Probe Requirements

The following are the requirements for smart probes in the SMEE.

Residence: - The probe must be able to function after the packet that carried it has been forwarded.

Unique name: - Each probe must have a globally unique name that distinguishes itself from other probes.

Single datagram: - It is a common design principle that control messages should fit within a single datagram in the network. To preserve control channel robustness, a small footprint is needed to ensure that the control programs of reasonable complexity can fit in single packet. For SENCOMM, ALL smart probes will to be encoded in a datagram that will fit within a single Ethernet frame.

Library access: - Probes must be able to access the methods contained in loadable libraries. Probes must be able to specify the minimum version of a particular library for use.

Sleep/Wakeup: - Probes should be able to register for an event, e.g. a packet matching a filter specification, with the EE and hibernate until the event occurs.

Soft State: - In general, probes should not stay around forever. Probes should use packet frequency or other implicit timers to determine life-time. Nevertheless, probes must be able to have an explicit life-time. The life-time should be able to take on the value of infinity for probes with the appropriate authorization.

3.2.2 SENCOMM Message Encapsulation

In general, SENCOMM messages are encapsulated in a ANEP datagram, which is transported using UDP/IP or TCP/IP.

```
+-----+-----------------+-------------------+
| IP  | UDP / TCP / ANEP | SENCOMM           |
+-----+-----------------+-------------------+
```

The ANEP header format [ABG+97] is:

13
The values of the first three fields of the ANEP header for SENCOMM are as follows:

- **Version:** 1
- **Flags:** 0 (if **Type ID** is unknown, forward packet on default path)
- **Type ID:** 25 (assigned by the ABOCC)

The following general options are used in the ABone: Source Identifier and Destination Identifier. The use of these options is not suggested if ANEP is used as a transport protocol, as the data included would replicate data in the IP header.

A set of security options has been proposed [Mur98, Mur99] for both hop-by-hop and stream active networks communications where portions of the packet may change from hop-to-hop. The proposed options, Hop Integrity, Origination Signatures, Credential Fields, and Static and Varying Payload, would allow the verification of data integrity of the static portion of the packet and hop-by-hop verification of the changes in the variable section of a packet. The options also permit the transport of both multiple signatures and multiple sources of credentials for the verification of multiple signatures.

These security options are mostly sufficient to provide the authentication required by SENCOMM. However, there are two deficiencies that should be addressed. First, the proposed security options provide authentication for the previous hop node and the originating node. Adding an option to provide authentication for every active node that has handled the data would add an extra degree of security. This can be accomplished by modifying the proposed security options to include an option that provides a signature over the static payload (and possibly the previous signatures in the option) for each node that handles the packet. This provides a chain of signatures proving where the data has been.

Additionally, the proposed security architecture depends upon the DNS to provide public-key certificates. While this is acceptable in the general case to be consistent with its conservative design, SENCOMM should not depend upon services such as DNS to be available when management functions demand them. SENCOMM packets, therefore, must provide a means for requesting and exchanging certificates to support the security architecture. SENCOMM will provide an internal mechanism to request and retrieve certificates. See the SENCOMM Design document for more information. Given that this mechanism may exist outside of and not depend on the availability of network services, the system performance may degrade when it is used. This is an appropriate tradeoff for network management where the goal is to preserve secure management over preserving system performance.
3.3 Loadable Libraries

Management libraries differ from smart probes in that they are not executable pieces of code in their own right. The libraries must be invoked by a smart probe to be executed. They are designed to provide classes, methods, and data structures to be used by one or more smart probes. This allows common methods to be shared among multiple smart probes, so that each probe does not have to re-implement the contents of the library, reducing the size and complexity of the probes.

Libraries are loaded when the SMEE is started or as needed by smart probes. Each probe identifies the libraries it depends upon and possibly where the library may be found. Any required libraries that are not currently loaded are fetched and are dynamically loaded before executing the probe.

3.3.1 Loadable Library Requirements

The following are the requirements for loadable libraries in the SMEE:

**Sharable:** Multiple smart probes must be able to share a single library. They must be able to access the library concurrently or sequentially.

**Dynamically Loadable:** The SMEE must be able to load new libraries into a running environment.

**Unique Name:** Each library must have a globally unique name that distinguishes itself from other libraries.

**Version:** Each library must contain a version number to distinguish itself from other versions of the same library.

**Separation of State:** Each smart probe that uses a loadable library must be able to execute the library in its own address space so that it is not affected by other smart probes executing the library.

**Sharable State:** Loadable libraries must provide a means for correctly and securely sharing state between smart probes.

3.3.2 Message Length

While it is important for robustness that smart probes be deliverable in a single network datagram, not all information that is needed by SENCOMM is required to fit into a single packet. Data returned from a smart probe may exceed the size of a single packet. Also, loadable libraries are likely to exceed the space available in a single network datagram. Limiting the size of a library would result in an explosion in the number of libraries and the need to manage many small libraries with related functionality.

SENCOMM will relax the single packet size requirement for loadable libraries because they provide capabilities that are shared among multiple smart probes, and it makes architectural sense to keep all related methods in the same library. While SENCOMM will generally use the standard reliable transport protocol (TCP) to deliver loadable libraries, we recognize that there exists the possibility that a management program will request a library that must be retrieved and the network will not support the establishment of a reliable transport connection. We are investigating the options available, from implementing a reliable transport protocol specific to SENCOMM to specifying the use of local libraries only and trapping the missing library as an exception.
3.3.3 Sharing Libraries

Multiple smart probes may depend upon a similar set of functions. For example, several probes may use concasting for different purposes, each using the same concasting algorithm with a different filter and reduction method. Each probe could implement the classes and methods needed for the common functionality, however, this would unnecessarily increase the size and complexity of each probe.

Instead, the SMEE will provide access to libraries of functions that can be used by any smart probe. Smart probes identify the libraries they require so that the EE can confirm that they are available and load them, if necessary. The probe may then use any of the classes or methods available in the library. If the library is not available, then the smart probe cannot be executed.

In the concasting example, the SMEE provides a concasting library. The smart probes may then execute concasting by executing a library call with the required filter and reduction function. This greatly reduces the amount of concasting code that each smart probe must provide.

3.3.4 Dynamic Loading

The SMEE may load many libraries on initialization, such as those that are needed for running the EE or those that are likely to be used by smart probes. These initially loaded libraries may be specified in configuration files.

However, it is impossible for the SMEE to predict at startup which libraries will be required by all smart probes that will eventually run on the SMEE. Even if the SMEE could predict which libraries to load, it might be prohibitively expensive in resource utilization to do so. Additionally, over time libraries will likely be updated and it would be unreasonable to have to restart the SMEE to incorporate each update.

The solution is to dynamically load libraries as they are needed. Dynamic loading is accomplished in Java by sub-classing the ClassLoader class, which reads in a byte stream containing new classes to be loaded and loads them into the currently running process.

3.3.5 Naming Libraries

Every library must have a unique name so that smart probes can unambiguously refer to the library by name. There must also be a version number associated with the library name.

SENCOMM will name libraries using Uniform Resource Names (URNs) [Moe97]. The name will include the name of the library followed by a version number. The syntax must also allow the library to be named without a particular version number to refer to any version of the library.

Smart probes must identify which libraries, if any, must be loaded before the probe can be executed. If a particular version of the library is required, then the probe must identify the version. It must also be possible to specify a library older or newer than a particular version.

3.3.6 Locating Libraries

Since there currently exists no infrastructure to convert URNs to Uniform Resource Locators (URLs), it is necessary to provide a way to locate the named libraries. Additionally, the URL provides a hint to the protocol to use to fetch the library. There are several possible ways for an EE to get this information, which SENCOMM will use in combination:
• The SMEE should always first check to see if the library is resident on the active node.

• The SMEE may be configured with one or more servers that may have a copy of the library. For example, a trusted server can deploy libraries that meet performance or security criteria.

• Each smart probe may provide one or more URLs along with eachURN that may be interpreted as hints where the library can be found.

• The SMEE may provide a method that allows the code carried by smart probes to indicate the libraries that it depends upon and URLs for each library. When the code is run, the methods initiate the library fetching from the indicated locations.

3.3.7 Updating Libraries

Over time, it will be necessary to load new versions of libraries. This may be necessary on a node-by-node basis or over a large number of active nodes at one time. SENCMM will provide mechanisms for both these scenarios.

The first case will be handled using the normal mechanisms for handling loadable libraries. A request for an updated library will result in fetching the new library, if necessary, and loading the library. This case may occur when a smart probe requires a newer version of a library that is already loaded.

In the second case, an administrator would update the libraries on a set of active nodes. While this case could be viewed as multiple instances of the case described above, a mechanism should be designed to perform such management tasks in an efficient manner. There are several possible solutions that could be designed using smart probes. A probe could be created to find all nodes that require updating. At each node the probe would fetch the new library and have the SMEE load it. Another possible solution is to use smart probes to implement concasting. The probes could compile a list of all nodes which require the new library version which would then be used to update the library on each node.

If a different version of a library is currently loaded at the time it is updated, it is necessary to start a new instance of the class loader. Otherwise, Java will use the previously cached version of the library, even if both are concurrently loaded. Libraries that are not in use by any probes may be flushed from the cache.

3.3.8 Library State

Each probe loads any methods or data structures in each library into its current process or thread. The execution and state maintained for the library is separate for each probe using the library. This helps prevent one probe from accidentally or maliciously altering the execution of the library in another probe.

However, state may need to be shared between instances of a library or execution of a smart probe. This can be accomplished by writing state to a data structure in a loadable library. The library may then be dynamically loaded by another currently executing probe, or a probe that executes at some time in the future. This process is similar to using Unix domain sockets, except that they may exist over an extended period, between execution of probes or library methods.

There are two important requirements to support dynamic shared loadable libraries. Security is necessary to provide access control to the data in the library. A globally unique naming mechanism is needed to allow the library to be referenced by the probes. If the library is not temporary, this name will have to be advertised to other nodes.
3.3.8.1 Naming

Knowing the names of loadable libraries that are dynamically generated by libraries or probes varies in difficulty for different users of the library. For some users of the library it is straightforward to use the loadable library methods described above. These users know the name of the library when it is created. Loading the most recent version of the dynamic library finds the latest version.

One example is a library that writes a data library to share information between instantiations of the library. The library defines the name of the data library and knows where to look for it. Another example is a follow-up smart probe that is initiated from the same source as its predecessor. The initial probe may leave state for the second probe. Since both probes are created and sent by the same source, the second probe can know where to expect the data library.

If in the latter example, the second probe is from a different source and needs to use data left by the first probe, it must determine the name of the library where the first probe stored the data. However, the second probe does have information that can be used to identify the name of the required library, such as a particular object.

3.3.8.2 Security

Security for data libraries consists primarily of providing access control over which libraries and probes can read and/or write a library. A basic file locking mechanism is also necessary.

Some data libraries may only be read or written by certain probes or other libraries. The NodeOS should provide hooks to provide this level of access control, however, for those that do not the SMEE will have to provide this functionality. Access control lists may be stored in the table with the naming mapping discussed in the previous section. The mechanism for loading libraries will then have to check the permissions before loading a requested library. The library functions for writing to a library will also have to make a similar check.

Whether the access control mechanisms are provided in the NodeOS or the SMEE, the specification of the access control list and the policies it embodies are controlled by the SMEE. If the NodeOS is to provide the mechanisms, then hooks should be added to provide it securely add/modify/delete list and policy entries. SENCOMM prefers the use of a role-based authorization system that would allow the dynamic mapping of a authenticated principal to a role, where the access control decision is made on the role. Mapping the principals to roles allows flexibility in making access control decisions and minimizes the management overhead of the access control system.

The SMEE also must assure that there are not errors with multiple probes simultaneously accessing a library. A locking scheme will be required to arbitrate access between multiple smart probes.

3.4 Management API

This subsection describes the API between the SMEE and other EEs resident on the node.

3.4.1 Management API for EEs

The SMEE is responsible for managing other EEs as well as the active node. In order to accomplish this, the SMEE must be able to access EE specific management functions and variables for each EE running on the active node. The SMEE architecture must overcome the difficulties of communicating with another EE that is in a different flow in the NodeOS and may be written in a different language.
It must also overcome the difficulty that it will not know what EE specific functions must be supported at startup. The architecture should require few changes to the NodeOS to support the SMEE, so that if the changes are not made, the management architecture will still operate. However, the architecture can include optional changes to the NodeOS that would to provide better services for inter-EE communication.

Loadable libraries provide the mechanism for this interface. Each EE to be managed should provide a loadable library that includes function wrappers for all the management functions that it has available. This library is written in Java (the language of the SMEE) regardless of the language in which the particular EE is written.

The function wrappers provide an interface for smart probes to manage the EE. Each wrapper provides the same prototype as the function that it is wrapping. However, instead of implementing the function, it calls the actual function in the appropriate EE. This involves using the appropriate interface for inter-EE communications.

Ideally, these wrappers could take advantage of bi-directional inter-EE communications interfaces provided by the NodeOS. However, this functionality is not currently part of the standard NodeOS architecture, so SENCOMM will support other options.

Since the EE that is providing the wrapper loadable library controls both sides of the interface, it is free to use whatever inter-EE interface it prefers. However, it is not reasonable to expect each EE to implement its own inter-EE interface. SENCOMM, therefore, needs to provide at least one example inter-EE interface. A minimum set of required functionality for the interface must be identified.

### 3.4.1.1 EE Management API Functionality

Due to the very flexible nature of the EE management API, there are few specific requirements. However, experience with an implementation may show that there are certain functions that are common among many EEs or may even be required for facilitating operation of the SMEE. If such functions are found useful, some function IDs may be reserved in the future for the functions and the SMEE can implement its side of the API for these functions.

An example of such a function is \texttt{EE\_is\_running()} which is a simple function that determines whether a particular EE is currently running on an active node. The check may be made as part of monitoring activity on the node or as an initial check before executing other code that requires the EE.

### 3.4.2 Management API for AAs in Managed EEs

The SMEE does not directly provide an API for active applications that execute in managed EEs. Each EE may provide its own management interface to AAs. The EE would then have to generate a loadable library of the AA interface functions and pass them to the SMEE. This could be either established upon startup or dynamically, depending upon the interface the EE provides to the AAs.
4 Network Security Requirements

4.1 Threats

Several of the classical communication threats to network protocols are applicable to the network management problem and therefore would be applicable to a SENCOMM security architecture. In addition to the threats to communications, there exist threats that occur by executing code contained in smart probes. The correct operation of the network requires that individual routers are not subverted from forwarding packets correctly. Thus, protecting the correct operation of the router and its configuration is also a goal of these requirements. This section discusses the principal threats to SENCOMM and threats which are of lesser importance.

4.1.1 Communication Threats

The principal communication threats against which a security model should provide protection are:

**Modification:** the danger that some unauthorized entity may alter in-transit messages generated on behalf of an authorized user, and

**Masquerade:** the danger that operations not authorized for some user may be attempted by assuming the identity of another user that has the appropriate authorizations.

A few secondary threats are also identified. The security requirements outlined in this document identify the need for protection against:

**Sequence Modification:** the danger that messages may be maliciously re-ordered to an extent which is greater than can occur through the natural operation of a sub-network service. SENCOMM is based upon a connectionless transport service which may operate over any sub-network service. Protecting against this threat may be required as a matter of local policy,

**Disclosure:** the danger of eavesdropping on the exchanges between managed agents and a management station. Protecting against this threat may be required as a matter of local policy, and

**Denial of Service:** in particular, some types of flooding attacks where many copies of a packet are maliciously sent to a receiver to consume resources that would otherwise be available for legitimate uses.

There are at least two communication security threats that a SENCOMM security architecture need not protect against. The security requirements defined in this document do not request protection against:

**Denial of Service:** a broad range of attacks, not including the flooding attacks described above, by which service on behalf of authorized users is denied. Indeed, such denial-of-service attacks are in many cases indistinguishable from the type of network failures with which any viable network management protocol must cope as a matter of course, and

**Traffic Analysis:** the danger that observing the flow of traffic infers information on the contents of the traffic.
4.1.2 Operating Environment Threats

The principal operating environment threats against which a security architecture should provide protection are:

**Unlimited consumption**: the danger that a program can consume node resources without bound,

**Unlimited access**: the danger that an unauthorized program can access or effect sensitive areas, and

**Unsafe evaluation**: the danger that a fault during program execution can cause harm to the node evaluating the program.

Reference monitors or low-level instrumentation in the NodeOS can be used to protect against unlimited consumption. The NodeOS architecture specifies that enforcement mechanisms should exist control running programs. Access control mechanisms, if implemented at the correct layer, should prevent unlimited access. Dangers from unsafe evaluation can be addressed by various methods, including evaluating the program in a tightly controlled sandbox to verifying that a proof carried with the program meets the security/resource policies of this node.

The requirements described in this document do not include protection against:

**Message Replay**: the danger that a valid message can be stored and resubmitted to a receiver without notice. There are two levels of danger: arbitrarily old messages and recently received messages.

4.2 Goals and Constraints

Based on the foregoing account of threats in the network management environment, the goals of a security architecture meeting these requirements are as follows:

- Verify that each received message has not been modified during its transmission through the network.
- Verify the identity of the user who generated the received message.
- Verify that the user who generated the received message or program is authorized to consume the node resources required for execution of the probe.
- Limit the ability of a running program to consume node resources.
- Protect the node running a program and the integrity of its configuration from program faults.
- Protect, if necessary, the contents of each transmitted message from disclosure.

In addition to the principal goal of supporting secure network management, the design of this security architecture is also influenced by the following constraints:

- Neither the security protocol nor its underlying security mechanisms should depend upon the ready availability of other specific network services (e.g., a specific key management protocol or Domain Name Service (DNS)).
- A security mechanism should entail no changes to the basic SENCOMM philosophy as discussed in the introduction.
4.3 Services

The communication security services necessary to support the goals of this security architecture are as follows:

**Data Integrity**: ensuring that the data is transmitted without undetected alteration.

**Data Origin Authentication**: ensuring that the claimed sender is in fact the actual sender.

**Access Control**: ensuring that the claimed sender and associated probe is authorized to consume the resources needed by the probe.

**Limited Replay Protection**: ensuring that duplicates of recently received valid messages are not accepted. Note that message reordering is not dealt with and can occur in normal conditions.

**Data Confidentiality**: ensuring that the intended recipients know what is being sent but unintended parties cannot determine what was sent.

Supporting the services is out of the current scope of the SENCOMM project. However, SENCOMM will be designed so that these services may be supported by the design in the future.

5 Acknowledgments

The authors are grateful to the following people for ideas and stimulating discussions – The original Smart Packets project team: Craig Partridge, Beverly Schwartz, W. Timothy Strayer, Wenyi Zhou, and R. Dennis Rockwell. The DARPA Active Networks community, including: Bobby Bhattacharjee, Bob Braden, Ken Calvert, Ted Faber, Jay Lepreau, Richard Liston, Russ Mundy, Sandra Murphy, and Ellen Zegura. Finally the DARPA Active Networks Program Managers: Hilary Orman and Douglas Maughan.

6 References


