1999-03-01

Prototyping of an active and lightweight router

Kaplan, Namik

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/8871

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PROTOTYPING OF AN ACTIVE AND LIGHTWEIGHT ROUTER

by

Namik Kaplan

March 1999

Thesis Advisor:	Geoffrey Xie
Second Reader	Chris Eagle

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: PROTOTYPING OF AN ACTIVE AND LIGHTWEIGHT ROUTER

5. FUNDING NUMBERS

6. AUTHOR(S) Namik Kaplan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(maximum 200 words)***

A new network management system named Server and Agent based Active Management (SAAM) has been proposed [Ref: 1]. SAAM can locate and fix network problems much more quickly than today's systems. Stand-alone routers are used in current network architectures. In contrast, SAAM employs dedicated servers that collect packet performance information from the routers and use the collected information to predict, detect and respond to network problems. In other words, SAAM relieves individual routers from most routing and network management tasks. SAAM allows the development of a lightweight router. The primary goal of this thesis is to prototype a lightweight router that is suitable for the SAAM architecture. The Active Networking approach was explored. Active Networking refers to the addition of user-controllable computing capabilities to the network. The result of this thesis is a lightweight router running on a Linux machine. The router is connected to the Active Network Backbone (ABONE) by using a software package called Active NETworks Daemon (ANETD). ABONE is an experimental wide area network, where more in-depth research of SAAM router and server can be conducted. All major active network programming languages and their underlying support were evaluated. Verification of the lightweight router concept was conducted using server-probing experiments. The results demonstrate that it is straightforward for a SAAM server to collect performance information from lightweight routers that support active networking.

14. SUBJECT TERMS Networks, Active Networking, Internet, SAAM, ABONE, ANETD, ANTS, PLAN

15. NUMBER OF PAGES
142

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
298-102

Approved for public release; distribution is unlimited

PROTOTYPING OF AN ACTIVE AND LIGHTWEIGHT ROUTER

Namik Kaplan
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1999

p
ac
ra

ABSTRACT

A new network management system named Server and Agent based Active Management (SAAM) has been proposed [Ref: 1]. SAAM can locate and fix network problems much more quickly than today's systems. Stand-alone routers are used in current network architectures. In contrast, SAAM employs dedicated servers that collect packet performance information from the routers and use the collected information to predict, detect and respond to network problems. In other words, SAAM relieves individual routers from most routing and network management tasks. SAAM allows the development of a lightweight router.

The primary goal of this thesis is to prototype a lightweight router that is suitable for the SAAM architecture. The Active Networking approach was explored. Active Networking refers to the addition of user-controllable computing capabilities to the network.

The result of this thesis is a lightweight router running on a Linux machine. The router is connected to the Active Network Backbone (ABONE) by using a software package called Active NETworks Daemon (ANETD). ABONE is an experimental wide area network, where more in-depth research of SAAM router and server can be conducted.

All major active network programming languages and their underlying support were evaluated. Verification of the lightweight router concept was conducted using server-probing experiments. The results demonstrate that it is straightforward for a SAAM server to collect performance information from lightweight routers that support active networking.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT/DEDICATIONS

I would like to express my deepest gratitude to Professor Geoffrey XIE whose support, guidance, knowledge and enthusiasm have been a constant inspiration to me. His patience and positive attitude were invaluable to this research. I am also deeply indebted to my second reader LCDR Chris EAGLE for his support.

I. INTRODUCTION

A. SAAM(SERVER AND AGENT BASED ACTIVE NETWORK MANAGEMENT)

In current networks, when a user notices a network problem with his/her application, he or she will notify a network administrator. The network administrator then uses some network management system to identify the problem by querying various network nodes (e.g., routers) for usage information such as whether a particular link is up. After the network administrator has located the problem, he or she will attempt to solve the problem by reconfiguring software or hardware. This process can require anywhere from a few minutes to several days, far too long for the response time requirements of the Next Generation Internet (NGI), which must support real-time applications and provide stringent Quality of Service (QoS) to individual users.

To address the above problem, a new management system, named Server and Agent based Active Management (SAAM), has been proposed. [Ref: 1]. SAAM can locate and fix network problems much more quickly than today's systems. Current network architectures requires each stand-alone router to participate in almost all routing and management tasks. This approach is becoming too inefficient to meet the stringent Quality of Services (QoS) requirements of the NGI. SAAM addresses this problem by relieving individual routers from most routing and network management tasks. In particular, SAAM employs dedicated servers that collect packet performance information from routers and use the collected information to predict, detect and respond to network problems.

To illustrate the approach used in SAAM, consider road traffic monitoring and control during commute hours in a large city such as New York. In this case radio stations are the main management entities. They send out helicopters to monitor traffic on roads in their respective coverage region. The information from the helicopters is aggregated at the stations and advice messages are broadcast in real-time to commuters. The use of helicopters has several advantages. First, a helicopter maintains a global view of a region, making it possible to monitor traffic over long routes ("paths"); such monitoring is required to produce real-time advice such as "It will take about 30 minutes to go to place A from place B following road X." Second, a helicopter can spot traffic trends, predicting or detecting congestion

before the problem grows; such early warning is key for effective traffic control. In contrast, each individual motorist can only monitor traffic within a short radius.

Current network management systems behave like road traffic monitoring. They depend mostly on reports from individual motorists. SAAM follows the helicopter model. Specifically, SAAM maintains a global view for each network region in terms of packet performance as well as resource usage and availability. As such, quick responses and proactive control are possible as seen in the Helicopter example.

It is obviously inefficient to require each router in the region to maintain this global view. Unfortunately this is exactly what some current routing algorithms (e.g., OSPF) are doing. SAAM addresses this problem by employing a dedicated server ("helicopter") that will manage the global view for the region. With the global view, the server becomes a much more appropriate place than the router to perform decision-making tasks such as routing and resource reservation. In other words, SAAM allows the development of a lightweight router that delegates most decision-making to a SAAM server. In this thesis I will investigate how to prototype such a lightweight router using Active Networking.

B. THE LIGHTWEIGHT ROUTER

SAAM deploys dedicated servers, at least one for each Autonomous Region, that perform decision-making tasks for the routers (Such servers are formally called SAAM servers). As a result, we can design a router that is lightweight in terms of functionality requirements for a SAAM environment.

Although lightweight, such a router should be carefully designed to ensure good network performance. There are minimum tasks that the SAAM routers will be performing.

Specifically, there are four tasks that the lightweight router must perform even with the assumption that the SAAM server makes decisions on routing, resource reservation, network management and security.

1) The router will perform the task of packet forwarding. Actual data packets will not go through the server. They will be handled by the routers as before.

2) The SAAM server will make the routing decisions for routers. However, the router must be able to accept server commands to update its routing table. In general the router should be able to act based on the server commands. The situation is illustrated in Figure 1.1. The server sends instructions to the router; the routers carry out the instructions. The server can send a program to a specific router and that router can run this program. The router should also have the execution platform for that instruction. Any execution platform can be loaded to any router by the server with the active networking approach. This program, which is sent by the server, is called a resident agent. This program has the ability such as to change the state of the router to update its routing table.

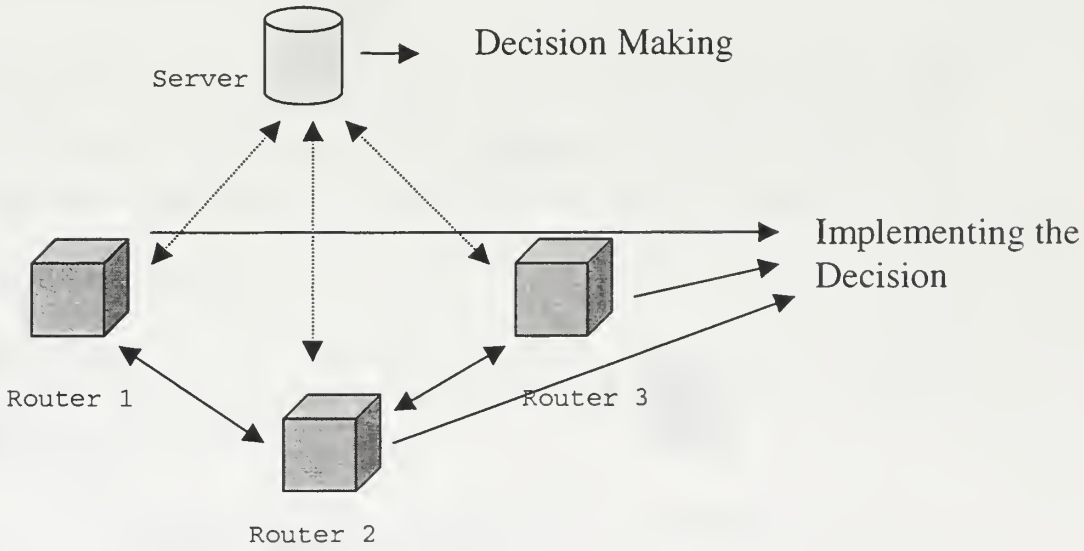


Figure 1.1 Server - Router Relation.

3) The router must be able to measure the packet performance (delay, loss, etc.) of its links. It must be able to pass such

performance information to the server. This monitoring should also be customizable by the servers. Because this monitoring is essential for the server to maintain the global view of the region and make correct decisions. In Figure 1.1 the bidirectional arrows between the server and the routers reflect this requirement that the routers send their performance measurements to the server.

4) The router must support server probing. Figure 1.2 illustrates the concept of probing by the server. The aircraft represents a probe, which is a mobile agent carrying code (e.g., JAVA applet) that will collect information from a specific path of routers and bring the information back to the server. Such probing gives the server a fast way to verify the authenticity of the global view that it maintains, independent of router measurements.

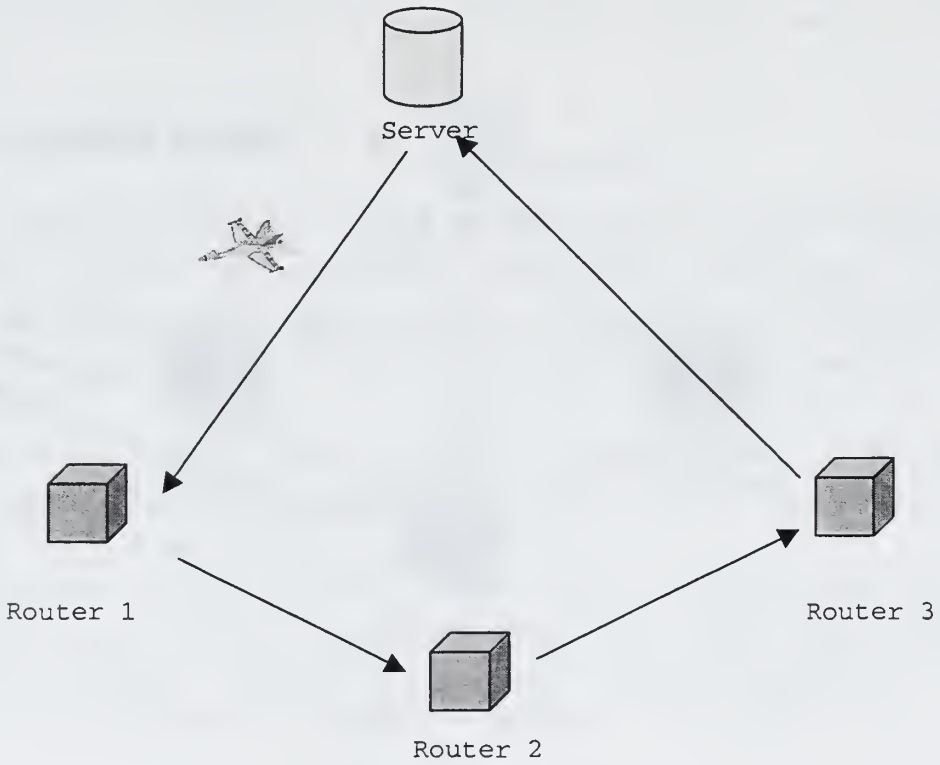


Figure 1.2 Server Probing of the Server.

C. BUILDING A LIGHTWEIGHT ROUTER

Active networking is a solution to the problem of automatically collecting pertinent network management information for a SAAM server.

Applications can inject customized programs into network nodes with Active Networking. Now the network nodes do not only forward what they receive, but they can also perform computations with them. Information injected in the network can be modified, stored or redirected while it is being forwarded. So we can say that Active Networking refers to the addition of user-controllable computing capabilities to the network.

Active Network can achieve its goal through the concept of self-directing units. This unit, as we will see in the next chapters, is a piece of code, which can be written in different languages or platforms. There are two advantages to this approach.

1) The server will be able to dispatch resident agents to perform necessary functions at the lightweight router. This meets requirements 1 through 3 stated in the previous section by sending necessary agents.

2) The server will be able to verify the information, which routers collect with server probing. The SAAM server can probe the routers at any time with the Active Networking approach. This meets requirement 4 stated in the previous section.

D. THESIS ORGANIZATION

In this thesis I will investigate how to prototype a lightweight router for SAAM using the Active Networking approach. This thesis is organized into the following chapters:

Chapter I: Introduction. This chapter provides an introduction to Server and Agent based Active Network Management (SAAM) and the Active Networking approach that is used to build a lightweight router for SAAM.

Chapter II: Background. This chapter provides a detailed explanation of the SAAM architecture and the Active Networking approach.

Chapter III: Design. This chapter explains the issues involved in designing an active router, such as choosing the correct hardware, O/S, and the execution environments. This chapter also explains how to connect a router to the Active Network Backbone (ABONE) testbed.

Chapter IV: Implementation. This chapter explains the steps that are taken to build an active router.

Chapter V: Verification. This chapter describes a preliminary evaluation of the active router. In particular, a server probing experiment is conducted using such routers.

Chapter VI: Conclusions. This chapter presents an overall assessment and lessons learned. It also contains suggestions for future work.

II. BACKGROUND

SAAM architecture, the hierarchical design of SAAM and the advantages of this architecture are explained in the first part of this chapter. Active Networking is explained in the second part.

A. SAAM ARHITECTURE

We envision SAAM to be the common platform where different network functions such as routing, resource reservation, network management, accounting, and security can be integrated. By concentrating network management and control among a smaller number of servers, SAAM can potentially be used for faster deployment of new services than is currently possible [Ref: 2].

SAAM deploys dedicated servers that perform decision-making tasks for the routers. This enables designing a lightweight router in terms of functionality requirements for a SAAM environment. A SAAM server has two major functions. First, a SAAM server maintains a global view (Path Information Base) of a network region. The helicopter example in the previous chapter explains this situation. SAAM servers can monitor traffic over long routes. Second, SAAM servers make decisions on behalf of the routers in the region. This is the first step for building our lightweight router. The server can send these decisions to routers, then the routers carry out the instructions.

SAAM architecture should be able to automatically and efficiently reconfigure the network before problems occur. We need this for the NGI. So it should be proactive which means the servers can detect and react to changing network conditions in a very short time, perhaps within fractions of a second. The term proactive can be explained with the example in Figure 2.1. Suppose we have a path from A to F, packets

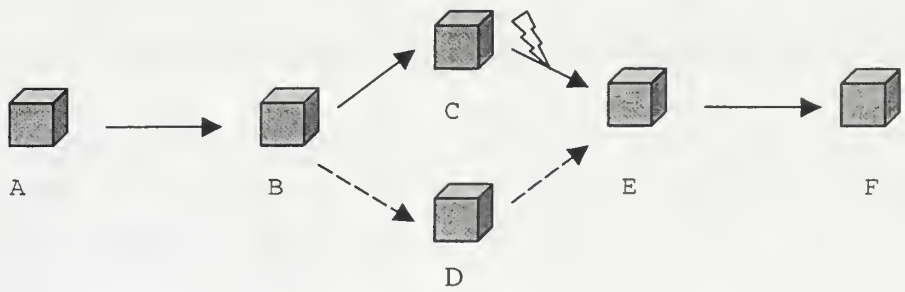


Figure 2.1. An Example.

follow the path A, B, C, E and F. But consider that a problem arises in the link between C and E. The architecture can provide a list of suitable candidates for a replacement sub-path to E. The packets can be directed to E via D.

SAAM organizes its servers in a hierarchy the same way as the Domain Name Service (DNS). Each server is responsible for only a small number of network nodes, which can either be routers or servers at lower levels. At the first level, SAAM partitions the network into autonomous regions, and sets up one server for each region. An example two-level sever hierarchy is illustrated in Figure 2.2. B1, B2 and B3 are first level servers, interacting with routers (C1-C8) directly. A1 is the second level server, it treats B1, B2 and B3 as routers and manages communications between them.

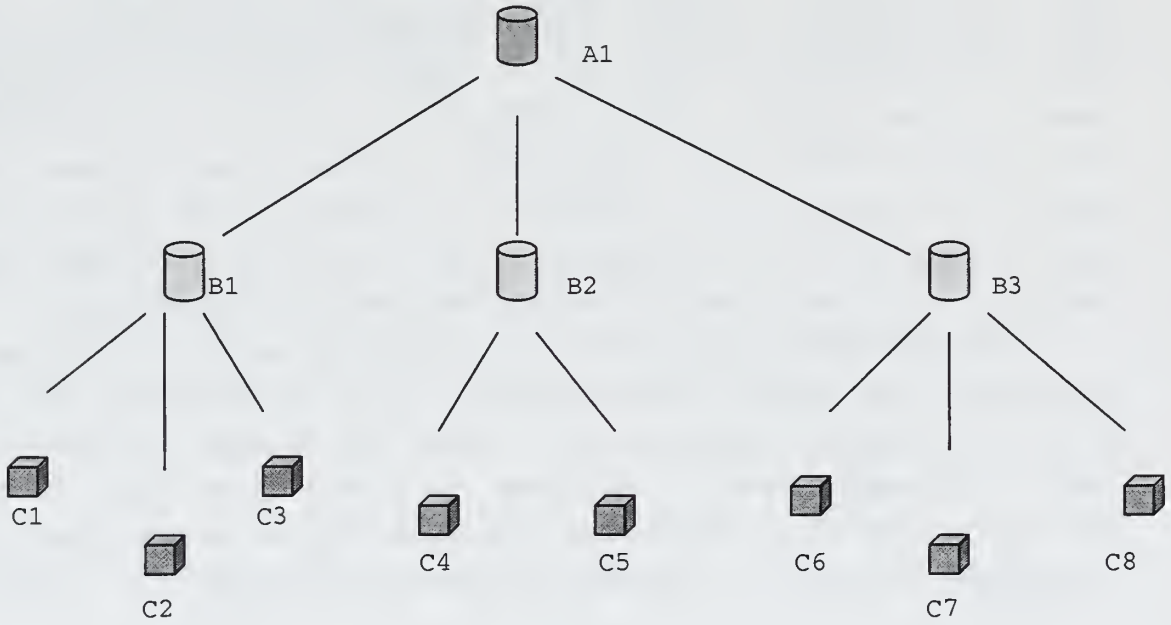


Figure 2.2. Hierarchical Organization of SAAM servers.

In addition to being scalable, another big advantage of the server hierarchy approach is that it makes it possible to gradually deploy SAAM into today's networks. Top level servers in the architecture exchange information for its region. A1 in Figure 2.2 speaks for its region with other top-level servers.

SAAM servers can provide new logical layers between the management station of the network and the routers. For example; B1 in Figure 2.2 will collect and maintain up-to-date path information for its region. B1 can get information from C1, C2 and C3. All servers at a given level will communicate with a parent server that can maintain performance information for paths crossing multiple regions. In Figure 2.2 A1 is the parent server that can maintain performance information for paths crossing B1, B2 and B3. So we can say that the SAAM servers can perform most management and control tasks in an automated and timely fashion.

B. ACTIVE NETWORKING

Active Networking means the addition of user-controllable computing capabilities to the network. The network is no longer viewed as a passive mover of bits, but rather as a more general computation engine. Active Networks allow individual users, or groups of users, to inject customized programs into the network. The information injected into the network can be modified, stored or redirected as it is being transported.

Active Networks are capable of producing a new networking platform, flexible and extensible at runtime to accommodate the rapid evolution and deployment of networking technologies.

Active networking is a solution to the problem of automatically collecting pertinent network management information for our network, which can also support NGI. This approach also satisfies the needs of the SAAM server. Because the SAAM server can query the routers in its region, this helps to verify the QoS information coming from each router. The SAAM server can also send commands, which the routers understand and accept. So the server can tell the router to update its routing table in the way it wants.

SAAM servers can obtain the information for a path by sending a probe ("active packet") through that path. The probe records its transfer delay and other performance statistics. An example is illustrated in Figure 2.3. S2 is a SAAM server. It sends a probe through the R1, R2, R3, R4, and R5 and records the transfer delays for the path between the routers R1 and R5.

One important advantage of this approach is that the routers can be programmable by the server. This means that processing is moved to the server. The processing is important for a lightweight router.

The Active Network consists of active nodes connected to each other. Every node has an operating system and one or more execution environments. The operating system at the node is responsible for allocating and scheduling the node's resources (like bandwidth, CPU cycles and storage). Each execution environment at the node implements a virtual machine that understands a particular type of active packets. Java's virtual machine is an example of an execution environment.

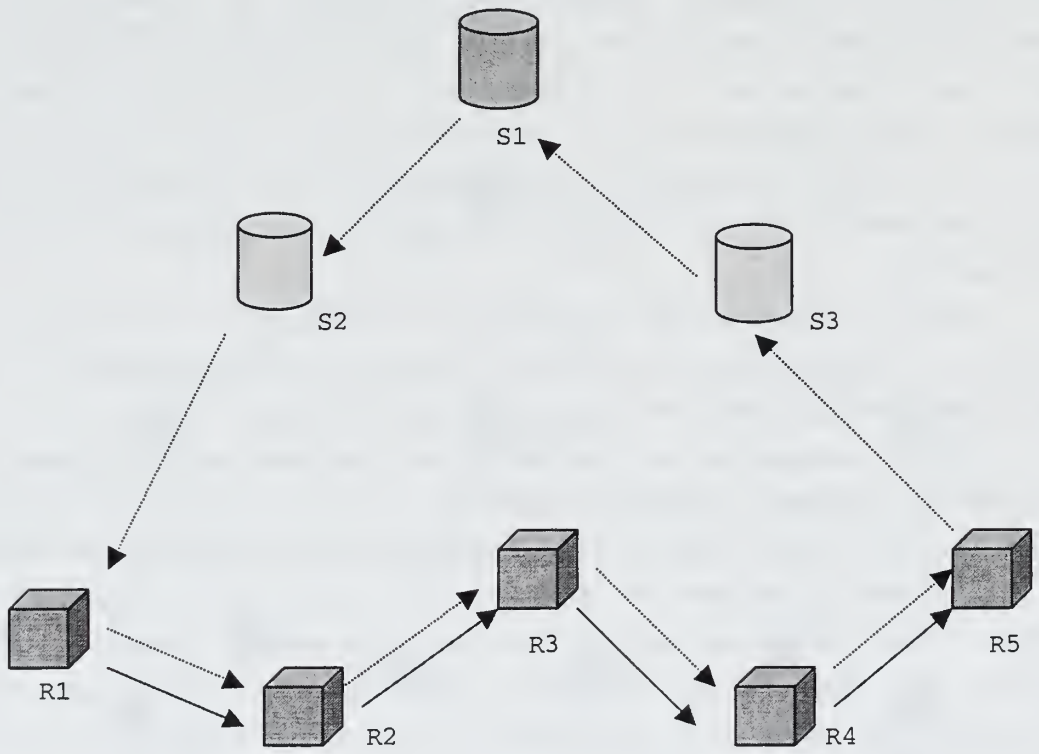


Figure 2.3. Server Probing.

The relation between an execution environment and the operating system is illustrated in Figure 2.4. The functionality of the node is divided between the Execution Environment and the Node Operating

system. Every Execution Environment exports an API or virtual machine that users can program by sending packets to it.

The node Operating System hides the details of resource management and isolates the behavior of Execution Environments from each other. Meanwhile the Execution Environment hides the details of interaction with the end user (through active packets) from the node Operating System.

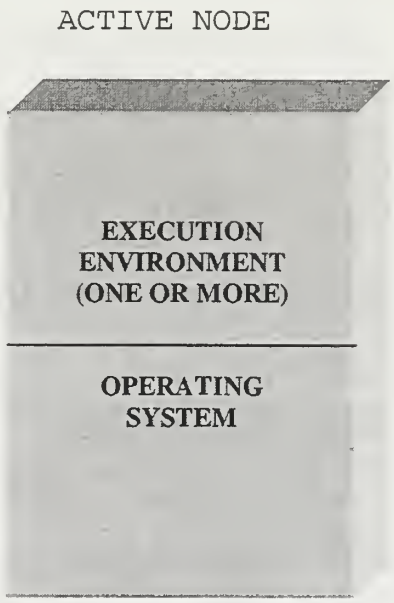


Figure 2.4. Active Node.

The flow of packets through an active node is shown in Figure 2.5 [Ref: 3]. The active node classifies the packets it receives according to their ANEP headers. Then these packets are placed in channels according to this classification.

In Figure 2.5 EE1 receives ANEP packets of a particular type. Each packet is encapsulated in a UDP datagram, which in turn is encapsulated in an IP packet. EE2 also receives UDP datagrams containing ANEP packets of a different type. EE3 receives TCP packets encapsulated in IP. As we see in the figure every packet matches an Execution Environment (EE). Each incoming packet should match at most one EE. Incoming packets that match no description are dropped. In the figure the IP packet is dropped because it does not match any EE. So no EE receives it.

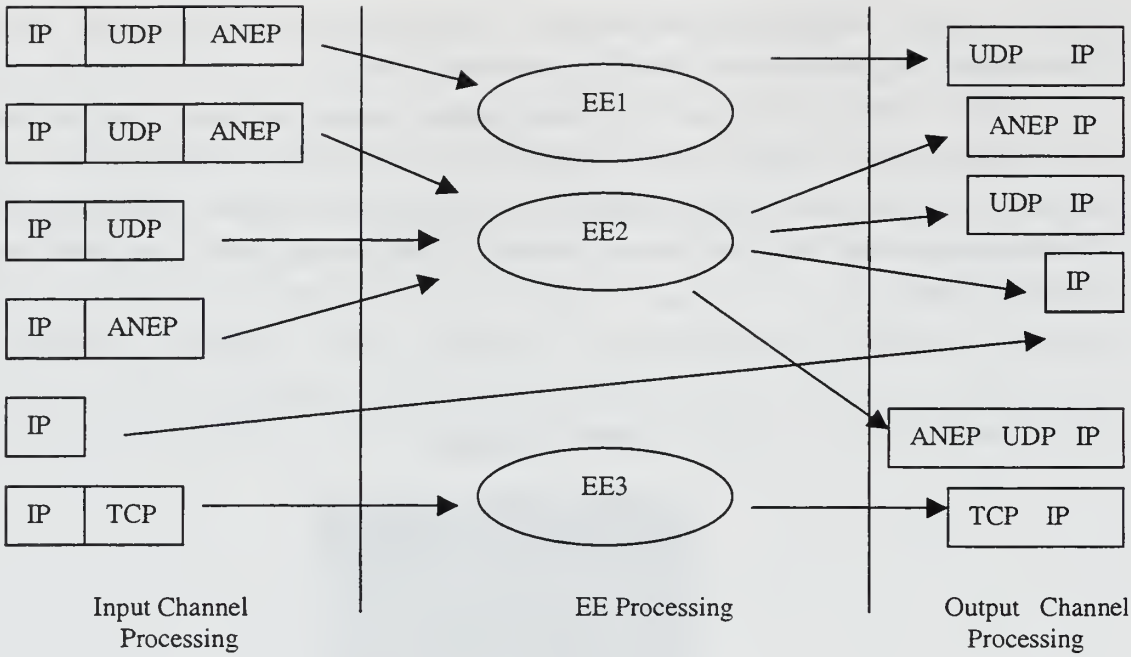


Figure 2.5. Active Node Processing.

Active Network Encapsulation Protocol (ANEP) allows users to send their packets to a particular Execution Environment. It specifies a mechanism for encapsulating Active Network frames for transmission over different media.

The programs sent to the active node are carried as the payload of an active network frame. The format of the ANEP header is shown in Figure 2.6. Execution environments are assigned a type identifier number in the packet header. So each packet can go to a different execution environment according to the type identifier field of the ANEP header. Detailed explanation of the ANEP fields can be found in [Ref: 4].

Version	**Flags**	**Type ID**
ANEP Header Length		**ANEP Packet Length**
Options		
Payload		

Figure 2.6. The Format of the ANEP Header.

III. DESIGN

Several important design decisions must be made when prototyping a lightweight active router. They include the choice of hardware and the operating system and the selection of an Active Networking platform and the associated execution environments. The issues surrounding these decisions are discussed in this chapter. Our design decisions are also presented, which include installation and operation details for each component that we have selected.

A. SELECTION OF HARDWARE AND OPERATING SYSTEM

It is very important to select the appropriate hardware and operating system for the lightweight router because these decisions affect every aspect of the router. The choice of hardware depends largely on budget considerations. Today a very powerful PC e.g., a 400Mhz Pentium II processor runs around $2500. Such a PC works well for prototyping a lightweight router as we have done. More expensive hardware will be required to support higher data transmission speeds in real networks.

The CPU, RAM and hard disk are the most critical components of the PC. For our prototype we used a DELL XPSR400 PC. The CPU is a 400Mhz Pentium II. There are 256 MB of RAM (100Mhz SDRAM). The Hard drive is an 8.4 GB EIDE Ultra ATA. The requirement of hard disk space depends on the set-up of the PC. If one operating system were to be installed, then 3GB harddrive would be enough. It should be noted that if you plan to install multiple operating systems you should check the hardware requirements for each of them. This is because each operating system may have special hardware requirements. You have to find the optimum configuration for your PC that will work with all target operating systems. We selected a relatively large (8 GB) hard drive because we wanted to try out more than one operating system.

The operating system at the router is also very important because as discussed in the previous chapters, the operating system determines which execution environments can be used. Specifically, the operating system is a middle layer between the execution environments and the hardware resources.

The DOS, NT and Linux operating systems are installed to our prototype router. We used a program called System Commander Deluxe to switch between them at boot time. System Commander is a software tool made by VCOMMUNICATIONS, Inc. It simplifies the task of maintaining multiple operating systems on a PC. Figure 3.1 shows the basic hard disk organization, independent of any operating system. The Master Boot Record is the first sector on the hard disk, controlling which operating system will be loaded at boot time. When System Commander is installed, it replaces the master boot record with its own master boot record to control the boot up process. System Commander ensures that different operating systems at the PC do not interfere with each other.

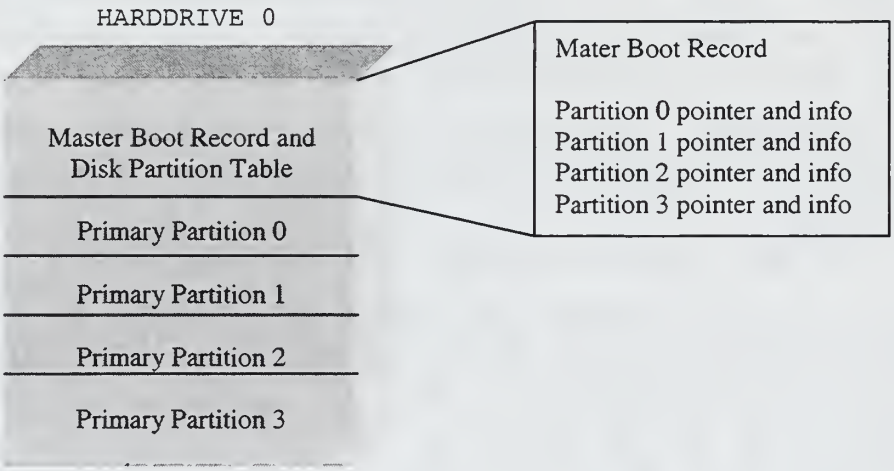


Figure 3.1. Basic Hard Disk Organization.

Choosing the proper operating system for the active node depends on the Active Networking Platform on which you will implement all execution environments. In our case the Active Networking Platform is Active Networks Daemon (ANETD v1.0) which will be explained in the next section in detail. One important thing that we should know about ANETD is that you can run it with Linux on X86, FreeBSD on X86 and Solaris on Sparc. So the main reason for our selecting Linux as our main test operating system is that we wanted to run ANETD.

B. ACTIVE NETWORKING PLATFORM

ANETD is the software that connects our prototype router to the Active Network Backbone (ABONE). The reason for choosing ANETD as the networking platform is that first it is needed to connect to ABONE second it allows the deployment of different execution environments to the routers. ANETD is an experimental daemon specifically designed to support the deployment, operation and control of active networks. ANETD performs two major functions [Ref: 5].

1) It allows the deployment, configuration and control of networking software (including experimental execution environments for active networking) into the network. ANETD allows routers to share the same software, and allows the deployment and control of distributed network services through a centralized source. Host1 in Figure 3.2 installs an EE from a central server to host2 and host3.

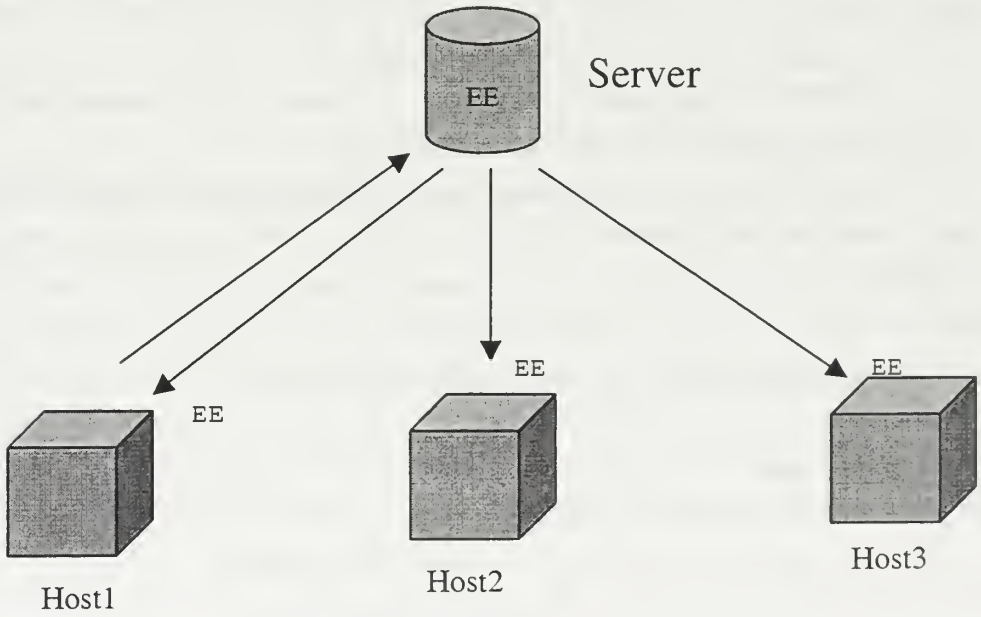


Figure 3.2. Deployment of EE from a centralized source.

2) It demultiplexes active packets (encapsulated using ANEP) to multiple execution environments located on the same network node

and sharing the same input port. ANETD listens to a unique user-assigned UDP port and accepts ANEP encapsulated packets. ANETD also checks the ANEP header of each packet to determine which EE it is for. An application at the router can receive traffic demultiplexed by ANETD coming from a port. It can be seen in Figure 3.3 that the incoming packets are demultiplexed by ANETD and they are sent to the appropriate execution environments.

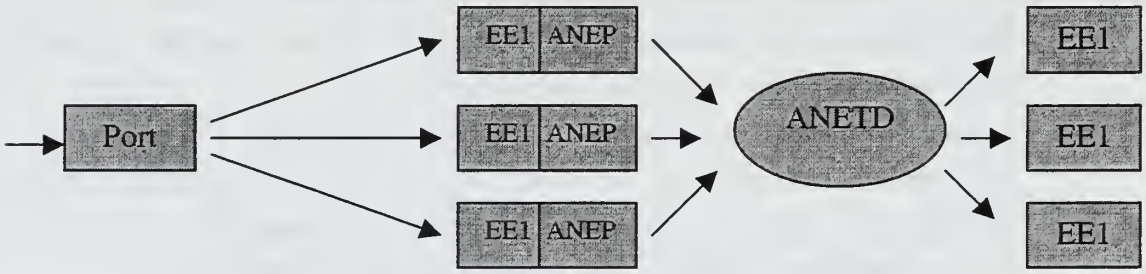


Figure 3.3 Demultiplexing of packets by ANETD.

ANETD is implemented in "C". It uses the standard UNIX API. So it should be portable to any Unix platform.

ANETD can be invoked as a user application and it does not need any runtime privileges [Ref: 5]. Typing the following command line can start it

ad.<ostype>[-p <ANEPport>] [-u <localportpoolstart>] [-s]

- <ostype>
  can have one of these values
  - solaris = SunOS 5.5.x running on Sparc
  - linux = Linux running on Intel x86
  - bsd44 = FreeBsd running on Intel x86
- <ANEPport> is the port on which ANETD listens and is also demultiplexed (Default is 3322).
- <localportpoolstart> is the starting port number for dynamically allocating local ports (Default is 8000).

- -s switch authorizes ANETD to send a small heart-beat UDP packet every 30 seconds to the main ANETD server. The current main ANETD server is sequoia.csl.sri.com.

For example, to start ANETD on a Linux system using ANEP port 3323 and allocating local ports starting at port 8010, one would enter the following command line:

```
ad.linux –p 3323 –u 8010
```

As another example, when the command below is executed, ANETD starts to listen on port 3322 and it uses 8000 for dynamically allocating local ports. It also sends a small UDP packet every 30 seconds to the main ANETD server.

```
ad.linux –p –u –s
```

Control commands [REF: 5] used in ANETD are explained in Appendix A. There are 6 commands, load, query, kill, get, put and conf. These commands allow a client to deploy, configure and manage network application software.

C. ACTIVE EXECUTION ENVIRONMENTS AND LANGUAGES

There are a few experimental execution environments available for download. Each of them has some claimed advantages over the others. The important thing is to choose the one that fits your needs the best. The execution environments ANTS, PLAN and Smart Packets are installed and tested on our prototype router. Programs written for these execution environments can be found easily. They are explained in the following sections.

1. ANTS

ANTS is a Java-based toolkit for constructing an active network and its applications. Software Devices and Systems Group Laboratory for Computer Science Massachusetts Institute of Technology developed it.

The latest version is 1.2.A. It can be downloaded at http://www.sds.lcs.mit.edu/activeware/ants/.

ANTS requires Java and Tcl to build and run. If you have Linux RedHat 5.1, you do not have to worry about installing Tcl, it is in the packages. ANTS is written entirely in JDK 1.0.2 compliant Java and can be run as a user level process with no special privileges. Before building from the ANTS distribution you must set your Java class path environment variable to include the home directory of the downloaded ANTS distribution. For example if that directory is /home/ants, then you must set,

```
setenv CLASSPATH $HOME/ants:$CLASSPATH
```

Then you type "make" at the top-level directory and ANTS will be fully built. The full distribution contains the following directories.

- ants, the implementation of the active node runtime.
- apps, network applications and their samples.
- docs, papers and javadoc generated APIs.
- runs, network configurations and result files.
- utils, general purpose utilities.

Classes of particular interest are: Node, Channel, UDPChannel, ConfigurationManager, Capsule, Protocol, Application.

The apps directory contains sample applications. This directory may also be used as a platform to develop new applications. There are a couple of useful techniques for debugging the applications you construct. One of them, the log method of the Node class causes a message to be echoed on standard error. Another one is the recompiling with the static field Entity.logging set to true, which causes status messages to be printed.

The docs directory contains two papers and the javadoc generated API reference pages. One paper describes the ANTS architecture. The other paper describes the general design and usage considerations for the ANTS programming model.

The runs directory contains the configuration files that describe network arrangements, route files that describe route tables, start scripts that launch a network experiment, log files that record the output of the node messages.

The utils directory contains several classes that do not depend on the ANTS architecture. MD5 class, which is related to security, is also found in this directory.

Next we describe an example ping program written to work with ANTS. The complete listing of the program is available in Appendix B. The program consists of three Java files: the PingCapsule class, the PingProtocol class and the PingAplication class. The ping program can be run with the script file ping.start, which can be found in the distribution. The script file to run this ping program can be found in the runs directory.

A configuration file is needed to start the ping program. The configuration file is shown in Figure 3.4 and the start file is shown in Figure 3.5. The hosts in this example are three ports in the same machine.

```
# simple ping configuration -
# - three nodes (source, router, destination)
# - duplex connected by udp
# - all running on the same machine
# - source pinging destination


node 18.31.12.1 -routes ping.routes -log 255
channel 18.31.12.1 melon.cs.nps.navy.mil:8001 -log 255
application 18.31.12.1 apps.PingApplication -target 18.31.12.3
manager 18.31.12.1 -gui true -log 255

node 18.31.12.2 -routes ping.routes -log 255
channel 18.31.12.2 melon.cs.nps.navy.mil:8002 -log 255
manager 18.31.12.2 -gui true -log 255

node 18.31.12.3 -routes ping.routes -log 255
channel 18.31.12.3 melon.cs.nps.navy.mil:8003 -log 255
manager 18.31.12.3 -gui true -log 255

connect 18.31.12.1 18.31.12.2
connect 18.31.12.2 18.31.12.3
```

Figure 3.4 Ping Configuration File.

The ping.route file is shown in Figure 3.6. This file in fact is created from the configuration file in Figure 3.4. The contents of these files will be explained in the next chapter. The ping program can

also be run in an existing network. There are also script files for this purpose in the distribution of ANTS. These files are called join files.

```
java ants.ConfigurationManager ping.config 18.31.12.3 >&
18.31.12.3.log &
java ants.ConfigurationManager ping.config 18.31.12.2 >&
18.31.12.2.log &
java ants.ConfigurationManager ping.config 18.31.12.1
kill %?18.31.12.3
kill %?18.31.12.2
```

Figure 3.5 Ping Start File.

```
# shortest routes, automatically generated

# source    destination next        addr

18.31.12.1  18.31.12.2   18.31.12.2    melon.cs.nps.navy.mil:8002
18.31.12.1  18.31.12.3   18.31.12.2    melon.cs.nps.navy.mil:8002
18.31.12.2  18.31.12.1   18.31.12.1    melon.cs.nps.navy.mil:8001
18.31.12.2  18.31.12.3   18.31.12.3    melon.cs.nps.navy.mil:8003
18.31.12.3  18.31.12.1   18.31.12.2    melon.cs.nps.navy.mil:8002
18.31.12.3  18.31.12.2   18.31.12.2    melon.cs.nps.navy.mil:8002
```

Figure 3.6 Ping.Routes File.

2. PLAN

a. Installation

Plan (Programming Language for Active Networks) is a new language for programs that are carried in the packets of a programmable network. Plan is a strict functional language providing a limited set of primitives and datatypes.

You can install plan software in two ways. You can obtain the class files and execute them directly or you can obtain the entire source and build it yourself. Both distributions contain all of the documents and the sample programs.

Three packages are needed for source installation.

-JDK 1.1.X: Java Development Kit

You can get this at

http://www.blackdown.org/java-linux/Mirrors.cgi/.

-Pizza: A Substantial Companion to Java, version 0.39

You can get this at

http://www.cis.unisa.edu.au/~pizza/.

-JavaCC: The Java Compiler Compiler, version 0.6.1

You can get this at

http://www.suntest.com/JavaCC/index.html

These three packages should be installed first. Next plan source can be unpacked. All these operations should be executed in the directory that you would like the source to be unpacked.

If you are not interested in acquiring the source code, you can get the classfiles and install them. For classfile installation you need the pizza distribution, but you do not need to install JavaCC. Both source installation and classfile installation are explained in Appendix C in detail.

Plan is implemented in Java, API 1.1, and is composed of a number of packages with names in the form of PLAN.*. Basic knowledge of compiling and running Java files is needed for running plan programs. The examples below are executed from the plan directory created in accordance with Appendix C. If you like to execute the code from some other directory, simply prepend the class names given to the Java command with plan. For example you would type in the command lines

```
java PLAN.ARMain -?
```

instead of

```
java ARMain -?
```

This is useful if you install plan in some other machine. You have to set your classpath in order to run plan. If plan is installed

home/nkaplan/planc/PLAN

you would update your classpath to

home/nkaplan/planc

The full distribution contains anep, log and plan directories shown in Figure 3.7.

Figure 3.7 Plan Distribution.

The anep directory has the ANEP distribution. Recall that ANEP (Active Network Encapsulation Protocol) specifies a mechanism for encapsulating Active Network frames for transmission over different media types. The contents of the anep directory are shown in Figure 3.8.

The log directory has the log file. This file records the output when the plan programs are run so this is very useful for tracing the programs. The log file is shown in Appendix D.

The plan directory contents are shown in Figure 3.9. There are the Anon, basis, docs, fixedroute, install, interp-tests,

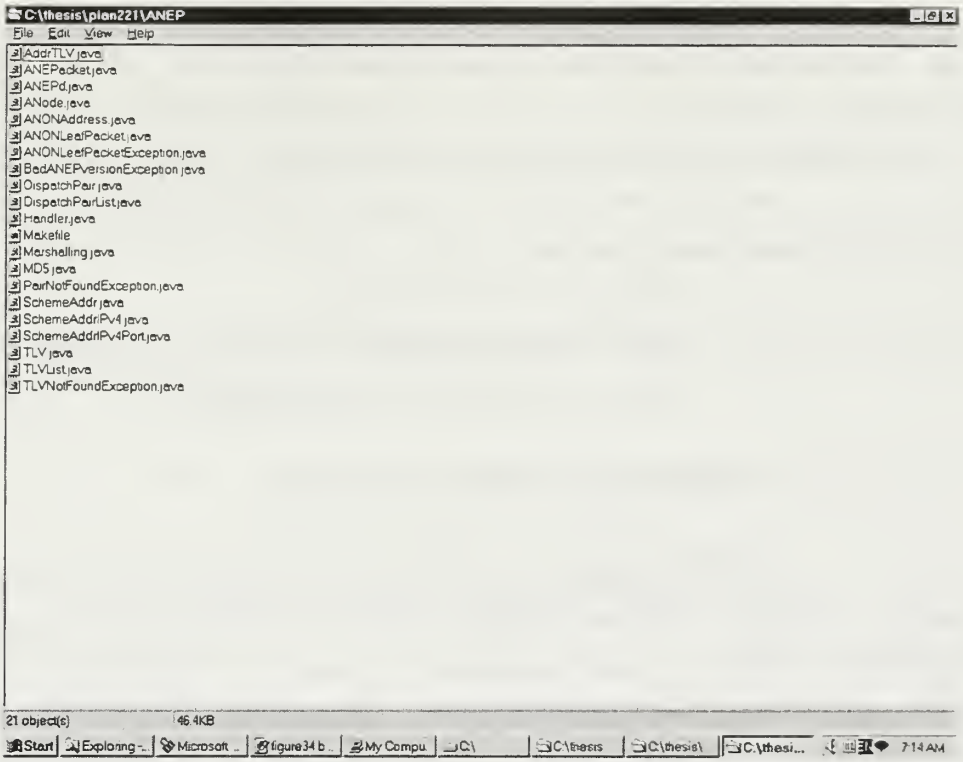


Figure 3.8 Anep Contents.

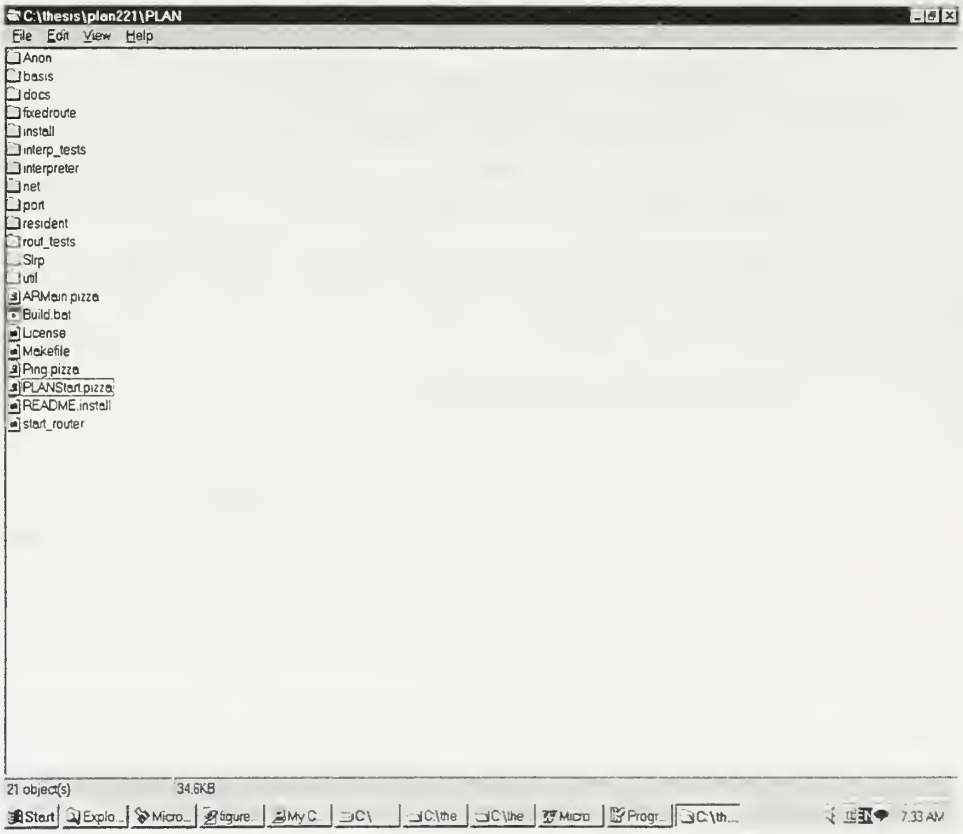


Figure 3.9 Plan Contents.

interpreter, net, port, resident, rout-tests, slrp and util directories and the other important files. One file that should be noted is theARMain.pizza file. This file contains the ARMain class that serves as the entry point for the router. This file is shown in Appendix D.

You can use the plan distribution in a number of ways. There are three ways you can

- set up an Active Router
- send a plan program to an Active Router
- run a non-network plan interpreter

b. Starting up PLAN aware Router

A local machine is set up as an active router first. Active router in this content means that the router is ready for PLAN programs. ANETD is not necessarily used to install PLAN execution environment from some other place. The ARMain class is the entry point for the active router. When %java ARMain -? is typed, the usage of ARMain can be seen.

```
Java ARMain [i] [-v|-d|-q] [-l logfile] [-ip port]
        [[-m master] [-h hub] | [-rf routtab file]]
        [-n n1,n2,........, nk]
```

An example is described below, which shows to start an Active Router in one machine.

```
%java ARMain -v -ip 7081

24-Nov-98 4:16:18 PM: ARMain: verbose mode on.
24-Nov-98 4:16:18 PM: ActiveRouter.start: Active router
up!
24-Nov-98 4:16:18 PM: ActiveRouter: OUT to Ipv4UDP
:(m,7081) :fstCookie
24-Nov-98 4:16:18 PM: ActiveRouter: OUT: succeeded
24-Nov-98 4:16:18 PM: SLRPmaster:received a request to
add: Ipv4UDP : (m, 7081)
24-Nov-98 4:16:19 PM: ActiveRouter: OUT to Ipv4UDP
(m, 7081)
```

```
24-Nov-98 4:16:18 PM: ActiveRouter: OUT : succeeded
24-Nov-98 4:16:18 PM: ActiveRouter: IN from Ipv4UDP :
(m, 7081):fstCookie
24-Nov-98 4:16:18 PM: ActiveRouter: IN from Ipv4UDP :
(m, 7081):newRT
24-Nov-98 4:16:18 PM: SLRP: received new route table.
```

Now we can start another active router by opening another terminal window and typing:

```
%java ARMain -ip 7082 -m melon.cs.nps.navy.mil:7082 -v

24-Nov-98 4:17:11 PM: ARMain: verbose mode on.
24-Nov-98 4:17:12 PM: ActiveRouter.start: Active
router up!
24-Nov-98 4:17:12 PM: ActiveRouter: OUT to Ipv4UDP
:(m,7081) : addme
24-Nov-98 4:17:12 PM: ActiveRouter: OUT: succeeded
24-Nov-98 4:17:12 PM: ActiveRouter: IN from Ipv4UDP :
(m, 7081):fstCookie
24-Nov-98 4:17:12 PM: ActiveRouter: IN from Ipv4UDP :
(m, 7081):newRT
24-Nov-98 4:17:12 PM: SLRP: received new route table.
```

A plan program is a list of definitions, which could be function definitions, exception declarations or variable bindings. There is just one function in the Hello World program in Figure 3.10.

```
fun   doit ( ): unit =
   (print (thisHost ( ));  print("says : Hello   World!\n"))
```

Figure 3.10 Hello World Program.

We should learn how to use PLANStart tool before we run the Hello World program. The PLANStart tool provides us to inject the plan programs into the network directly from the command line. Typing java PLANStart gives us the usage of the PLANStart tool.

%java PLANStart

java PLANStart [-v] [-p port]<code><RB><router Ipv4 2address>

-v	: produces a verbose output.
-p	: allows you to specify the outgoing TCP port.
-code	: is the name of the file that contains the plan program.
-RB	: specifies the initial amount of resource to hand the packet going into the network.
- router Ipv4 2adress	: specifies the host that is to be the entry point into the network.

Table 3.1 Arguments of the PLANStart.

We can now run the Hello World program. Suppose we have written the program in a file called HelloWorld.plan. We can start the program by typing,

java PLANStart HelloWorld.plan 10 melon.cs.nps.navy.mil

Now the program waits for us to give the next input. The next input is the initial invocation. Plan is a list of definitions so when the program arrives at an active host, we must specify which function to start executing and with what arguments. In the HelloWorld program there is only one function and it takes no argument, so we type:

doit ( )

Then the response comes :

IPv4 : melon.cs.nps.navy.mil/131.120.1.244 : HelloWorld!

We can now run more useful programs. The second example is a simple ping program, which uses the active network. The ping program tries to reach a remote host. When it reaches the host, it tries again to reach the original host. And finally it prints "Success" when it returns to the original host. The ping program is shown in Figure 3.11.

```
fun ping ( source : host,destination : host,outgoing : bool) : unit =
     if outgoing and (thisHost ( ) = destination) then
           OnRemote ( ping (destination, source, false),
                        source, getRB ( ), defaultRoute)
     else
           If not outgoing and (thisHost ( ) = destination ) then
                 print ( "Success" )
           else  OnRemote (ping (source, destination, outgoing),
                        destination, getRB ( ), defaultRoute)
```

Figure 3.11 Ping Program.

We can use the same active network that we set up for the HelloWorld program. Suppose we want to ping melon.cs.nps.navy.mil:7082 from melon.cs.nps.navy.mil:7081. The ping program can be saved as the Ping.plan. To run this program, we would type:

%java PLANStart ping.plan 60 melon.cs.nps.navy.mil:7081

The initial invocation would be

```
ping(getHostByName("melon.cs.nps.navy.mil:7081"),
    getHostByName ("melon.cs.nps.navy.mil:7082),
    true)
```

The initial invocation here is different from the one used for the HelloWorld program. The invocation starts with ping stating that it is the function call. The first two arguments in the invocation are the function calls, which are evaluated locally before being sent.

Let us look at the details of the plan program how it works. Plan programs are injected by a host into the active network. The situation is illustrated in Figure 3.12.

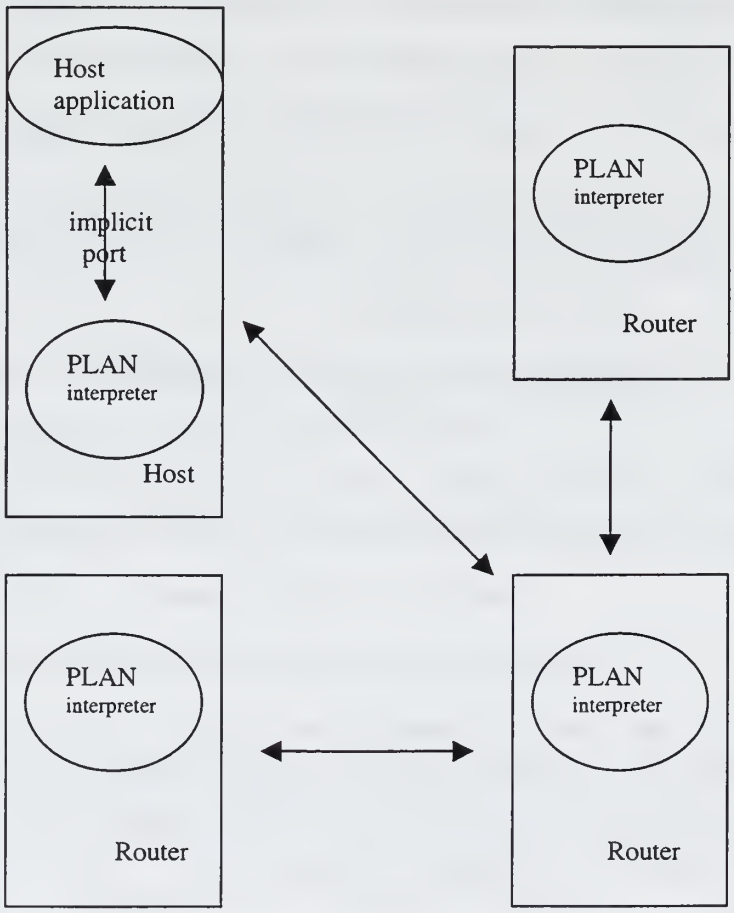


Figure 3.12 Plan Network Environment.

The host application opens a connection to the local plan interpreter. The application sends the plan packet via this connection. The plan packet contains the plan code to be executed. In our first example it is the HelloWorld program, which will be placed in the packet. The connection created also serves to pass the output from the plan program back to the host. When a plan packet is injected into the active network, it is recognized by its port number by the other routers. By this way the output can be sent to the same port of that plan program.

3. Smart Packets

Smart packets is a DARPA-funded Active Networks project. They are written in a tightly encoded, safe language specifically designed to support Active Networks. Smart Packets improve the management of large complex networks by [Ref: 6]

- moving management decision points closer to the node being managed
- targeting specific aspects of the node for information rather than scatter-shot collection
- abstracting the management concepts to language constructs, allowing nimble network control.

The programs can be injected into the network with smart packets. The programs are capable of performing computations and manipulations on behalf of the user. These programs increase user and application control over the network.

Smart packet format is shown in Figure 3.13. Basically smart packet header has four fields: version number, type, context and sequence number. A common smart packet header is encapsulated within ANEP, which will be explained in the next chapter. ANEP Daemon is the injection and reception point for smart packets. It also contains the virtual machine for executing the programs received. The daemon injects the smart packet into the network. The smart packet can be sent to either an end host or to each router in a hop-by-hop manner.

a. Programming Languages used in Smart Packets

There are two programming languages used in smart packets. They are Sprocket and Spanner. Sprocket is a high level language like C. It is based on C's grammar and keywords. C++ style comments can also be used in the sprocket code. Primitive types like int, short and char, which are used in C, are replaced with types that show the type size and sign. There are also built-in array, string and list types. An

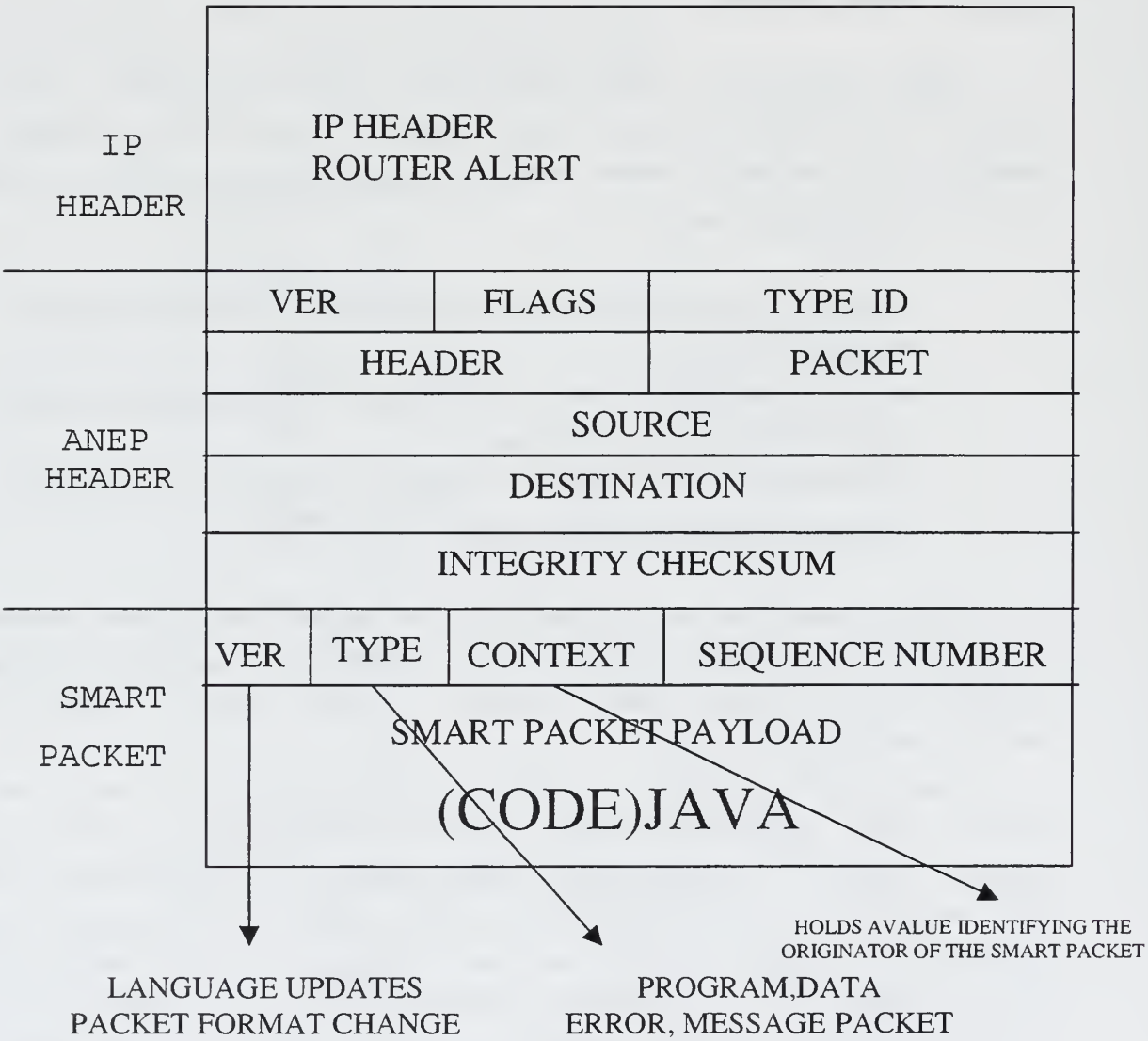


Figure 3.13 Smart Packet Format.

example Sprocket program is shown in Figure 3.14. This program gets the number of interface devices, the addresses associated with those devices and the Maximum Transmission Units for those devices. This information then placed into a packet and sent back to the originating host.

```
Main ( ) {
  Array of address addr;
  Packet pkt;
  unsigned8  num_interfaces  =  num_ifaces ( );
  pkt.data_append ( num _interfaces);
  unsigned8 index;
  for (index = 1; index <= num_interfaces; i++) {
    addr.set_size_of_dimensions(num_adresses(index));
    get_addresses(addr);
    pkt.data_append(index);
    pkt.data_append(addr);
    pkt.data_append(get_iface_mtu(index));

  }
  pkt.send( );
}
```

Figure 3.14 Example Sprocket Program.

Spanner is similar to assembly language. Spanner is designed especially to yield very small-encoded programs. There are differences in Spanner from assembly language. One difference is that there are declared variables in Spanner. Another difference is that Spanner has no access to memory for this reason the storage is done either on the stack or in variables. There are also branch operations in Spanner. A simple Spanner example is shown in Figure 3.15. This is equivalent to the Sprocket program shown in Figure 3.14.

b. Installation of Smart Packets

The last version of smart packets, which is 1.0.1, can be obtained at http://www.net-tech.bbn.com/smtpkts/smtpkts-index.html. The

file spkt-1.0.1.tart.gz should be installed. To install it, first unpack it; by typing

```
gunzip spkt-1.0.1.tart.gz
tar spkt-1.0.1.tart
```

The complete distribution includes Freebsd-2.2.6, anepd, bardemo, data, doc, injector, libsrc, scripts, spanner, sprocket, tools and vm directories. The installation instructions of smart packets are in Appendix E.

```
        decl-addr-arr-np %addresses;
        decl-pkt  %pkt;
        niface;
        papp @  &;
        decl-u8  %index #1;
$ loop:lt & &;
        brt  $done;
        papp  %pkt &;
        naddr &;
        sdim %addresses @;
        gaddr & &;
        papp  %pkt  @;
        mtu  &;
        papp  %pkt  @;
        ainc-np  &;
        bru  $loop;
$done :send %pkt;
        cont;
```

Figure 3.15 Example Spanner Program.

IV. IMPLEMENTATION

The detail of our implementation of a lightweight active router is presented in this chapter. There are four major steps to build the router and connect it to the ABONE: (1) choose the hardware, (2) select and install the operating system, (3) install ANETD, and (4) connect to the ABONE. The first 3 steps have been explained in the previous chapter. Step 4 is explained in this chapter. There are basically three phases in step 4. They are user registration, download and installation of ANETD, and node registration. These phases are explained in section A. Then in Section B two examples will be given to illustrate how to download an EE to the router and how to build a small active networking testbed within the ABONE.

A. CONNECTING TO ACTIVE NETWORK BACKBONE (ABONE)

1. Register as a user

The ABONE is an experimental network consisting of active routers built by different institutions. New ideas related to Active Networking can be tested on this network. So being a part of the ABONE is very helpful when researching Active Networking techniques. The ABONE is open to everyone who is interested in Active Networking. The main Web site about ABONE is located at http://www.csl.sri.com/ancors/abone/.

A user must register to be part of the ABONE. Specifically, he or she must supply information described in Table 4.1. The information entered is made available to the other nodes connected to the ABONE. Registered users can be seen in Appendix F. One of the important items that you enter is a public key. You can get the key generation program at the same site http://www.csl.sri.com/ancors/abone/. There are three key generation programs written for FreeBSD on X86, Linux on X86 and Solaris on Sparc. You can get the appropriate one for your machine. After getting your public key you can register as a user to the abone.

There are some rules that abone users should obey.

- Hosts should have a decent connection to the Internet.
- Hosts should not periodically shut down or loose

connectivity.

- Hosts should be reachable through non-standard UDP ports.

The last bullet above is important. Because there may be a firewall in your organization and you must open some ports publicly to

User Registry Information
Name: (Name of the administrator)
Password: (Needed to be able to modify the user information later)
Reenter password:
IP Address (aa.bb.cc.dd): (Only this machine would be authorized to administer the ABONE nodes)
Public Key: (Public Key of the administrator)
Organization:
Address of Organization:
Phone Number:
Email:
Purpose of registering with ABone:

Table 4.1 Registering Table.

the ABONE. For example, I requested the security officer to open some ports as UDP to the public for our computer melon.cs.nps.navy.mil. It is advised for you to get all the port numbers before trying to open them in your organization.

2. Invoke ANETD

The ANETD can be invoked after user registration is completed. The following command line should be used:

ad.linux –p 3323 –u 8010 –s

Once started, ANETD interacts with the main server, which is sequoia.csl.sri.com. It installs two important files in the directory where ANETD was started. The first file is shown in Table 4.2. This file includes pointers to the code servers in the ABONE. The code servers are sequoia.csl.sri.com and bro.isi.edu. The second file is shown in Table 4.3. This file basically includes the IP addresses of all the nodes in the ABONE and their public keys.

```
IPAddress: sequoia.csl.sri.com
Port: 7000
Organization: SRI International
Organization_Address: 333 Ravenswood Ave. Menlo Park, CA
95024
Name: Livio Ricciulli
Phone_Number: 650-8592969
Email: livio@csl.sri.com


IPAddress: bro.isi.edu
Port: 80
Organization: USC/ISI
Organization_Address: 4676 Admiralty Way, Marina del Rey,
CA 90292
Name: Jeff Kann
Phone_Number: 310-822-1511
Email: kann@isi.ed
```

Table 4.2 Code Servers in the Abone.

d00.csl.sri.com
udoN0w7B0K65hhwpwgpOzp/Pj//aYUTfEo2N4s8bW2kjs3rVtbfohk0UJA6cvcbLXZO
fGJybjgnaIj6G2NPtvQEAAQ==
capri.sri.brainstorm.net
umVy3uv1LpaSx7W83haqRoNVrEH/cNatGaa7B4YgQYRl4K9qPzrBpdoVk7rKVTDyX+O
pUc2B6aNepLsyD/FjWQEAAQ==
zaria.csl.sri.com
r8K+gZ4ZRo5usA6751RD5Kh7HNwGAu10HvuxGE5v5epOFcW0EgkLcTsBq7fjDWkW6EU
57oNG6F6e451rUln2oQEAAQ==
130.107.16.135
xc5Z6TVuR26Y7HtiAQ3Y7hXjzZwss6g+z2KignNBr1Da2YWo3KFVFsgUwfksUlPH27I
DLeU7ioqV5wmNhfwJ8QEAAQ==
138.100.10.152
xcNrTqbIK3oPl/k9ovUeWhfoNatzT5t7g0lhDguI+VC9Ef0GUHDwUtym3LqYCZudPlg
NrRtxTGMwSoHyJzJ8wwEAAQ==
128.9.160.165
sPEY1JLWL+Ovft6+JEPbi7tfSnX3PfCI68PqAKwiWrGWfqKl8UsZHRD1N4G194fUa5/
tXH1lW2HM7Pv93zEoRwEAAQ==
128.9.160.194
zQYprXgdnYjqkehgKqAh6RgEOOM15PJjp/EDCq/kBzl3F+B5lLhXmpfoox4SKukyBNP
6sv4PdPiLqIEy4xPkLQEAAQ==
207.3.230.162
pOP54oiR+Wvi/iKQzcAfxy2kazJWYFdAOkUx96WDmS3trnaPMlrn8GkYBDwSR8DX8Yh
JdNMTDzVzbG/gGUXurQEAAQ==
155.99.212.119
ysmUPZAyvTZOC8EygGpv5jQqVWth644B3bG+zhQXnDYuXk2dT2kOnZauqyNVJjeupOF
aGeYl5IeUnNi/ClzpSQEAAQ==
129.55.10.190
+KHXfCL/OdjIcbNoOVh3mUY8vD1XXpO3kronjWwKUZu/vJh1N+vO3Unpn/mYcOO8sMU
E417dlMGakWdZrSlLeQEAAQ==
199.171.39.3
5hTIUzwLCD3WtRK1flkUZssrTvfcW99oKPFmv9+CkDhcAZUjPAk+UIxiHgOCe9/2moY
Q5foAhSGkXeFnl12zCQEAAQ==
166.104.36.173
4jhjie
166.104.45.177
m/aOb8BxyxbiknsOAjEMKHXlOdhWr4c39FDDrzXDyiOL7wqpZiXDKZNEUCfViMvvBmO
0OQCLDcrRv4gldXRnEwEAAQ==
166.104.45.194
rZLD5aZb+8iyy7iGtRQVrugl3diOn42lPOqMwuTeQ09t05MsulQzk17c58MKWHRyKFz
eJOyCCd6kOuVSXeboQwEAAQ==
128.174.240.14
mQCNAy8b62UAAAEEAMbYY9kAyOHAFb9cblO7QiACmFdvcy3WjNZNc/mRrk9Qcp0v
158.130.12.150
1U0URh6BazDPNk2A6wMJ783IPE7rlGAyx0ARW3PKKo4rJy5Z3ppMg4rC/Um6YN9qw9f
m7JPLQo9LGMEywM+QawEAAQ==
131.120.1.244
t2pdU7tWzikq3cklFxyYpe6xDLqd4mdTiDp7cpIjGGnGHlsc+w3miVKE88rfpa39DIm
eIELTDHWFfIxVg0BOrQEAAQ==

Table 4.3 Hosts.allow File.

The machine is now connected to the ABONE with the invocation of ANETD. When ANETD is invoked with the command ad.linux -p 3323 -u 8010 -s, -s option makes ANETD to send a small UDP packet every 30 seconds to the main ANETD server. This is good because the server can now make some measurements about the nodes with UDP packets.

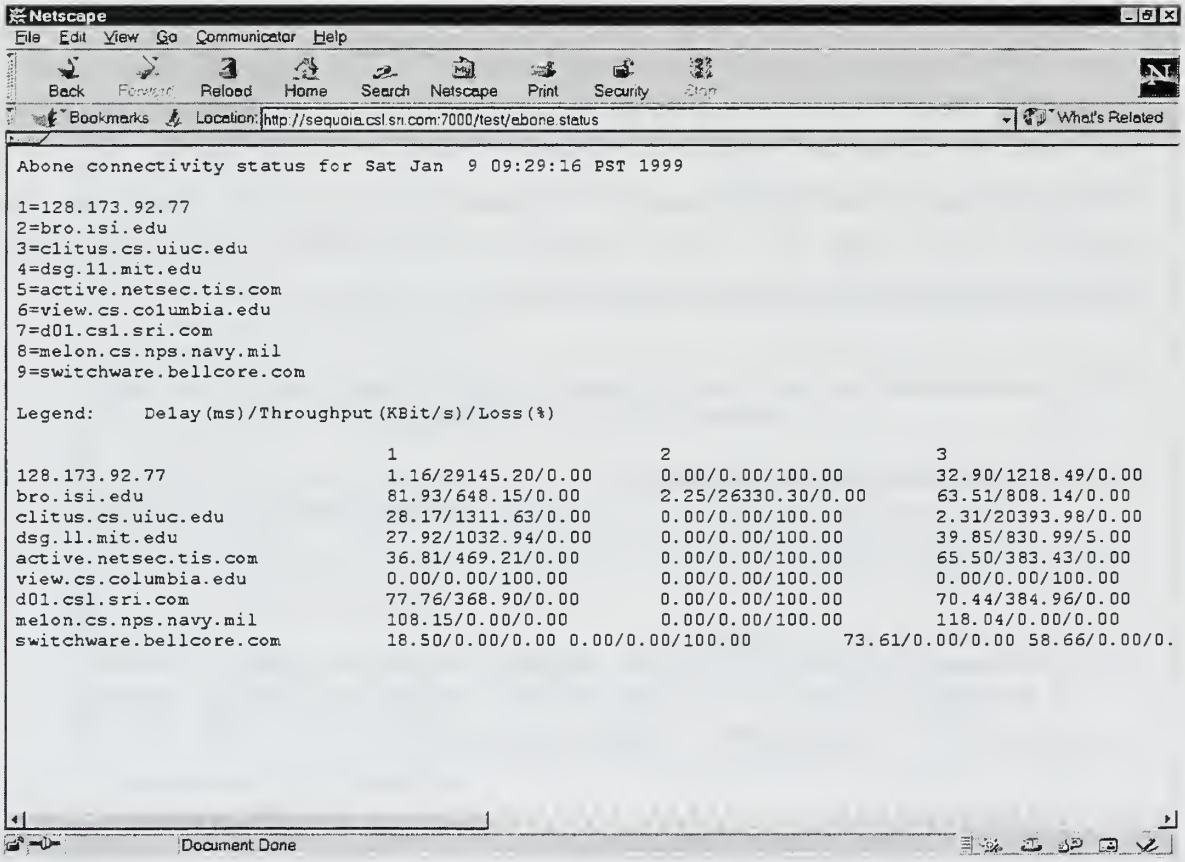
3. Registering the node

The next step is to register the node to be a part of the ABONE. The node registration table is shown in Table 4.4. A list of all nodes registered for the ABONE is maintained at [Ref: 7]. You can also register your node as a code server of the abone. The code server registry entry can be found at [Ref: 8].

Node Registry Information
IP Address (aa.bb.cc.dd): (This machine would run the ancors daemon and participate in ABONE activities)
Listening port number: Please use port 3322 as default. If you want to run multiple virtual networks on the same host, repeat this registration process changing port number each time.
Organization:
Address of Organization:
Name (point of contact):
Phone Number (for the point of contact):
Email (of the point of contact):

Table 4.4 Node Registry Table.

Our node melon.cs.nps.navy.mil is part of the abone at the moment. Active Networking experiments can be tested in this network now. The examples about active networking will be shown in the next chapter. The text output of the current abone status is shown in Figure 4.1. Melon.cs.nps.navy.mil is listed as the eighth node in the figure.




```
Abone connectivity status for Sat Jan  9 09:29:16 PST 1999

1=128.173.92.77
2=bro.isi.edu
3=clitus.cs.uiuc.edu
4=dsg.11.mit.edu
5=active.netsec.tis.com
6=view.cs.columbia.edu
7=d01.csl.sri.com
8=melon.cs.nps.navy.mil
9=switchware.bellcore.com

Legend:     Delay(ms)/Throughput(KBit/s)/Loss(%)

                                1                    2                    3
128.173.92.77                   1.16/29145.20/0.00   0.00/0.00/100.00     32.90/1218.49/0.00
bro.isi.edu                     81.93/648.15/0.00    2.25/26330.30/0.00   63.51/808.14/0.00
clitus.cs.uiuc.edu              28.17/1311.63/0.00   0.00/0.00/100.00     2.31/20393.98/0.00
dsg.11.mit.edu                  27.92/1032.94/0.00   0.00/0.00/100.00     39.85/830.99/5.00
active.netsec.tis.com           36.81/469.21/0.00    0.00/0.00/100.00     65.50/383.43/0.00
view.cs.columbia.edu            0.00/0.00/100.00     0.00/0.00/100.00     0.00/0.00/100.00
d01.csl.sri.com                 77.76/368.90/0.00    0.00/0.00/100.00     70.44/384.96/0.00
melon.cs.nps.navy.mil           108.15/0.00/0.00     0.00/0.00/100.00     118.04/0.00/0.00
switchware.bellcore.com         18.50/0.00/0.00 0.00/0.00/100.00     73.61/0.00/0.00 58.66/0.00/0.
```

Figure 4.1 Abone Status.

Information about the nodes can also be viewed graphically. The transfer latency information is shown in Figure 4.2. The packet loss rate information is shown in Figure 4.3. You can see the current situation of the nodes at anytime at this web site [http://sequoia.csl.sri.com:7000/java/abonestat.html].

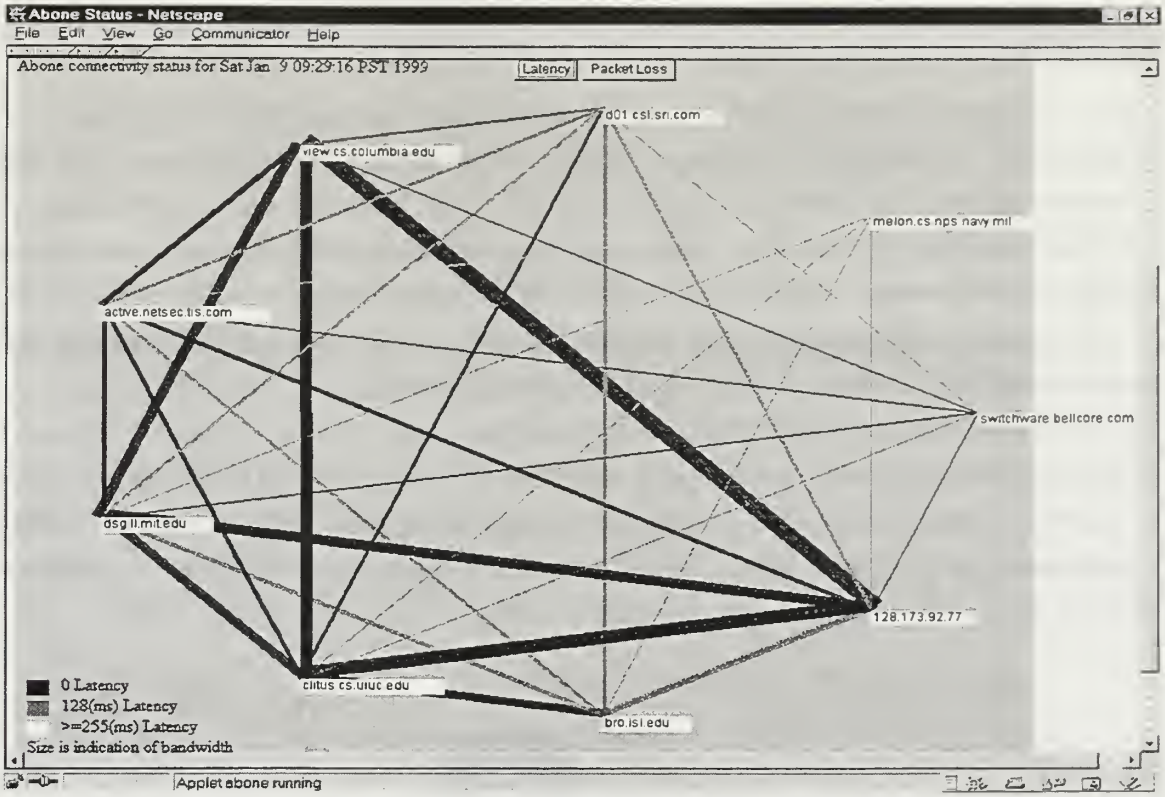


Figure 4.2 Latency Status of Abone.

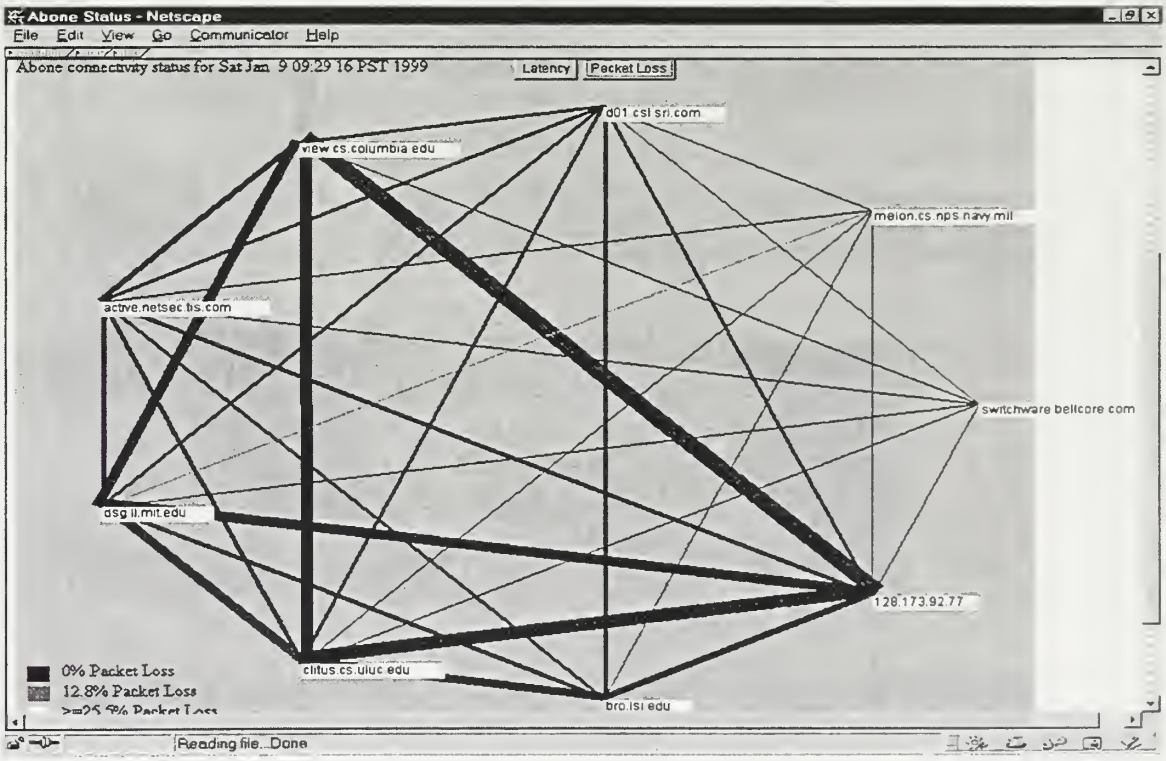


Figure 4.3 Packet Loss Status of Abone.

B. EXAMPLES

These examples show how to install an execution environment from the main server to different hosts. This main server is the main ANETD code server, which is sequoia.csl.sri.com. There are two examples in this section. First one shows how to install ANTS execution environment and the second one shows the PLAN execution environment. These execution environments are installed to hosts by using ANETD, so ANETD commands and files will be used in the example.

The situation is shown in Figure 4.4. Melon.cs.nps.navy.mil is part of the ABONE; we showed how to become part of the ABONE in the previous sections. Our node now can install an execution environment and send a program, which is written for that execution environment.

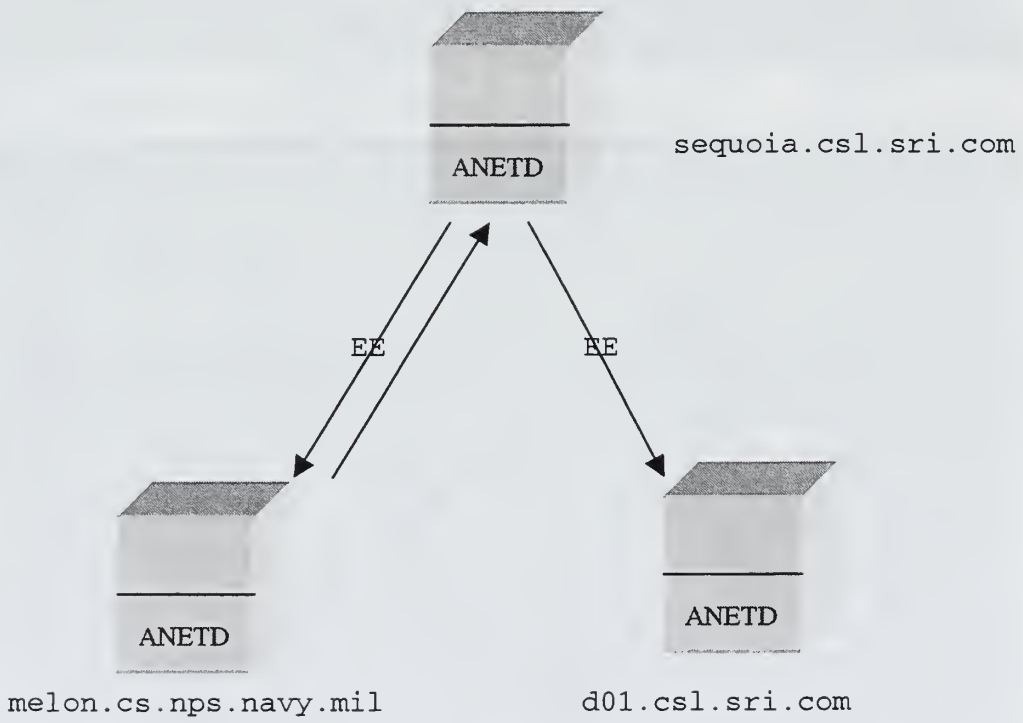


Figure 4.4 Installing execution environments from the code server.

1. ANTS Example

This example shows installing a small ANTS active network. There should be ANETD running in the hosts to run this example. So if the hosts that you are going to make experiments do not have ANETD, you should install ANETD. The hosts used in this example are part of the ABONE, so all of them have ANETD running. We do not have to worry installing ANETD.

Our node melon.cs.nps.navy.mil is going to install ANTS and run an ANTS program in the active network in this example. The situation can be illustrated in Figure 4.4. We are going to use the files in ANETD distribution. First we have to change the data.config file in ANETD distribution, which is configured for three ports in one machine. This file is shown in Figure 4.5. The hosts are shown underlined in the file.

```
# - three nodes (source, router, destination)
# - duplex connected by udp
# - all running on the same machine
# - source pinging destination


node 18.31.12.1 -routes data.routes -updateRoutes true
channel 18.31.12.1 melon.cs.nps.navy.mil:3322 -log 0
application 18.31.12.1 apps.DataServerApplication -target
18.31.12.3

node 18.31.12.2 -routes data.routes -updateRoutes true
channel 18.31.12.2 sequoia.csl.sri.com:3322 -log 0

node 18.31.12.3 -routes data.routes -updateRoutes true
application 18.31.12.3 apps.DataClientApplication
channel 18.31.12.3 d01.csl.sri.com:3322

connect 18.31.12.1 18.31.12.2
connect 18.31.12.2 18.31.12.3
```

Figure 4.5 Data.config File.

We can set the environment variable display to the X11 display being used. This is for output purposes. If this variable is set, then the output can be seen on screen while it is being executed. To do this type;

export DISPLAY="melon.cs.nps.navy.mil:0.0"

We can now type the command "make" in the top directory that you are running this example. This creates a couple of scripts and a static routing table. The routing table is fed to the active network. The routing table is shown in Figure 4.6.

```
# shortest routes, automatically generated

# source     destination next        addr

18.31.12.1  18.31.12.2  18.31.12.2     sequoia.csl.sri.com:3322
18.31.12.1  18.31.12.3  18.31.12.2     sequoia.csl.sri.com:3322
18.31.12.2  18.31.12.1  18.31.12.1     melon.cs.nps.navy.mil:3322
18.31.12.2  18.31.12.3  18.31.12.3     d01.csl.sri.com:3322
18.31.12.3  18.31.12.1  18.31.12.2     sequoia.csl.sri.com:3322
18.31.12.3  18.31.12.2  18.31.12.2     sequoia.csl.sri.com:3322
```

Figure 4.6 Data.routes File.

We are now ready to start our small ANTS network. Data.start file can be used to test the active network. This file is a script file generated with the make command. You can find the makeroutes file, which is invoked with make command in [Ref:9]. Data.start file is shown in Figure 4.7.

When we start our active network, the routing table and the configuration file are sent to hosts where they are stored. Then the necessary Java code from the code server is downloaded to the hosts and the active network starts to run.

The file data.stop stops the existing active network and gives back all the resources. It is shown in Figure 4.8. The active network

should be stopped after we are done. Because other nodes sharing the ABONE also need the resources. It is strongly advised to stop the network at the end.

```
java ants.ConfigurationManager ping.config 18.31.12.3 >&
18.31.12.3.log &
java ants.ConfigurationManager ping.config 18.31.12.2 >&
18.31.12.2.log &
java ants.ConfigurationManager ping.config 18.31.12.1
kill %?18.31.12.3
kill %?18.31.12.2
```

Figure 4.7 Data.start.

```
kill_node melon.cs.nps.navy.mil 3322
kill_node sequoia.csl.sri.com   3322
kill_node d01.csl.sri.com       3322
```

Figure 4.8 Data.stop.

A log file is also recorded, while the active network is running. The log file is shown in Appendix G. The first lines show the query statements, which start the active network. And the last lines show the kill statements, which stop the active network.

2. PLAN Example

This example shows how to install a small PLAN active network. This example is very similar to the ANTS example. The same hosts will be used that are used in the ANTS example. The concept is the same. Melon.cs.nps.navy.mil will install PLAN execution environment to the hosts.

ANETD is used also for this example. So the file hostfile, which is found in ANETD distribution, is configured for our example active

network. Hostfile is shown in Figure 4.9. Melon.cs.nps.navy.mil is the host0 as it is seen.

```
130.107.4.25   host2 sequoia.csl.sri.com
130.107.19.101 host1 d01.csl.sri.com
131.120.1.244  host0 melon.cs.nps.navy.mil
```

Figure 4.9 Hostfile.

The variable DISPLAY can be set to see the output of the active network. It is set the same way as the ANTS is set. Type the command below to set the DISPLAY environment in a bash shell.

export DISPLAY="melon.cs.nps.navy.mil:0.0"

The hostfile and the environment variable are ready. Now type setup in the directory where your hostfile file is stored. The setup file can be found in [Ref 10]. Setup creates the "start_network" script based on the information in the hostfile.

When start_example script is run, the configuration files are sent to all the active nodes. The PLAN execution environment is then downloaded to the nodes and executes to start the PLANet daemons.

Now the start_example script can be run. This script sends a PLAN traceroute program, which is written in PLAN. Start_example file can also be found in [Ref 11]. The active network can be stopped with the stop_network script.

V. VERIFICATION

We have verified the functionality of the prototype active router with respect to SAAM requirements. In particular, we have examined its support for server probing. The detail is presented in this chapter.

Any program can be sent to an active router if the execution environment for that program has been installed and activated on that router. As shown in the previous chapter, an execution environment can be deployed to the prototype router using ANETD. Two execution environments, which were deployed in the examples, were PLAN and ANTS. Now any program written in PLAN or ANTS can be sent to those routers.

Three programs will be presented in this chapter. The programs are written for the PLAN execution environment. All three programs are server-probing examples. The objective of the programs is to traverse the selected routers and record the arrival time to each router. Such functionality is required by SAAM. A SAAM server will need to send probes similar to these programs that travel along a specific path and record the arrival times and other statistics at each router.

The programs were evaluated on a PLAN active network consisting of three routers. For simplicity, we simulated the network on the same machine, melon.cs.nps.navy.mil. The PLAN execution environment was activated on three ports, 3324, 3325 and 3326, each of which emulates a PLAN aware router.

The test configuration is shown in Figure 5.1. The deployment of the PLAN execution environment to the nodes is shown with the continuous lines. The dotted lines show the PLAN programs sent by the server to the nodes. Each of the programs traverses the nodes and collects information for the server.

First, the command below was typed to start the PLAN active router on the default active port 3324.

```
java PLAN.ARMain –v
```

The other active routers are started with the commands below. The main active router is also declared, which is using the port 3324. The detailed messages generated for establishing the PLAN active network are shown in Appendix H.

java PLAN.ARMain –v –ip 3325 –m melon.cs.nps.navy.mil:3324

java PLAN.ARMain –v –ip 3326 –m melon.cs.nps.navy.mil:3324

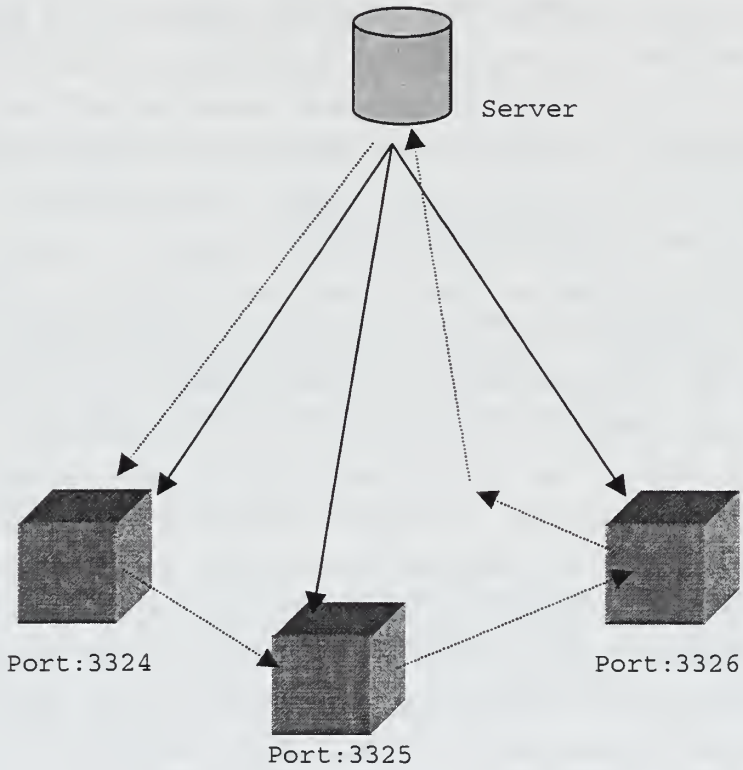


Figure 5.1 Active Network Example

A. PROGRAM TRACEROUTE_TIMESTAMP

The program traceroute_timestamp is shown in Figure 5.2. PLAN is a functional language as it is seen in the program and all the PLAN programs are stored with the plan extension like the java extension in java programs.

There are two functions in the program. The functions are ack and collect. Collect is the main function. This function is called to invoke the program. The program can be injected into the network with the command below;

java PLAN.PLANStart –v traceroute_timestamp.plan 60

melon.cs.nps.navy.mil:3324

```
(* Acknowledgement for each hop *)

fun ack(count:int,where:host, record_time:int*int) : unit =

(* prints the name and the ip address of the nodes , then
 the entering time to that node *)
(
     print("lines");print(where);print(":"); print(count);
     print(" trip time is ");print(record_time);print("\n")
)

(* this is the main function of the program *)

fun collect(source:host,destination:host,count:int):unit =
(
  let val record_time: int * int = getTime() in

    (* First send response back to source that we got this
     far *)

    OnRemote( ack(count,thisHost(),record_time), source,
    count, defaultRoute )
  end;

  (* Then continue on our way *)

  if (thisHost () <> destination)
  then
    let val next:host = defaultRoute(destination)   in

         OnNeighbor(collect( source, destination, count+1 )
          ,next, getRB () )
     end

    (* We've reached the destination, so we're done *)

  else ()
  )
```

Figure 5.2 TRACEROUTE_TIMESTAMP.PLAN.

This command injects the traceroute_timestamp.plan program into the existing PLAN active network from the designated port. When the program comes to the first node it is executed. The command line below is typed to invoke this program after injecting it into the network.

collect (getHostByName("melon.cs.nps.navy.mil:3324"),
getHostByName("melon.cs.nps.navy.mil:3326"), 1)

```
nxterm
Using port 3324
Getting Initial Resource ...
Getting Address of Active Router ...
Checking for parse errors in source code ...
Binding Top Environ ...
Parsing the initial invocation ...

[root@melon rout_tests]# java PLAN.PLANStart -v traceroute_timestamp.plan 60 mel
on.cs.nps.navy.mil:3324
Using port 3324
Getting Initial Resource ...
Getting Address of Active Router ...
Checking for parse errors in source code ...
Binding Top Environ ...
Parsing the initial invocation ...
collect(getHostByName("melon.cs.nps.navy.mil:3324"), getHostByName("melon.cs.nps
.navy.mil:3326"), 1)
IPv4UDP : (melon.cs.nps.navy.mil/131.120.1.244, 3324) : 1
 trip time is ( 917415870, 445 )
IPv4UDP : (melon.cs.nps.navy.mil/131.120.1.244, 3325) : 2
 trip time is ( 917415870, 733 )
IPv4UDP : (melon.cs.nps.navy.mil/131.120.1.244, 3326) : 3
 trip time is ( 917415870, 960 )
```

Figure 5.3 The output of TRACEROUTE_TIMESTAMP.

The invocation line and the output of the program are shown in Figure 5.3. The collect function takes three arguments. First argument is the source address. The source address is entered with the getHostByName service available in PLAN. This service takes a string according to the grammar below and converts it to a host. Host is a type in PLAN.

```
name        ::= host | host : port-num
host        ::= ip-addr | domain-name
port-num    ::= int-literal
domain-name ::= string-literal
```

Where ip-addr has the form n.n.n.n where each n is an integer from 0 to 255. If the port-num argument is not provided, then it is assumed to be the same port as the invoking node. The service getHostByName attempts to resolve this name into a value of the PLAN host type.

The second argument is another host name. It is invoked again with the getHostByName service. The third argument is an integer. The number 1 is entered and is assigned to integer variable count. This integer can be chosen arbitrarily, it is chosen just to give numbers to the nodes that the program visits.

After the program is invoked with the initial arguments, The next line below is started to execute.

let val record_time: int * int = getTime() *in*

This line shows how to assign a variable. The getTime() service exists in PLAN core services. It returns a pair (2-tuple) consisting of the number of seconds and milliseconds since January 1, 1970, 00:00:00 GMT. The word val shows that the record_time is a variable and its type is int * int. It means that record_time consists of two integers , the first integer shows the seconds of the time and the second integer shows the milliseconds of the time. The words *let*, *in* and *end* define the scope of the variable.

Next we explain two network primitive operations in PLAN. They are OnRemote and OnNeighbor. In some sense , the network primitives are the most interesting aspect of PLAN, as they enable mobile computation via generating and sending new active packets. These network primitives make the PLAN execution environment superior to other execution environments.

OnRemote is the basic netwrok primitive. Its syntax is:

OnRemote (E, H, Rb, Routing)

The meaning of this primitive is: evaluate E on host H. Furthermore, use the Routing function to determine how to get to H. E must be a function call. H is an expression of type host. Rb is an integer indicating how much of the parent's global resource bound should be transferred to the child packet. Routing should be a function or service. Routing (h) should return some host h', which is a neighbor of the current node. H' is supposed to be the next hop on a route to h in the routing scheme represented by Routing.

On success, the call to OnRemote will create a PLAN packet, which is sent to next hop. In the event of an error, one of the two exceptions can be raised. They are NotEnoughRB and HostNotLocal.

The second network primitive OnNeighbor is similar to OnRemote with the restriction that the child packet generated with this command must be executed on a neighbor of the current node. Its syntax is:

OnNeighbor (E, H, Rb)

The meaning of this primitive is: evaluate E on H. E must be a function call. H is an expression of type host and it must be a neighbor of the current node. Rb is an integer indicating how much of the parent's resource bound should be transferred to the child packet. In the event of an error the same exceptions will be raised.

The invocation line can be seen in Figure 5.3. After invoking the program with this line, record_time variable gets assigned the current time in the first node. Then the OnRemote line is executed. This line sends the Ack function to the source to be evaluated.

OnRemote(ack (count, thisHost(), record_time), source, count, defaultRoute)

Ack function takes three arguments and prints the result. The first Ack message sent can be seen in Figure 5.4. Then it is compared whether thisHost is equal to destination. The current host is 3324 and destination is 3326 so it is not equal. Then statement is executed next. The inequality operator < > is evaluated to true.

Next variable inside the then statement is assigned the next hop. OnNeighbor call is executed after that. OnNeighbor creates a new packet sends it to the next hop, it also calls the collect function with three arguments to be evaluated in the next hop. The collect function becomes a recursive call, when it is called. The flow of the program can be seen in Figure 5.4. The Acks show the OnRemote calls. The direct lines from 3324 to 3325 and from 3325 to 3226 show the OnNeighbor calls. The OnNeighbor is executed till the if statement becomes false. When thisHost is equal to destination the line below is executed.

OnNeighbor(collect(source, destination, count+1), next, getRB())

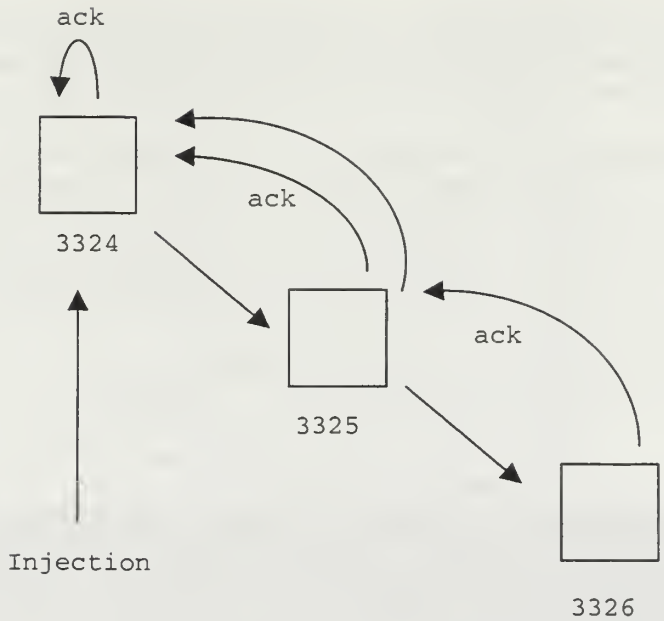


Figure 5.4 The evaluation of Traceroute_timestamp.plan.

B. PROGRAM TRACEROUTE_LIST

This program is shown in Figure 5.5. There are two functions in this program. The collectroute is the main function. It takes five arguments. The first argument is the source address. The second argument is the destination address. The third argument is the path of nodes, which is entered as a list. The fourth argument is the empty list, this list will gather the seconds of the time when the program visits the node. The fifth argument is another empty list. This list will gather the milliseconds of the time when the program visits that node.

The invocation line for the program can be seen in Figure 5.6 and Figure 5.7. It is:

```
collectroute( getHostByName("melon.cs.nps.navy.mil:3324"),
              getHostByName("melon.cs.nps.navy.mil:3326"),
              [getHostByName("melon.cs.nps.navy.mil:3324");
               getHostByName("melon.cs.nps.navy.mil:3325");
              getHostByName("melon.cs.nps.navy.mil:3326")],
              [], [])
```

```
(*Code to send the list back and print it upon completion*)
fun ack( secs:int list, milisecs:int list) : unit =
(
  let val    thirdsec:int    = (hd secs)
      val    secondsec:int   = (hd (tl(secs)) )
      val    firstsec:int    = (hd (tl(tl(secs))) )
      val    thirdmili:int   = (hd milisecs)
      val    secondmili:int  = (hd (tl(milisecs)) )
      val    firstmili:int   = (hd (tl(tl(milisecs))) ) in
      (
      print("Timestamp at node 1 is ");
      print(firstsec);print(" seconds ");
      print(firstmili); print(" miliseconds ");
      print("\n"); print("Timestamp at node 2 is ");
      print(secondsec);print(" seconds ");
      print(secondmili); print(" miliseconds ");
      print("\n"); print("Timestamp at node 3 is ");
      print(thirdsec);print( " seconds ");print(thirdmili);
      print(" miliseconds ");print("\n")

      )
  end
 )


(* Workhorse *)
fun collectroute (source:host,destination:host, ipaddress:
host list, secs: int list, milisecs: int list) : unit =

let val this:host = (hd ipaddress)
     val time:int  = (snd getTime())
     val sec:int  = (fst getTime())      in
 (

    if (this <>  destination )
    then
     (
      let val next:host =( hd  (tl ipaddress)) in
          (

        OnNeighbor (collectroute (source,destination, (tl
        ipaddress), sec::secs, time::milisecs), next,
        getRB ())
           )
      end
      )
    else
    (
       OnRemote(ack(sec::secs, time::milisecs ), source,
       getRB(), defaultRoute) ))
  end
```

Figure 5.5 TRACEROUTE_LIST.PLAN.

The port number 3324 is the source, 3326 is the destination node. The program will go through 3324, 3325 and 3326. This is the same path like the previous program.

This program visits the nodes according to the input list. It records the visiting time into the lists. When the program is started, the variables this, time and sec are assigned in the first node. Then the if statement checks whether it reaches the destination. If it does not reach the destination, the OnNeighbor call is executed on the next node. This means a new plan packet is sent to the next node calling the collectroute function recursively.

```
nxterm
[root@melon rout_tests]# java PLAN.PLANStart -v traceroute_list.plan 60 melon.cs
.nps.navy.mil:3324
Using port 3324
Getting Initial Resource  ...
Getting Address of Active Router  ...
Checking for parse errors in source code ...
Binding Top Environ ...
Parsing the initial invocation ...
collectroute(getHostByName("melon.cs.nps.navy.mil:3324"), getHostByName("melon.c
s.nps.navy.mil:3326"),[getHostByName("melon.cs.nps.navy.mil:3324");getHostByName
("melon.cs.nps.navy.mil:3325");getHostByName("melon.cs.nps.navy.mil:3326")], [],
 [])
Timestamp at node 1 is 917462479 seconds 610 miliseconds
Timestamp at node 2 is 917462480 seconds 114 miliseconds
Timestamp at node 3 is 917462480 seconds 559 miliseconds

[root@melon rout_tests]# java PLAN.PLANStart -v traceroute_list.plan 60 melon.cs
.nps.navy.mil:3324
Using port 3324
Getting Initial Resource  ...
Getting Address of Active Router  ...
Checking for parse errors in source code ...
Binding Top Environ ...
Parsing the initial invocation ...
collectroute(getHostByName("melon.cs.nps.navy.mil:3324"), getHostByName("melon.c
s.nps.navy.mil:3326"), [getHostByName("melon.cs.nps.navy.mil:3324");getHostByNam
e("melon.cs.nps.navy.mil:3325");getHostByName("melon.cs.nps.navy.mil:3326")],[],
[])
Timestamp at node 1 is 917464270 seconds 927 miliseconds
Timestamp at node 2 is 917464271 seconds 356 miliseconds
Timestamp at node 3 is 917464271 seconds 820 miliseconds
[]
```

Figure 5.6 First output of TRACEROUTE_LIST.

When the program reaches the destination the if statement becomes false and the OnRemote call is executed. This line creates a new packet and sends it to the source to be evaluated. OnRemote invokes the ack function in the source with two lists. The ack function takes these two lists and prints them.

```
[root@melon /root]# cd /
[root@melon /]# cd home/nkaplan/planc/PLAN/rout_tests
[root@melon rout_tests]# java PLAN.PLANStart -v traceroute_listONE.plan 60 melon
.cs.nps.navy.mil:3324
[root@melon rout_tests]# java PLAN.PLANStart -v traceroute_list.plan 60 melon.cs
.nps.navy.mil:3324
Using port 3324
Getting Initial Resource  ...
Getting Address of Active Router  ...
Checking for parse errors in source code ...
Binding Top Environ ...
Parsing the initial invocation ...
collectroute(getHostByName("melon.cs.nps.navy.mil:3324"), getHostByName("melon.c
s.nps.navy.mil:3326"),[getHostByName("melon.cs.nps.navy.mil:3324");getHostByName
("melon.cs.nps.navy.mil:3325");getHostByName("melon.cs.nps.navy.mil:3326")],[],[
])
Timestamp at node 1 is 917415385 seconds 684 miliseconds
Timestamp at node 2 is 917415386 seconds 205 miliseconds
Timestamp at node 3 is 917415386 seconds 669 miliseconds
[]
```

Figure 5.7 Second output of TRACEROUTE_TIMESTAMP.

C. PROGRAM TRACEROUTE_ROUNDTRIP

This program is similar to traditional ping programs. There are three functions in this program. They are startcollect, collect and ack. The startcollect is the main function so it is invoked first. The startcollect function takes three arguments. They are the source address, destination address and an integer. This integer can be entered arbitrarily. It is just for node numbering.

The main function can be invoked with below line. The invocation line can also be seen in Figure 5.8.

startcollect (getHostByName (“melon.cs.nps.navy.mil:3324”),
getHostByName (“melon.cs.nps.navy.mil:3326”), 1)

When the program is started , the variable start_time is assigned the current time in the first node. Then the if statement is checked if the program reached the destination. The program is still at the first node so the OnNeighbor is called. OnNeighbor sends e new packet to the next node with the collect function. The collect function has three arguments. The source address, destination address, count+1 and the start_time variable. When the collect function is executed at the second node, the OnRemote sends a new packet to the source by invoking the ack function with the start_time variable.

The ack function takes three arguments. The third argument is the start_time variable, which indicates the time when the program visits the first node. Inside the ack function the getTime () service is called again and the current time is assigned to the variable now.

The fst and snd are two operators, which fst returns the first element of a tuple and snd returns the second element of a tuple. Sec is assigned the seconds of the the time and the diff is assigned to the milliseconds of the time.

```
Let  val now:int * int = getTime()
     val diff: int = (snd now) - (snd start_time)
     val sec : int = (fst now) - (fst start_time)  in
```

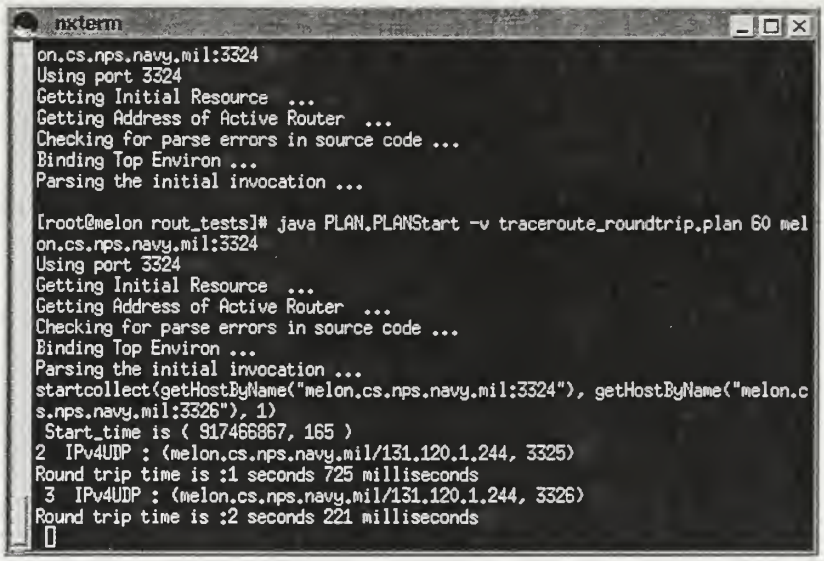


Figure 5.8 Output of TRACEROUTE_ROUNDTRIP.

The sec and diff show the passed time while the program travels from the first node to the second node and also the time passed from the second node to the third node.

When the if statement in the collect function is equal to false, the program reaches the destination. The output of the program is shown in Figure 5.8. The line starting with number 2 shows the visited node after the first node and gives the time difference between the second node and the first node. The line starting with number 3 shows the

third node and the time difference. The program traceroute_roundtrip is shown in Figure 5.9.

These programs collect the time when they enter the router and bring them back. The SAAM server can interpret the delays between the routers from the results coming with these programs. Other programs can easily be written to probe the routers in different ways. As a result the active router can support server probing.

```
(* Acknowledgement for each hop *)
fun ack(count:int,where:host, start_time:int * int) :
unit =


   let val now:int * int = getTime()
       val diff:int = (snd now) - (snd start_time)
       val sec : int = (fst now) - (fst start_time)  in

   (* prints the name and the ip address of the nodes ,
      then the entering time to that node *)
   (
    print(count);print("  "); printwhere); print("\n");
    print("Round trip time is :"); print(sec);
    print("  seconds ");print(diff);
    print(" milliseconds "); print(" \n ")

    )
   end

fun collect  (source:host, destination:host, count:int,
start_time:int * int) : unit =
(  (* start of function *)


 (* First send response back to source that we got this
    far *)
   (
   OnRemote(ack(count,thisHost(),start_time), source,
            count, defaultRoute);

 (* Then continue on our way *)
   if (thisHost () <> destination)
   then
     let val next:host = defaultRoute(destination)  in

         OnNeighbor ( collect( source, destination,
                  count+1,start_time), next, getRB () )
     end
```

```
(* We've reached the destination, so we're done *)
   else ()
   )

) (* end of function *)



(* this is the main function of the program *)
fun startcollect (source:host, destination:host,
count:int) : unit =

   let val start_time: int * int = getTime()
        in
      (
        print(" Start_time is "); print(start_time);
        print("\n");
        if (thisHost () <> destination)
        then
          let val next:host = defaultRoute(destination)
          in
             OnNeighbor ( collect( source, destination,
                  count+1,start_time), next, getRB () )
          end
        else ()
     )end
```

Figure 5.9 Output of TRACEROUTE_ROUNDTRIP.

VI. CONCLUSIONS

SAAM deploys dedicated servers that perform decision-making tasks for the routers. This enables the deployment of lightweight routers in a SAAM environment. A lightweight router was designed and implemented in this thesis.

The lightweight router was built using the Active Networking approach. A SAAM server can inject the customized programs (such as server probes) into the network made of this type of routers. ANETD was chosen as the Active Networking platform for the router. There are two reasons for this choice. First, the router must support ANETD for it to be a node in the ABONE. The ABONE facilities automated deployment of an execution environment to a set of selected nodes. An active networking testbed can be formed rapidly this way. Second, ANETD provides the router the capability of binding multiple execution environments (EEs) to a single port. ANETD will de-multiplex an active packet arriving at the port to the appropriate EE. This capability is useful when there is a need to compare probe programs written for different EEs.

A DELL XPSR400 PC was used as the base platform on which the lightweight router was emulated. Linux was chosen as the node operating system for the lightweight router. All major execution environments for active networking, i.e., PLAN, ANTS and SMART PACKETS were evaluated in this thesis. Particularly PLAN was chosen for implementing the server probing programs.

In summary, the work in this thesis has laid the groundwork for more in-depth SAAM server and router research by establishing an experimental router that is a part of a wide area testbed (ABONE). A set of server probing experiments was conducted using the router and testbed. The results show that it is straightforward for a SAAM server to collect performance information from lightweight routers that support active networking. The successful completion of the experiments also demonstrates the usefulness of the testbed.

A. LESSONS LEARNED

The major lesson learned is that careful planning is required when installing multiple operating systems to a PC. The reason is that the hardware requirements for different operating systems could be

dramatically different. In particular, experimental operating systems like Linux and NetBSD often do not have the drivers for more recent hardware devices. For example, Linux and NetBSD do not support the 3Com905 network card, that originally came with the DELL XPSR400 PC. Therefore, the optimum hardware configuration should be determined and acquired before attempting to install multiple operating systems.

When NetBSD is required, it should be installed on a separate hard drive. This is because when NetBSD is installed to the first hard drive, NetBSD wipes out everything including System Commander files from the drive.

B. EVALUATION OF ACTIVE NETWORKING APPROACH

Active Networking is a new approach for computer networks. The main idea is to make the network programmable. As an important part of this thesis, different aspects of the active networking approach were studied. The main conclusion drawn from the experience is that Active Networking is a promising approach for developing SAAM and similar systems that require a central entity to exercise fine control over the routers. ANETD has all the necessary functionality to support programmable networks.

All the execution environments used in active networking approach are experimental at the moment. All were written in object-oriented languages such as JAVA. Our experience with PLAN showed that this execution environment is robust and easy to use. Compared with ANTS, PLAN programs have a smaller byte code size. It is straightforward to write a PLAN program with built-in services such as OnRemote or OnNeighbor. One problem with these execution environments is that there is not enough documentation for them available at the moment.

C. SUGGESTIONS FOR FUTURE WORK

A lot of work remains to be done for SAAM server probing. This thesis has laid the groundwork for such effort by establishing an experimental router that is a part of a wide area testbed (ABONE).

Our server probing programs simply collect the arrival time to each router. Assuming that all clocks are synchronized, A SAAM server

may use data gathered by these programs to deduce the total packet delay at a link.

Two extensions for the probing programs should be considered. First, it will be worthwhile to develop probing programs that work without the (potentially expensive) assumption of clock synchronization. Second, the SAAM server will need to keep track of more types of router performance information than just packet delays. In many cases, the server is required to know the packet loss rate and the delay jitter at a link. Therefore, it will be essential to develop a set of effective probing programs for each type of information.

Performance is another issue that requires future research. The processing overhead incurred by a probing program at a router must be minimized for two reasons. First, such overhead takes away precious CPU cycles that otherwise can be used for packet forwarding. Second, the extra delay because of probe processing could affect the accuracy of the timing data collected. It would be interesting to measure and compare the processing overheads of different EEs.

LIST OF REFERENCES

1 SAAM proposal, SAAM: Network Management for Integrated Services.

2 Geoffrey Xie, Debra Hensgen, Taylor Kidd, John Yarger, SAAM: An Integrated Network Architecture for Integrated Services.

3 Architectural Framework for Active Networks Version 0.9 Active Networks Working Group August 31, 1998.

4 ANEP: Active Network Encapsulation Protocol, http://www.cis.upenn.edu/~switchware/ANEP/

5 ANETD: Active NETworks Daemon (v1.0) Livio Ricciulli August 10, 1998.

6 Smart Packets for Active Networks Beverly Schwartz, Wenyi Zhou, Alden W.Jackson, W.Timothy Strayer, Dennis Rockwell, Craig Partridge BBN Technologies.

7 Registered nodes in the ABONE, http://www.csl.sri.com/ancors/abone/nodedb/nodes.html

8 Code server registry entry, http://www.csl.sri.com/ancors/abone/web-registry.html

9 Makeroutes file, http://sequoia.csl.sri.com:7000/anetd/ants_ex/makeroutes

10 Setup file, http://sequoia.csl.sri.com:7000/anetd/plan_ex/setup

11 Start_example file, http://sequoia.csl.sri.com:7000/anetd/plan_ex/

APPENDIX A. ANETD CONTROL COMMANDS

LOAD

The load command instructs anetd to download a number of files specified with URLs and start a network service. The load command has the format:

LOAD [T=< anepid >] [J=< jurl > | X =< url > | A =< url >] [F=< url > ...] [E=< var : val > ...] [D=< dir >] [O=< file >] [R=< file >]

- T=< anepid > is the ANEP ID of the service being deployed. If this argument is not present, the deployed service will be assigned type 0 and no packets will be demultiplexed to it by anetd.

- J=< jurl > specifies a Java application.
< jurl > is of the form http:servername.edu:port/classpath/~class where

servername.edu:port specifies the server where the data is located following the normal URL conventions.
classpath is a path pointing to the base classpath of the Java application.
class is the class to invoke.

Both classpath and class can be a series of directories separated by "/". For example,J=http://sequoia.csl.sri.com:7000/java/ants1.2/~ants/ConfigurationManager specifies the URL of the ANTS application ants.ConfigurationManager located on SRI's http code server in the directory java/ants-1.2/.

- X=< url > means that < url > specifies a native binary executable.

- A=< url > means that < url > specifies an ANCORS thread

- F=< url > means that < url > specifies a data file that should be simply transferred and written on the local installation directory.

- S=< string > tells anetd to invoke the deployed service with < string > as a command line argument. < string > cannot contain any white spaces.

- E=< variable >:< value > tells anetd to set the environment variable < variable > to the value < value >.

- D=< dir > tells anetd to use the directory < dir > as the Root directory for the service installation. Anetd by default uses the directory /homedirectory/< clientip >/ for installing all downloaded code; by specifying the D=< dir > option, anetd installs all downloaded code in /homedirectory/< clientip >/<dir >.

- O=< file > redirects the standard output of the network

service to the file < file > (to be created in the installation directory).

- R=< file > redirects the standard error of the network service to the file < file >(to be created in the installation directory).

- C=< description > specifies a description < description > for the deployed services. This description is then returned to the client when query commands are invoked. The description string cannot contain white spaces.

Because the X and A types specify native executables, anetd automatically appends the extensions solaris, linux, and bsd44 to the URLs, depending on what platform anetd is running on. For example, suppose that anetd is running on a Linux machine; the URL http://sequoia.csl.sri.com:7000/executables/ps would actually fetch the file http://sequoia.csl.sri.com:7000/ executables/ps.

QUERY

The query command returns, to the client originating the command, a list of network services that were forked by anetd. The query command format is simply query. The list of forked services has the format:
< index > < clientip > < description >

< index > is an index generated by anetd (0,1,2 etc.).
< clientip > is the IP address of the client that installed the service.
< description > is a textual description of the service.

KILL

The kill command allows a client to terminate a network service by sending a sigint signal. The kill command has the format:

KILL < index >

< index > is the index of the thread to be terminated by anetd (0,1,2 etc.).

The < index > value should be retrieved by using the query command. Anetd will automatically garbage-collect all resources allocated to the terminated service and will only allow the client that originally deployed the service to perform the operation.

GET

The get command allows a client to retrieve a file through anetd. The get command has the format:

GET [D=< dir >] < file >

D=< dir > tells anetd to look in the directory < dir >. Anetd by default uses the directory /homedirectory/< clientip >/ to look for the file < file >; by specifying the D=< dir > option, anetd will look in /homedirectory/< clientip >/< dir >.

< file > specifies the name of the file to retrieve.

Anetd only allows the client that originally created the file < file > to retrieve it. The content of the file is returned to the client in the acknowledge message.

PUT

The put command allows a client to upload a file through anetd. The PUT command has the format

PUT [D=< dir >] < file > < content... >

- D=< dir > tells anetd to look in the directory < dir >. Anetd by default uses the directory

- < file > specifies the name of file to be created.

- < content... > is the data to be stored.

Anetd allows clients to upload files. For example the command PUT config ... will create a file "config" in the installation directory of the client and write the data that follows into it. From the client side this is specified by invoking sc PUT < port > < host > config < filename > where < host > is the name of the machine on which anetd is running, < port > is the port on which anetd is listening, config is the remote file name and < filename > is the local filename.

CONF

The conf command aplies to ANCORS (Adaptable Network COntrol and Reporting System) threads and is similar to a remote procedure call. The CONF command has the format:

CONE < symbol > [args ...]

- < symbol > specifies the name of the function to invoke. The symbol < symbol > is resolved by anetd to a local memory address, and the function is invoked.

- args are a sequence of command line arguments to be passed to the function <symbol >.

APPENDIX B. ANTS PING PROGRAM

PING APLLICATION

```
package apps;

import java.awt.*;

import ants.*;
import utils.*;

/**
 * test GUI application with ping buttons
 *
 * @author David Wetherall
 */

public class PingApplication
  extends Application
  implements Runnable
{
  final public static String[] defaults = {};

  int target, rTotCount = 0, delay;
  long beginTime;
  Button pinger;
  Label latency, received, thruput;

  TextField iterations, interval;

  synchronized public void receive(Capsule cap) {
    super.receive(cap);

    switch ((rTotCount % 100)) {
      case 0:
      beginTime = thisNode().time();
      break;
      case 99:
      double diff =
        (((double)thisNode().time() - (double) beginTime) / 1000.0);
      double thru = 100.0 / diff;
      thruput.setText(String.valueOf(thru));

      if (cap instanceof PingCapsule) {
        PingCapsule pcap = (PingCapsule)cap;
        Xdr buf = new Xdr(pcap.getData(), 0);
        long lat = thisNode().time() - buf.LONG();
        latency.setText(Long.toString(lat) + " ms");
      }
      received.setText(Integer.toString(rTotCount + 1));
      break;
    }

    rTotCount++;
  }
```

```
public boolean handleEvent(Event evt) {
  if (evt.id == Event.ACTION_EVENT && evt.target == pinger) {
    new Thread(this).start();
    return true;
  } else
    return super.handleEvent(evt);
}

public void run()
{
  int iter = Integer.parseInt(iterations.getText()),
    interv = Integer.parseInt(interval.getText());

  for (int i=0; i < iter; i++) {
    ByteArray buf = new ByteArray(Xdr.LONG);
    Xdr xdr = new Xdr(buf, 0);
    xdr.PUT(thisNode().time());
    PingCapsule c = new PingCapsule(port, port, target, buf);
    send(c);
    thisNode().sleep(interv);
  }
}

public void setArgs(KeyArgs k)
  throws Exception
{
  k.merge(defaults);

  for (int i = 0; i < k.length(); i++) {
    if (k.key(i).equals("-target")) {
    target = NodeAddress.fromString(k.arg(i));
    k.strike(i);
    }
  }

  super.setArgs(k);
}

public void start()
  throws Exception
{
  thisNode().register(new PingProtocol());

  resize(400, 200);


  // 2 rows by 2 columns with 2 pixel point spaces in between them
  setLayout(new BorderLayout());

  pinger = new Button("ping " + NodeAddress.toString(target));

  Panel outputDisplay = new Panel();
  outputDisplay.setLayout(new GridLayout(6, 1, 1, 1));
  outputDisplay.add(new Label("Capsules Received", Label.CENTER));
  received = new Label("0", Label.CENTER);
  outputDisplay.add(received);
  latency = new Label("0", Label.CENTER);
```

```
    outputDisplay.add(new Label("Capsule Latency (ms)", Label.CENTER));
    outputDisplay.add(latency);
    thruput = new Label("0", Label.CENTER);
    outputDisplay.add(new Label("Throughput (cap/s)", Label.CENTER));
    outputDisplay.add(thruput);

    Panel iterDisplay = new Panel();
    iterDisplay.setLayout(new BorderLayout());
    iterDisplay.add("Center",
                new Label("Num Iterations", Label.CENTER));
    iterations = new TextField("0");
    iterations.setEditable(true);
    iterDisplay.add("South", iterations);

    Panel intervalDisplay = new Panel();
    intervalDisplay.setLayout(new BorderLayout());
    intervalDisplay.add("Center",
                  new Label("Ping interval (in ms)", Label.CENTER));
    interval = new TextField("0");
    interval.setEditable(true);
    intervalDisplay.add("South", interval);

    Panel topPanel = new Panel(), botPanel = new Panel();
    topPanel.setLayout(new GridLayout(1, 2, 1, 1));
    botPanel.setLayout(new GridLayout(1, 2, 1, 1));

    topPanel.add(pinger);
    topPanel.add(outputDisplay);
    botPanel.add(iterDisplay);
    botPanel.add(intervalDisplay);

    add("South", botPanel);
    add("Center", topPanel);
    pack();
    show();
  }

  public PingApplication()
    throws Exception
  {
    super();
  }
}
```

PING CAPSULE

```
package apps;

import ants.*;


/**
 * ping capsule processing
 *
 * @author David Wetherall
```

```
 */

public class PingCapsule
  extends DataCapsule
{
  final private static byte[] MID = findMID("apps.PingCapsule");
  protected byte[] mid() { return MID; }

  final private static byte[] PID = findPID("apps.PingCapsule");
  protected byte[] pid() { return PID; }

  public boolean ping = false;

  public int length() {
    return super.length() + Xdr.BOOLEAN;
  }

  public Xdr encode() {
    Xdr xdr = super.encode();
    xdr.PUT(ping);

    return xdr;
  }

  public Xdr decode() {
    Xdr xdr = super.decode();
    ping = xdr.BOOLEAN();

    return xdr;
  }


  public boolean evaluate(Node n) {
    if ( n.getAddress() == getDst()) {
      ping = true;
    } else if (ping!=true) {
      return n.routeForNode(this, getDst());
    }

    if (n.getAddress() == getSrc()) {
      return n.deliverToApp(this, dpt);
    } else if (ping) {
      return n.routeForNode(this, getSrc());
    }
    return false;
  }

  public PingCapsule() { }

  public PingCapsule(short sa, short da, int na, ByteArray d) {
    super(sa, da, na, d);
  }

}
```

PING PROTOCOL

```
/**
 * Ping protocol definition
 *
 * @author David Wetherall
 */

public class PingProtocol extends Protocol {

  public PingProtocol() throws Exception {
    startProtocolDefn();

    startGroupDefn();
    addCapsule("apps.PingCapsule");
    endGroupDefn();

    endProtocolDefn();
  }
}
```

APPENDIX C. BUILDING PLAN

There are two ways that you can install the PLAN software. You can simply obtain the class files and execute them directly, or you can obtain all of the source and build it yourself. Note that BOTH distributions contain all of the documents and the sample programs (i.e., all contained within the docs, interp_tests, and rout_tests directories).

PLAN relies on ANEP, the Active Network Encapsulation Protocol. The source code for ANEP is provided with the PLAN distribution.

Source installation:

PLAN was built using a number of publicly available packages. They are:

(a) JDK 1.1.x -- Java Development Kit
Sun's site:
http://java.sun.com/products/JDK/index.html
We used the third-party Linux port, available via:
http://www.blackdown.org/java-linux/Mirrors.cgi

(b) Pizza, a Substantial Companion to Java, version 0.39
http://www.cis.unisa.edu.au/~pizza/

(c) JavaCC, the Java Compiler Compiler, version 0.6.1
http://www.suntest.com/JavaCC/index.html

(*Note: JavaCC is only needed if you change the grammar file, Parser.jjt, in any way. Otherwise, the .java files provided in the distribution will serve)

You must first install all of these packages. Please follow the installation instructions given at each of the sites.

Next, unpack all of the PLAN source. For a UNIX platform, you should have obtained PLAN-java-2.1-src.tar.Z. This can be unpacked simply by doing

```
uncompress PLAN-java-2.1-src.tar.Z
tar -xvf PLAN-java-2.1-src.tar
```

Note that these operations should be performed in the directory that you would like the source to be unpacked. This shall hereafter be referred to as the "top level directory." This will create 3 directories: "PLAN" which contains the PLAN source (this directory shall hereafter be referred to as the "PLAN directory"), "ANEP" which contains the ANEP source, and "Log" which contains code for a logging facility used by both ANEP and PLAN.

For Windows, you should have obtained PLAN-java-2.1-src.zip. This may be unpacked with PKzip, or WinZip, or a compatible utility.

Building:

The first thing that you must do is make the ANEP and Log packages which PLAN relies on. To build the ANEP package, go to the ANEP directory and read the README file which contains building instructions. For the Log package, simply go to the Log directory and type "javac Log.java" (which assumes the java compiler javac is in your PATH). Now you may build PLAN:

* Using make (UNIX, some Windows systems)

The easiest way to build the software is to use the provided Makefile. Simple type "make" from within the PLAN directory.

This Makefile assumes a couple of things:

1) jjtree, javacc are in your PATH (executables from JavaCC) and the JavaCC libraries are in your CLASSPATH.

2) a file "pc" exists in your PATH. This should be a script that executes the pizza compiler (the pizza installation instructions indicate that on UNIX, pc should be an alias for "java -ms8m pizza.compiler.Main"; instead, we choose to create a shell script "pc" that contains "java -ms8m pizza.compiler.Main $*"; for Windows, pc.bat is provided with Pizza). This presupposes that the pizza and java libraries are accessible from your CLASSPATH,
and that the java interpreter, java, is also in your PATH.
3) the PLAN, ANEP, and Log packages are in your CLASSPATH. This amounts to adding the top level directory of the PLAN distribution to your CLASSPATH (not the PLAN directory, but
the directory that it resides in).

Note that when making for Windows, you should make sure that the cleanup.bat file is being invoked, rather than the cleanup shell script (see the Makefile for more details).

* Building without make (Windows)

If you don't have access to make and are running DOS/Windows, do the following from the PLAN directory:

1) type "jjtree Parser.jjt". This will build the Parser.jj file
2) type "cleanup". This will execute the cleanup.bat script which removes some automatically generated files.
2) type "javacc Parser.jj". This will create the necessary .java files.
3) type "build". This will actually compile all of the .java and .pizza files.

If you modify any of the .pizza or .java files, but do not change the Parser.jjt file, you can safely rebuild the system using only Build.bat.

These instructions assume that the Java and JavaCC executables are in your PATH; particularly java, jjtree, and javacc.

In addition, your CLASSPATH must be set up properly to include the JavaCC and Pizza libraries, as indicated in the installation documentation for those packages. Finally, the PLAN, ANEP, and Log packages must also be in your CLASSPATH.

This amounts to adding the top level directory of the PLAN distribution to your CLASSPATH (not the PLAN directory, but the directory that it resides in).

Classfile Installation:

If you are not interested in acquiring the source code, you can instead just obtain the class files and use those directly. However, you will still need to install the Pizza distribution (see above), since the PLAN code relies on some of the provided Pizza class files. You will not need to install JavaCC.

For UNIX, you should have obtained PLAN-java-2.1.tar.Z. To unpack this file:

```
uncompress PLAN-java-2.1.tar.Z
tar -xvf PLAN-java-2.1.tar
```

Note that these operations should be performed in the directory that you would like the source to be unpacked. This shall hereafter be referred to as the "top level directory." This will create 3 directories: "PLAN" which contains the PLAN classes (this directory shall hereafter be referred to as the "PLAN directory"), "ANEP" which contains the ANEP classes, and "Log" which contains code for a logging facility used by both ANEP and PLAN.

For Windows, you should have obtained PLAN-java-2.1.zip. This may be unpacked with PKzip, or WinZip, or a compatible utility.

You are now ready to execute the PLAN software. Please see the tutorial document for specific instructions.

(*Note: your CLASSPATH must be set up properly to include the Pizza library class files, as indicated in the installation documentation for that package. It must also include the PLAN, ANEP, and Log packages. This amounts to adding the top level directory of the PLAN distribution to your CLASSPATH (not the PLAN directory, but the directory that it resides in)).

Contact Information:

PLAN Home Page:
http://www.cis.upenn.edu/~switchware/PLAN

ANEP Home Page:
http://www.cis.upenn.edu/~switchware/ANEP

Supported Architectures:

This version of the PLAN interpreter has been tested on

* i586 platforms running
  - Redhat Linux 4.1 and 4.2 (with JDK 1.1.1)
  - Windows95 (with JDK 1.1.3)
* Sun SPARC's running
  - SunOS 5.5.1 (with JDK 1.1.3)

APPENDIX D. LOG AND ARMAIN FILES

LOG FILE

```
package Log;
import java.text.*;
import java.io.*;
import java.util.Date;

public class Log {

  public final static int QUIET_LEVEL = 0;
  public final static int ERROR_LEVEL = 1;
  public final static int VERBOSE_LEVEL = 2;
  public final static int DEBUG_LEVEL = 3;

  // set default output to stdout
  private static String logfn = null;
  private static OutputStream logfs =
    new FileOutputStream(FileDescriptor.out);
  private static int level = ERROR_LEVEL;
  private static DateFormat df =
     DateFormat.getDateTimeInstance();
  private static boolean sol = true;

  private static String preamble() {
    Date d = new Date();
    return df.format(d);
  }

  // methods to alter the logging location
  public synchronized static void setLogFile(String logFileName)
  {
    logfs = null;
    logfn = logFileName;
  }

  // alter the logging level
  public synchronized static void setLevel(int l) {
```

```
  level = l;
}

// output methods
//
// if output is directed to a file, the file is reopened
// each time; this allows the file to be removed/truncated
// at any time without disturbing the logging function

public synchronized static void print(int priority, String s) {
  //    System.out.println("level = "+level+", priority =
                              "+priority);
  if (priority <= level) {
    try {
      if (logfn != null) {
        logfs = new FileOutputStream(logfn,true);
      if (sol)
          logfs.write((preamble()+": ").getBytes());
      logfs.write(s.getBytes());
        logfs.close();
      sol = false;
        logfs = null;
      }
      else {
      if (sol)
        logfs.write((preamble()+": ").getBytes());
      logfs.write(s.getBytes());
      sol = false;
      }
    } catch (IOException e) {
    }
  }
}

public synchronized static void println(int priority, String s)
{
  if (priority <= level) {
    try {
```

```
        if (logfn != null) {
          logfs = new FileOutputStream(logfn,true);
        if (sol)
            logfs.write((preamble()+": ").getBytes());
        logfs.write((s+"\n").getBytes());
          logfs.flush();
          logfs.close();
        sol = true;
          logfs = null;
        }
        else {
        if (sol)
          logfs.write((preamble()+": ").getBytes());
        logfs.write((s+"\n").getBytes());
          logfs.flush();
        sol = true;
        }
      } catch (IOException e) {
      }
    }
  }

  public static void print(String s) {
    print(VERBOSE_LEVEL,s);
  }

  public static void println(String s) {
    println(VERBOSE_LEVEL,s);
  }
}
```

ARMAIN FILE

```
package PLAN;

import PLAN.basis.*;
import PLAN.interpreter.*;
import PLAN.net.*;
import PLAN.SLRP.*;
import PLAN.util.*;
import PLAN.resident.*;
import PLAN.port.*;
import PLAN.install.*;
import PLAN.fixedroute.*;
import PLAN.ANON.*;
import java.net.*;
import java.io.IOException;
import pizza.lang.*;
import ANEP.*;
import Log.*;

class ARMain {

  static ActiveHost masterHost;
  static ActiveHost hubHost;
  static String routTabFile;
  static List<ActiveHost> neighbors = List.Nil;
  static int inport = 3324;
  static boolean interactive = false;
  static boolean IamTheMaster = true;

//   static String serviceDir = "./Services";
//   static String serviceIndexFile = "Index";

  static void usage() {
    System.out.println("Usage : java ARMain [-i] [-v|-d|-q] [-l
    logfile] [-ip port]\n" + "[[-m master] [-h hub] | [-rf
    routtab file]]\n" + " [-n n1,n2,...,nk] \n");
  }
```

```
static void parse_args (String [] argv) {
  int i;

  for (i=0; i<argv.length; i++) {
    if (argv[i].equals ("-?")) {
    usage();
    System.exit(1);
    }
    else if (argv[i].equals ("-v")) {
    Log.setLevel(Log.VERBOSE_LEVEL);
    Log.println("ARMain: verbose mode on.");
    }
    else if (argv[i].equals ("-d")) {
    Log.setLevel(Log.DEBUG_LEVEL);
    Log.println("ARMain: debug mode on.");
    }
    else if (argv[i].equals ("-q")) {
    Log.setLevel(Log.QUIET_LEVEL);
    }
    else if (argv[i].equals ("-i")) {
    Log.println("ARMain: interactive mode on.");
    interactive = true;
    }
    else if (argv[i].equals ("-m")) {
    if (i == argv.length - 1) {
      usage();
      System.exit(1);
    } else {
    masterHost =
    ActiveHost.parseActiveHost(argv[i+1],inport);
      if (masterHost == null) {
        Log.println(Log.ERROR_LEVEL,"ARMain.parse_args: "+
                "failed to parse master request |"+
                argv[i+1] + "|");
        usage();
        System.exit(1);
      }
```

```
    Log.println("ARMain: master set to " + masterHost);
    i++;
  }
  }
  else if (argv[i].equals ("-rf")) {
    if (i == argv.length - 1) {
      usage();
      System.exit(1);
    } else {
    routTabFile = argv[i+1];
    Log.println("ARMain: routTabfile set to " + argv[i+1]);
      i++;
    }
  }
  else if (argv[i].equals ("-l")) {
    if (i == argv.length - 1) {
      usage();
      System.exit(1);
    } else {
    Log.setLogFile(argv[i+1]);
    Log.setLevel(Log.VERBOSE_LEVEL);
    Log.println("ARMain: logfile set to " + argv[i+1]);
      i++;
    }
  }
  else if (argv[i].equals("-ip")) {
  if (i == argv.length - 1) {
    usage();
    System.exit(1);
  } else {
    Log.println("ARMain: using incoming port " + argv[i+1]);
    inport = Integer.parseInt(argv[i+1]);
    i++;
  }
  }
  else if (argv[i].equals("-n")) {
  if (i == argv.length - 1) {
    usage();
```

```
    System.exit(1);
  } else {
    ActiveHost h;
    int curindex = 0;
    int nextindex = argv[i+1].indexOf(",",curindex);
    while(nextindex != -1) {
    h = AtiveHost.parseActiveHost (argv[i+1].substring
       (curindex, nextindex),   inport);
      if (h == null) {
        Log.println(Log.ERROR_LEVEL,"ARMain.parse_args: "+
                "failed to parse neighbor request |"+
                argv[i+1].substring(curindex,nextindex) +
                 "|");
        usage();
        System.exit(1);
      }
      neighbors = List.Cons(h,neighbors);
      Log.println("ARMain: found neighbor " + h);
      curindex = nextindex + 1;
      nextindex = argv[i+1].indexOf(",",curindex);
    }
    h = ActiveHost.parseActiveHost (argv[i+1].substring
        (curindex),inport);
    if (h == null) {
      Log.println(Log.ERROR_LEVEL,"ARMain.parse_args: "+
              "failed to parse neighbor request |"+
              argv[i+1].substring(curindex) + "|");
      usage();
      System.exit(1);
    }
    neighbors = List.Cons(h,neighbors);
    Log.println("ARMain: found neighbor " + h);
    i++;
  }
  }
  else if (argv[i].equals("-h")) {
    if (i == argv.length - 1) {
      usage();
```

```
        System.exit(1);
      } else {
        hubHost = ActiveHost.parseActiveHost(argv[i+1],inport);
        if (hubHost == null) {
          Log.println(Log.ERROR_LEVEL,
                      "ARMain: failed to correctly obtain hub");
          usage();
          System.exit(1);
        }
        else
          Log.println("ARMain: using hub "+hubHost);
        i++;
      }
    }
    else {
      usage();
      System.exit(1);
    }
  }
}

/* Main function -
  First parse the necessary options, then initialize the router
  and set up the initial environment.  Finally, wait infinitely
  for packets to arrive; if they've reached their evalDest,
  interpret them, otherwise send them along their way
  with the specified routing function.
  */

public static void main (String [] argv) throws Exception {
  parse_args (argv);

  // Start the router

  try {
    ActiveRouter.start (interactive, inport);
  } catch (IOException e) {
    Log.println (Log.ERROR_LEVEL,
```

```
            "ARMain: Error while starting up router : " +
            e.getMessage ());
    System.exit (1);
  }

  // Initialize default services

  PLANParser.resetSvcsSymtab();
  PLANParser.hasNet = true;
  installServices();

  // Set up routing

  if (routTabFile == null) {

  // using SLRP

    if (masterHost == null)
    masterHost = ActiveRouter.whoAmI();
    if (!masterHost.equals(ActiveRouter.whoAmI()))
    IamTheMaster = false;

    SLRPSvcImpl.install(PLANParser.getSvcsSymtab());
    SLRP.init(masterHost,neighbors,3);

    try {
    if (IamTheMaster) {
      SLRPmaster.init();
      SLRPMasterSvcImpl.install(PLANParser.getSvcsSymtab());

      // Set up ANON; this is only permitted with SLRP masters
      if (hubHost != null) {
        ANONSvcImpl.install(PLANParser.getSvcsSymtab());
        ANON.init(hubHost);
      }
    } else {
      Value neighValList =
        Value.VList
```

```
        (neighbors.foldl
         (fun(List<Value> l, ActiveHost h) -> List<Value>
          {
           return List.Cons(Value.Host(h),l);
           }, List.Nil));
      ActivePacket p = new
        ActivePacket("fun addme(ver:int,me:host,ns:host
        list):unit = " + "  AddRequest(ver,me,ns)",
        "addme", Value.VList
                 (List.Cons(Value.Int(SLRP.version),
                  List.Cons(Value.Host(ActiveRouter.whoAmI()),
                  List.Cons(neighValList,
                  List.Nil)))),
                 masterHost,5, new Pair(masterHost,0),
                 "SLRP");
      try {
        // XXX create TLV list and set the destination

     ActiveRouter.send_active_packet(p,TLVList.Nil,masterHost);
      } catch (Exception uhe) {
        // if we get an error here, we should just abort
        Log.print(Log.ERROR_LEVEL,
        "ARMain: Error trying to negotiate with master: "+uhe);
        System.exit(1);
      }
    }
    } catch (UnknownHostException uhe) {
    Log.print(Log.ERROR_LEVEL,
            "ARMain: unable to look up local host: "+uhe);
    System.exit(1);
    }
  }

  // using static routing

  else {

    try {
```

```
  FixedRouteSvcImpl.init(routTabFile,neighbors);
  } catch (BadRouteTableFileException e) {
  Log.println(Log.ERROR_LEVEL,"ARMain: "+e);
  System.exit(1);
  }

  FixedRouteSvcImpl.install(PLANParser.getSvcsSymtab());
}

// Main event loop
//
// Waits for packets to arrive and deals with them (either
// via interpretation or forwarding)

Pair<ActivePacket,TLVList> in;
ActivePacket actPack;
TLVList TLVs;
ActiveHost h;

while (true) {
  in = ActiveRouter.recv_active_pkt ();
  actPack = in.fst;
  TLVs = in.snd;
  h = actPack.getEvalDest ();

  try {
  if (h.equals (ActiveRouter.whoAmI()))
    Go.execPacket (actPack,TLVs);
  else {
    Binding B = PLANParser.getSvcsSymtab() .get
                (actPack.getRoutingFn() ) ;
    if (B == null)
      throw new ExecException("Error: no such routing
      function "+  actPack.getRoutingFn());
    else {
      ActiveHost Hop =
        ASTOnRemote.getNextHop(B,h,actPack);
      ActiveRouter.send_active_packet (actPack, TLVs, Hop);
```

```
        }
      }
      } catch (PLANException e) {
      Log.println ("ARMain: Packet raised uncaught exception : "
       +    e.getMessage ());
      } catch (ExecException e) {
      Log.println ("ARMain: Execution exception : " +
      e.getMessage ());
      }
    }
  }

  /* This routine causes all of the "standard" services to be
  installed in the symbol table.  If you don't want
  certain    services installed, simply comment out the
  appropriate code portion */

  private static void installServices() {

    /* Standard router services */
    RouterSvcImpl.install(PLANParser.getSvcsSymtab());

    /* Resident data */
    TimerServer ts = new TimerServer();
    ts.start();

    ResidentSvcImpl.initResidentServices(PLANParser.
    getSvcsSymtab(), ts);

    /* Ports */
    PortSvcImpl.install(PLANParser.getSvcsSymtab());

    /* Dynamically loadable services */
    InstallSvcImpl.install(PLANParser.getSvcsSymtab());
  }
}
```

APPENDIX E. SMART PACKETS

This software and its documentation was written by BBN Corporation under sponsorship by the Defense Advanced Research Projects Agency.

1. SMART PACKETS

Smart Packets is a DARPA-funded Active Networks project focusing on applying active networks technology to network management and monitoring without placing undue burden on the nodes in the network. Some parts of the network are growing faster than Moore's Law predicts, straining the infrastructure, yet to accommodate this growth, the infrastructure must be monitored and managed. Current management techniques, such as polling MIBs, are notappropriate for these overtaxed components.

Messages in active networks are programs that are executed at nodes on the path to one or more target hosts. Smart Packets programs are written in a tightly-encoded, safe language specifically designed to support network management and avoid dangerous constructs and accesses. Smart Packets improves the management of large complex networks by (1) moving management decision points closer to the node being managed, (2) targeting specific aspects of the node for information rather than scatter-shot collection, and (3) abstracting the management concepts to language constructs, allowing nimble network control.

2. INSTALLATION

To install on a new system, first unpack the distribution:

```
$ gunzip spkt-1.0.1.tar.gz
$ tar xvf spkt-1.0.1.tar
$ cd spkt-1.0.1
```

Then read the installation instructions:

```
$ more INSTALL
```

Configure the software for your machine:

```
$ ./configure --prefix=/usr/spkt    # specify install location;
```

Make the software, then install it:

```
$ make
$ make install
```

There is one compilation warning, which occurs in the sprocket directory during the compile of sprcomp. Everything else compiles cleanly. If you receive a virtual memory exhausted error, increase the "datesize" process resource limit to at least 128K.

3. EXECUTABLES

sprcomp	Sprocket compiler
spanner.pl	Spanner encoder
anepd	ANEP Daemon
injector	Injectors Smart Packets object code into the network
spannervm	Runs a Spanner program through the Virtual Machine without first sending it on the network
bardemo	A demo user program

4. USE

Sprocket programs must be compiled into Spanner using sprcomp. Spanner programs must be assembled into wire encoding using spanner.pl. The resultant .o file from spanner.pl can be run in standalone mode with spannervm, or can be injected into the network with the injector.

When a program is injected into the network, for each host that receives it, the host must have an ANEP daemon running.If a program is sent along a multi-hop route, and it is being sent in hop-by-hop mode (see injector man page), then the intermediary routers must be running an ANEPD daemon, and must have a kernel installed which has the Smart Packets router alert and ANEP modifications (see doc/kernel-mods.txt).

5. DOCUMENTATION

README	This file
INSTALL	Generic autoconf installation instructions
doc/RELEASE_NOTES-1.0.0	Release notes
doc/anep.rfc	RFC for Active Networks Encapsulation Protocol
doc/encoding.txt	Technical details about how Spanner is encoded in bits and bytes (a companion document is encoding.gen which is generated when the distribution is made; this shows every opcode bit for every Spanner operation.)
doc/kernel-mods.txt	Instructions on how to apply ANEP and router alert kernel modifications
doc/lang_survey.txt	Survey of potential Smart Packets languages
doc/primitive.txt	Description of all Sprocket primitives, by functional grouping
doc/sec_arch.txt	Security architecture overview
doc/smart.ps	Conference paper on Smart Packets
doc/spanner.txt	Spanner assembly language overview
doc/spkthdr.txt	Smart Packets header layout

```
doc/sprocket.txt  Sprocket high-level language overview
man pages for all executables listed above
```

APPENDIX F. REGISTERED NODES

Name: Livio Ricciulli

IP Address: d00.csl.sri.com

Public_Key:udoN0w7B0K65hhwpwgpOzp/Pj//aYUTfEo2N4s8bW2kjs3rVtbfohk
0UJA6cvcbLXZOfGJybjgnaIj6G2NPtvQEAAQ==

Organization: SRI International

Organization_Address: 333 Ravenswood Ave, Menlo park, CA 94025

Phone_Number: 650-859-2969

Email: livio@csl.sri.com

Misc_Info: Network Engineering experiments

Name: Livio Ricciulli

IP Address: capri.sri.brainstorm.net

Public_Key:umVy3uv1LpaSx7W83haqRoNVrEH/cNatGaa7B4YgQYRl4K9qPzrBpd
oVk7rKVTDyX +OpUc2B6aNepLsyD/FjWQEAAQ==

Organization: SRI International

Organization_Address: 333 Ravenswood Ave

Phone_Number: 650-859-2969

Email: livio@csl.sri.com

Misc_Info: Anetd's experiments from home

Name: Pankaj Kakkar

IP Address: zaria.csl.sri.com

Public_Key:r8K+gZ4ZRo5usA6751RD5Kh7HNwGAu10HvuxGE5v5epOFcW0EgkLcT
sBq7fjDWkW6EU57oNG6F6e451rUln2oQEAAQ==

Organization: University of Pennsylvania

Organization_Address: 200 South 33rd St, Philadelphia PA 19104

Phone_Number: (215) 898 8116

Email: pankaj@gradient.cis.upenn.edu

Misc_Info: PLAN daemon admin

Name: Madhu Sudan

IP Address: 130.107.16.135

Public_Key:xc5Z6TVuR26Y7HtiAQ3Y7hXjzZwss6g+z2KignNBr1Da2YWo3KFVFs
gUwfksUlPH27IDLeU7ioqV5wmNhfwJ8QEAAQ==
Organization: SRI International
Organization_Address: 333 Ravenswood Avenue, Menlo Park, CA
94043, USA
Phone_Number: (650) 859 4247
Email: madhu@csl.sri.com
Misc_Info: Experimentation and administration

Name: Maria Calderon Pastor
IP Address: 138.100.10.152
Public_Key:xcNrTqbIK3oPl/k9ovUeWhfoNatzT5t7g0lhDguI+VC9Ef0GUHDwUt
ym3LqYCZudPlgNrRtxTGMwSoHyJzJ8wwEAAQ==
Organization: Facultad de Informatica - UPM
Organization_Address: Campus de Montegancedo - Boadilla del Monte
- 28660 Madrid - Spain
Phone_Number: +34 1 3367396
Email: mcalderon@fi.upm.es
Misc_Info: Active Network Deployment

Name: Jeff Kann
IP Address: 128.9.160.165
Public_Key:sPEY1JLWL+Ovft6+JEPbi7tfSnX3PfCI68PqAKwiWrGWfqKl8UsZHR
D1N4G194fUa5/tXH1lW2HM7Pv93zEoRwEAAQ==
Organization: USC/ISI
Organization_Address: 4676 Admiralty Way, Marina del Rey, CA
90292
Phone_Number: 310-822-1511
Email: kann@isi.edu
Misc_Info: ARP project experiment

Name: dartisi
IP Address: 128.9.160.194
Public_Key:zQYprXgdnYjqkehgKqAh6RgEOOM15PJjp/EDCq/kBzl3F+B5lLhXmp
foox4SKukyBNP6sv4PdPiLqIEy4xPkLQEAAQ==

Organization: USC/ISI
Organization_Address: 4676 Admiralty Way, Marina del Rey, CA 90292
Phone_Number: 310-822-1511
Email: kann@isi.edu
Misc_Info: Providing one login for all the people in ISI West that could use the Abone for the ARP project.

Name: Dana Chee
IP Address: 207.3.230.162
Public_Key:pOP54oiR+Wvi/iKQzcAfxy2kazJWYFdAOkUx96WDmS3trnaPMlrn8G kYBDwSR8DX8YhJdNMTDzVzbG/gGUXurQEAAQ==
Organization: Bellcore
Organization_Address: 445 South Street; Morristown, NJ 07960
Phone_Number: 973-829-4488
Email: dana@bellcore.com
Misc_Info: I'm currently working on the Active

Name: Kristin Wright
IP Address: 155.99.212.119
Public_Key:ysmUPZAyvTZOC8EygGpv5jQqVWth644B3bG+zhQXnDYuXk2dT2kOnZ auqyNVJjeupOFaGeYl5IeUnNi/ClzpSQEAAQ==
Organization: University of Utah
Organization_Address: 50 S Central Campus Drive, Rm 3190
Phone_Number: 801-581-4802
Email: kwright@cs.utah.edu
Misc_Info: Active network research

Name: Dan Van Hook
IP Address: 129.55.10.190
Public_Key:+KHXfCL/OdjIcbNoOVh3mUY8vD1XXpO3kronjWwKUZu/vJh1N+vO3U npn/mYcOO8sMUE417dlMGakWdZrSlLeQEAAQ==
Organization: MIT Lincoln Laboratory - Distributed Systems Group
Organization_Address: 244 Wood Street Lexington MA 02420
Phone_Number: 781-981-4153

Email: dvanhook@ll.mit.edu
Misc_Info: Active network research

Name: Edward Lewis
IP Address: 199.171.39.3
Public_Key:5hTIUzwLCD3WtRK1flkUZssrTvfcW99oKPFmv9+CkDhcAZUjPAk+UI
xiHgOCe9/2moYQ5foAhSGkXeFnl12zCQEAAQ==
Organization: TIS Labs at NAI
Organization_Address: 3060 Washington Rd, Glenwood, MD, 21738
Phone_Number: +1 301-854-5794
Email: lewis@tis.com
Misc_Info: DARPA research project

Name: Patrick Jie
IP Address: 166.104.36.173
Public_Key: 4jhjie
Organization: Network Computing Lab.
Organization_Address: Hanyang University, Korea
Phone_Number: 02-290-0355
Email: jhjie@hyuee.hanyang.ac.kr
Misc_Info: For testing the Active Networking Technology and
developing the new revolutionary Application.

Name: Patrick Jie
IP Address: 166.104.45.177
Public_Key:m/aOb8BxyxbiknsOAjEMKHXlOdhWr4c39FDDrzXDyiOL7wqpZiXDKZ
NEUCfViMvvBmO0OQCLDcrRv4gldXRnEwEAAQ==
Organization: Network Computing Lab.
Organization_Address: Hanyang University, Korea
Phone_Number: 082-02-290-0355
Email: jhjie@hyuee.hanyang.ac.kr
Misc_Info: Testing the active networking technology and
developing the revolutionary application.

Name: Patrick Jie
IP Address: 166.104.45.194
Public_Key:rZLD5aZb+8iyy7iGtRQVrugl3diOn42lPOqMwuTeQ09t05MsulQzk1
7c58MKWHRyKFzeJOyCCd6kOuVSXeboQwEAAQ==
Organization: Network Computing Lab.
Organization_Address: Hanyang University, Korea
Phone_Number: 082-02-290-0355
Email: jhjie@hyuee.hanyang.ac.kr
Misc_Info: For testing the dynamic network operation and the
revolutionary application

Name: David Raila
IP Address: 128.174.240.14
Public_Key:QCNAy8b62UAAAEEAMbYY9kAyOHAFb9cblO7QiACmFdvcy3WjNZNc/m
Rrk9Qcp0v
Organization: Univ Ill. CS Dept.
Organization_Address: 1304 West Springfield, Urbana Il, 61801
Phone_Number: 217 333 0108
Email: raila@cs.uiuc.edu
Misc_Info: research

Name: Pankaj Kakkar
IP Address: 158.130.12.150
Public_Key:1U0URh6BazDPNk2A6wMJ783IPE7rlGAyx0ARW3PKKo4rJy5Z3ppMg4
rC/Um6YN9qw9fm7JPLQo9LGMEywM+QawEAAQ==
Organization: Univ of Penn
Organization_Address: 200 St. 33rd St
Phone_Number: 215 898 8116
Email: pankaj@gradient.cis.upenn.edu
Misc_Info: UPenn ABONE node admin

Name: Geoffrey XIE
IP Address: 131.120.1.244
Public_Key:t2pdU7tWzikq3cklFxyYpe6xDLqd4mdTiDp7cpIjGGnGHlsc+w3miV
KE88rfpa39DImeIELTDHWFfIxVg0BOrQEAAQ==

Organization: Naval Post Graduate School

Organization_Address: NPS Monterey CA 93943

Phone_Number: (831)656-2693

Email: xie@cs.nps.navy.mil

Misc_Info: Our research is about Active Networking

Name: Cheryl DeMatteis

IP Address: 206.117.53.41

Public_Key:+k8dGlDPWGetFepCfafJZouZdTTTw1Xz8lfLEcLHx8yhXXWw8z1S81 yWuPfP+MIILwE30E0B0t1Va/Hz FjBU2wEAAQ==

Organization: The Aerospace Corporation

Organization_Address: 2350 E. El Segundo Blvd., El Segundo CA. 90245

Phone_Number: (310)336-1189

Email: cdematt@aero.org

Misc_Info: To participate in Active Networks research

APPENDIX G. LOG FOR ANTS EXAMPLE

```
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 sequoia.csl.sri.com
sc.Linux QUERY 3322 d00.csl.sri.com
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
```

```
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
```

```
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 sequoia.csl.sri.com
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
```

```
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000 E=DISPLAY::0.0
C=Standard_output_viewer
sc.Linux QUERY 3322 sequoia.csl.sri.com
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
```

```
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
```

```
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
```

```
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux QUERY 3322 d01.csl.sri.com
sc.Linux QUERY 3322 sequoia.csl.sri.com
```

```
sc.Linux QUERY 3322 melon.cs.nps.navy.mil
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
```

```
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
```

```
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
```

```
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux KILL 3322 d01.csl.sri.com 0
```

```
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux PUT 3322 d01.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 d01.csl.sri.com data.config data.config
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.1 T=18
O=18.31.12.1 C=ANTS_active_node
sc.Linux LOAD 3322 d01.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.1
S=18.31.12.1 S=18.31.12.1 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 sequoia.csl.sri.com data.routes data.routes
sc.Linux PUT 3322 sequoia.csl.sri.com data.config data.config
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.2 T=18
O=18.31.12.2 C=ANTS_active_node
sc.Linux LOAD 3322 sequoia.csl.sri.com
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.2
S=18.31.12.2 S=18.31.12.2 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.routes data.routes
sc.Linux PUT 3322 melon.cs.nps.navy.mil data.config data.config
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/ants-
1.2.a/~ants/ConfigurationManager S=data.config S=18.31.12.3 T=18
O=18.31.12.3 C=ANTS_active_node
sc.Linux LOAD 3322 melon.cs.nps.navy.mil
J=http://sequoia.csl.sri.com:7000/java/~GetStat1 S=18.31.12.3
S=18.31.12.3 S=18.31.12.3 S=5000
E=DISPLAY:melon.cs.nps.navy.mil:0.0 C=Standard_output_viewer
```

```
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 d01.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 sequoia.csl.sri.com 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
sc.Linux KILL 3322 melon.cs.nps.navy.mil 0
```

APPENDIX H. OUTPUTS OF THE PLAN ACTIVE ROUTER

A. PORT 3324, MAIN ROUTER

```
25-Jan-99 7:29:37 PM: ARMain: logfile set to m24
25-Jan-99 7:29:37 PM: ActiveRouter.start: Active router up!
25-Jan-99 7:29:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : fstCookie
25-Jan-99 7:29:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:37 PM: SLRPmaster: received a request to add:
IPv4UDP : (melon.cs.nps.navy.mil/131.120.1.244, 3324)
25-Jan-99 7:29:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : newRT
25-Jan-99 7:29:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): fstCookie
25-Jan-99 7:29:37 PM: ActiveRouter: IN from IPv4UDP : (
melon.cs.nps.navy.mil/131.120.1.244, 3324): newRT
25-Jan-99 7:29:37 PM: SLRP: received new route table.
25-Jan-99 7:29:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): addme
25-Jan-99 7:29:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : fstCookie
25-Jan-99 7:29:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:47 PM: SLRPmaster: received a request to add:
IPv4UDP : (melon.cs.nps.navy.mil/131.120.1.244, 3326)
25-Jan-99 7:29:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : newRT
25-Jan-99 7:29:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : newRT
25-Jan-99 7:29:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): newRT
25-Jan-99 7:29:47 PM: SLRP: received new route table.
25-Jan-99 7:29:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:29:47 PM: ActiveRouter: OUT: succeeded
```

25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): addme
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : fstCookie
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:57 PM: SLRPmaster: **received a request to add:**
IPv4UDP **: (melon.cs.nps.navy.mil/131.120.1.244, 3325)**
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : newRT
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : newRT
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : newRT
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): newRT
25-Jan-99 7:29:57 PM: SLRP: received new route table.
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT: succeeded

```
25-Jan-99 7:30:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:31:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:28 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:31:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:38 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324): livereport
25-Jan-99 7:31:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:47 PM: ActiveRouter: OUT: succeeded
```

B. PORT 3325

```
25-Jan-99 7:29:57 PM: ARMain: logfile set to m25
25-Jan-99 7:29:57 PM: ARMain: using incoming port 3325
25-Jan-99 7:29:57 PM: ARMain: master set to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324)
25-Jan-99 7:29:57 PM: ActiveRouter.start: Active router up!
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : addme
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): fstCookie
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): newRT
25-Jan-99 7:29:57 PM: SLRP: received new route table.
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT: succeeded
```

```
25-Jan-99 7:30:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:30:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
```

```
25-Jan-99 7:30:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:31:18 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:28 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326) : livereport
25-Jan-99 7:31:28 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:28 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : livereport
25-Jan-99 7:31:28 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
25-Jan-99 7:31:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325): livereport
```

C. PORT 3326

```
25-Jan-99 7:29:47 PM: ARMain: logfile set to m26
25-Jan-99 7:29:47 PM: ARMain: master set to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324)
25-Jan-99 7:29:47 PM: ActiveRouter.start: Active router up!
25-Jan-99 7:29:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3324) : addme
25-Jan-99 7:29:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:29:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): fstCookie
25-Jan-99 7:29:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): newRT
25-Jan-99 7:29:47 PM: SLRP: received new route table.
25-Jan-99 7:29:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:29:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): newRT
25-Jan-99 7:29:57 PM: SLRP: received new route table.
25-Jan-99 7:29:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:29:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:27 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
```

```
25-Jan-99 7:30:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:37 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:47 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:30:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:30:57 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:07 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:17 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:28 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:38 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:47 PM: ActiveRouter: OUT: succeeded
```

```
25-Jan-99 7:31:48 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:31:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:31:57 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:31:58 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:07 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:07 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:32:08 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:17 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:17 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:32:18 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:27 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:27 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:32:28 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:37 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:37 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:32:38 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:47 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:47 PM: ActiveRouter: OUT: succeeded
25-Jan-99 7:32:48 PM: ActiveRouter: IN from IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3326): livereport
25-Jan-99 7:32:57 PM: ActiveRouter: OUT to IPv4UDP :
(melon.cs.nps.navy.mil/131.120.1.244, 3325) : livereport
25-Jan-99 7:32:57 PM: ActiveRouter: OUT: succeeded
```

INITIAL DISTRIBUTION LIST

1. **DEFENSE TECHNICAL INFORMATION CENTER** 2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. **DUDLEY KNOX LIBRARY** 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. **DENIZ KUVVETLERI KOMUTANLIGI** 2
   Personel Daire Baskanligi
   Bakanliklar
   Ankara, TURKEY

4. **DENIZ HARP OKULU KOMUTANLIGI** 1
   Kutuphane
   Tuzla, Istanbul- 81704, TURKEY

5. **CHAIRMAN, CODE CS** 1
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. **PROFESSOR GEOFFREY XIE, CODE CS/XG** 1
   Naval Postgraduate School
   Monterey, CA 93943-5100

7. **LCDR CHRIS EAGLE, CODE CS/EC** 1
   Naval Postgraduate School
   Monterey, CA 93943-5100

8. **LTJG NAMIK KAPLAN** 2
   Yakacik Mahallesi Sedefciler Sokak
   Senlikkoy Sitesi B-Blok N0:66
   Kecioren
   Ankara, TURKEY