NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MAGMA©: A LIQUID SOFTWARE APPROACH TO FAULT TOLERANCE, COMPUTER NETWORK SECURITY, AND SURVIVABLE NETWORKING

by

Scott Margulis

December 2001

Thesis Advisor:	Geoffrey Xie
Second Reader:	Daniel Warren

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Title (Mix case letters) MAGMA©: A Liquid Software Approach to Fault Tolerance, Computer Network Security, and Survivable Networking.	**5. FUNDING NUMBERS**
6. AUTHOR(S) Margulis, Scott A.	
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) SSC-SD	**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)*

Next Generation Internet (NGI) will address increased multi-media service demands, requiring consistent Quality of Service (QoS), similar to the legacy phone system. Server Agent-based Active network Management (SAAM) acts like rush-hour traffic reporting helicopters. Upon routing request arrivals, SAAM server determines the best route and assembles the routing path, freeing up routers to provide faster, more reliable, forwarding services. SAAM server is a critical network node; therefore, it must be extremely robust. With Margulis Agent-Based Mobile Application (MAGMA©) liquid software, a SAAM server agent will remain inactive in resident memory of each router until it is stimulated by a message from the departing server. Then that agent will begin running a new server at a starting point determined from the prior server's recent state information or a pre-determined point if that state information is not available. MAGMA© will provide SAAM increased fault tolerance and security against malicious attacks. In this thesis, the foundation for a mobile SAAM server was developed along with a protocol that extracts critical state information from the current server and periodically transports a comporessed form of the information to potential next SAAM servers. MAGMA© will provide a revolution in today's computer fault tolerance and security paradigms, benefiting industry through more survivable networks with guaranteed QoS.

14. SUBJECT TERMS MAGMA©, Liquid Software, Mobile Code, Fault Tolerance, Computer Network Security, Survivable Networking, Agent-based Software	**15. NUMBER OF PAGES** 154
	16. PRICE CODE

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

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

MAGMA©: A LIQUID SOFTWARE APPROACH TO FAULT TOLERANCE, COMPUTER NETWORK SECURITY, AND SURVIVABLE NETWORKING

Scott A. Margulis
Lieutenant Commander, United States Navy
B.S., University of Florida, 1988
MBA, Webster University, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2001

Author: [illegible]
Scott A. Margulis

Approved by: [illegible]
Geoffrey Xie, Thesis Advisor

[illegible]
Daniel Warren, Second Reader

[illegible]
Chris Eagle, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Next Generation Internet (NGI) will address increased multi-media Internet service demands, requiring consistent Quality of Service (QoS), similar to the legacy phone system. Server Agent-based Active network Management (SAAM) acts like a rush-hour traffic reporting helicopter. Upon routing request arrivals, SAAM server determines the best, least traffic/resistance route and assembles the routing path, freeing up "light-weight" routers to provide faster, more reliable, forwarding services.

The SAAM server is a critical network node; therefore, it is imperative to make it extremely robust. With Margulis Agent-Based Mobile Application (MAGMA©) liquid software, a SAAM server agent will remain inactive in resident memory of each router until it is stimulated by a message from the departing server. Then the agent will begin running a new server at a starting point determined from the prior server's recent state information or a pre-determined point if that state information is not available. MAGMA© will provide SAAM an increased fault tolerance and security against malicious attacks.

Liquid software research has taken place since 1996 (University of Arizona/University of Pennsylvania); however, there is no known application currently providing fault tolerance and system security. In this thesis, the foundation for a mobile SAAM server was developed, with the researcher being able to manually move the server from one host to the next. Furthermore, this thesis designed a protocol thatcompresses critical state information, providing condensed messages to efficiently configure the next SAAM server across the network with the state information from the departing server extracts critical state information from the current server and periodically transports a compressed form of the state information to potential next SAAM servers in the network. MAGMA© will provide a revolution in today's computer fault tolerance and security paradigms, benefiting industry through more survivable networks with guaranteed QoS.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge the Space and Naval Warfare Systems Center- San Diego (SSC-SD) for selecting me for their fellowship and sponsoring this thesis research. Also, my personal thanks to Dr. Stephan Lapic for being my sponsor at SSC-SD.

I would also like to extend my gratitude to Capt Vic See, LtCol Mike Folsom, and LT Tom Newman for their help and interest in this thesis project.

I would also like to extend my most sincere gratitude to Professor Geoffrey Xie and Mr. Daniel Warren for their patience and sagely advice with my thesis. Without their countless hours of guidance and editorial comments, this thesis project would have not been completed on time.

Furthermore, many thanks to Cary Colwell for his countless hours clarifying SAAM's implementation and architecture, and design, which ultimately saved numerous hours of decoding and understanding SAAM and Jane Wu for her contributions with the SAAM and MAGMA GUI improvements.

Also, thanks to LT's Bill Wilkins, US Navy and George Stavritis, Hellenic Navy for help in design of the Digest PIB and work with Java compression techniques.

Additionally, I would like to thank my parents, Frank and Rosalyn Margulis for the sagely advise throughout my life. Without their love and guidance, I would not be where I am today.

Mostly, I would like to thank my wife, Lisa, my son, Brandon, and my daughter, Ashley for their unwavering support, love, friendship, and understanding throughout my many long hours spent "away from home" while working on my Masters Degree and Thesis work. Without their unselfish love and support, I would have never been able to complete this journey.

I would like to dedicate this work to my friend LT Thomas Wilcox, to the men and women of the US military who lost their lives at the September 11th attack on the

Pentagon, and to the many other men and women of the United States Armed Forces who have given the “Ultimate Sacrifice” for their country and whose memories will never be forgotten.

EXECUTIVE SUMMARY

Today's Internet protocol, version 4 (IPv4), acts similar to the postal service. Delivery of a packet is not guaranteed, but there is a promise to make a best effort to get it to the addressee. The Next Generation Internet (NGI) is currently being developed to address an increasing demand of multi-media services over the Internet, which requires a consistent Quality of Service (QoS). One proposed solution to this need for a consistent QoS, similar to the legacy phone system, is Server Agent-based Active network Management (SAAM).

SAAM is sponsored by Defense Advanced Research Projects Agency (DARPA), SPAWAR System Center- San Diego (SSC-SD), and the National Aeronautics and Space Administration (NASA). The key component of the SAAM system is a logical software-based server that acts like a helicopter reporting on rush-hour traffic. When a routing request comes in from a host or router, the SAAM server (like the helicopter) looks over the network, determines the best route with the least traffic/resistance, and assembles the routing path. Since the SAAM server makes the routing decisions, the "light-weight" routers are freed-up to provide faster, more reliable, forwarding services.

The SAAM server is such a critical component of this system that it is imperative to the network that the server remains operative. Therefore, there is a need to provide the most robust server possible. Magulis AGent-based Mobile Application (MAGMA$^{©}$), a liquid software like programming method, uses a programming concept similar to mobile agents. The basic idea behind MAGMA$^{©}$ is to create an agent-based system, with one instance of the same server agent installed in the resident memory of each router. This agent remains inactive until it is stimulated by a message from the network administrator or the departing server. Upon activation, the agent begins running a new SAAM server on its host.

The starting point of this new server is determined from its prior server's recent state information or a pre-determined starting point if that state information has been lost.

MAGMA© will provide SAAM with an increased fault tolerance as well as security against malicious attacks. Like a shell game, the MAGMA© enhanced server maneuvers between hosts in a random walk, making it harder for malicious hackers to find their quarry and complete their attack. If the hacker should get lucky or if the server's host is shut down due to mechanical or maintenance purpose, a server agent will stand up a new server on another host and provide the necessary fault tolerance.

Research in the field of liquid software and mobile code has been conducted from 1996 to the present at the University of Arizona and the University of Pennsylvania. However, to date, there is no known application, which uses this concept as a method for providing fault tolerance and system security. In this thesis, the foundation for a mobile SAAM server was developed, with the researcher being able to manually move the server from one host to the next. Furthermore, this thesis designed a protocol that compresses critical state information, providing condensed messages to efficiently configure the next SAAM server across the network with state information from the departing server.

A liquid software based SAAM server will provide a revolution in today's computer security and fault tolerance paradigms. It will benefit industry through a more robust Integrated Services network with guaranteed QoS. The military's conundrum for information and knowledge superiority will be assisted through the liquid software's approach to reconstitution of information networks.

I. INTRODUCTION AND OVERVIEW

A. BACKGROUND

Today’s Internet protocol, version 4 (IPv4), acts similar to the postal service. Delivery of a packet is not guaranteed, but there is a promise to make a best effort to get it to the addressee. The Next Generation Internet (NGI) is currently being developed to address an increasing demand of multi-media services over the Internet, which requires a consistent Quality of Service (QoS). One proposed solution to this need for a consistent QoS, similar to the legacy phone system, is Server Agent-based Active network Management (SAAM).

SAAM is sponsored by SPAWAR Systems Center-San Diego (SSC-SD), Defense Advanced Research Projects Agency (DARPA) and the National Aeronautics and Space Administration (NASA). SAAM acts like a helicopter reporting on rush-hour traffic. When a routing request comes in from a computer, the SAAM server (like the helicopter) looks over the network, determines the best route with the least traffic/resistance, and assembles the routing path. Since the SAAM server takes the routing decisions away from the routers, the “light-weight” routers are freed-up to provide faster, more reliable, forwarding services.

The SAAM server is such a critical component of this system that it is imperative to the network that the server remains operative. Therefore, there is a need to build an extremely robust server. Liquid software is a programming concept similar to mobile agents. Basically, the idea is to create a SAAM server agent which will reside in the memory of each router. This agent will remain inactive until it is stimulated by a message from the departing server. Upon activation, the agent will begin running on the new host. The starting point of this application will be determined from its prior server agent’s state information or a pre-determined starting point if the prior server agent’s state information has been lost. Liquid Software will provide SAAM with an increase fault tolerance as well as security against malicious attacks. Research in the field of liquid software and mobile code has taken place in 1996 to present at the University of Arizona and the University of Pennsylvania. However, to date, there is no known application, which uses this concept as a method for providing fault tolerance and system security.

A liquid software based SAAM server will provide a revolution in today's computer security and fault tolerance paradigms. It will benefit industry through a more robust Integrated Services network with guaranteed QoS. The military's conundrum for information and knowledge superiority will be assisted through the Liquid software's approach to reconstitution of information networks.

B. APPROACH

SAAM is an ongoing IPv6-based prototype research project of Professor Geoffrey Xie, which includes the work of over 20 different student theses. A full time programmer, Mr. Cary Colwell, also supports the project. Since its inception in May 1998, SAAM has evolved into its current baseline with the capability to handle up to 40 light-weight routers, utilizing a primary and back-up server.

When the author first looked at the SAAM project and the possibility of contributing to its effort, one distinct area warranted further improvement. There was a critical node in the system, which could bring the entire SAAM Region to a standstill. The threats to this critical node include malicious means or physical problems. This critical node was the SAAM server.

Under its current configuration, the SAAM server has a built-in fault tolerance mechanism, which was designed by Efraim Kati in March 2000. Kati's thesis, entitled "Fault-Tolerant Approach for Deploying Server Agent-based Active network Management (SAAM) Server in Windows NT Environment to Provide Uninterrupted Services to Routers in Case of Server Failure(s)," examines building fault tolerance into the SAAM prototype by introducing a backup SAAM server design. This backup server provides "robust error detection and a fast recovery from failure of the primary SAAM server" through the means of a new protocol called "Heartbeat Query." Though this did build in some strength to the system, it did not address the weakness from malicious attack or multiple host failures.

In order to improve SAAM's fault tolerant posture and add a degree of security to this system, MAGMA©, which stands for **M**argulis **AG**ent-based **M**obile **A**pplications, will be added to the SAAM project. Whereas MAGMA© is similar to past SAAM related theses in that it uses a Mobile Agent code concept, it is unique from any other projects and from past theses by virtue of its liquid software characteristics. The term "Liquid Software" entails maintaining current state information of the SAAM Region architecture/traffic flows; while being able to seamlessly move the server's functionality and state information to another host with virtually no degradation of system performance.

C. SCOPE

The thesis work done by Efraim Kati brought limited fault tolerance to the SAAM topology, but did not fully address this issue. Also, there are no current mechanisms in the SAAM topology to address security related concerns. It has been determined that the SAAM Server is the critical node of this network, providing a vital service to the entire network under its control. Therefore, it is imperative that this critical node be protected in all circumstances. It is this thesis's intent to design in fault tolerance and system security through a technique known as liquid software, which utilizes mobile agents. In order to do this, code will be written which creates a MAGMA© agent. This agent will reside in a dormant state on each router of a SAAM Region. The agent and follow-on updates may be uploaded to a router at run time via SAAM's extensible service router architecture. The server's dynamic state information will be stored in a smaller state table than the current Base Path Information Base (Base PIB) and will be compressed to provide the smallest possible footprint when sent over the network. This compressed "Digest PIB" will be distributed to all designated SAAM routers so that they have some knowledge of the network when they need to take over the job of server. Code will also be written for a GUI to control the MAGMA© agents and a protocol to maneuver the new mobile MAGMA© server around the SAAM Region randomly.

D. ORGANIZATION OF THIS THESIS

Chapter II: Background. Cover the history of computer networking from its genesis to present. Discuss the development of SAAM and its current state of fault tolerance. Explain all pertinent known work performed with liquid software and mobile agents.

Chapter III: Fault Tolerance, Network Security and Related Issues. Explores basic fault tolerant/computer security definitions and techniques. Areas covered include redundancy and fault tolerant concepts, Denial of Service (DoS) attacks such as SYN attacks and Smurf attacks, and IP spoofing.

Chapter IV: MAGMA©'s Design of Security, Fault Tolerance, and Survivable Networking. Discusses how MAGMA© was developed and integrated into the SAAM topology. Talks about the design and development of MAGMA©. Looks at experimental results from the research.

Chapter V: Digest PIB And Data Compression for Reducing Overhead. Looks at the current Path Information Base (PIB) (State Information) and how it is condensed to the Digest PIB, which is more manageable for transport over the network. Provides a look at Java compression techniques and describe which technique MAGMA© is using. Concludes with compression/Digest PIB experimental results.

Chapter VI: Conclusions and Recommended Future work. Summarizes this thesis and suggests areas for future work in MAGMA© and SAAM.

Appendices. MAGMA© Code.

List of References

Distribution List

II. BACKGROUND

A. BRIEF HISTORY OF COMPUTER NETWORKING

The first published material discussing links to the modern day Internet was one on packet-switching techniques written in 1961 by Leonard Kleinrock, an MIT graduate student. Six years later, colleagues of Kleinrock from MIT, J.C.R. Licklider and Lawrence Roberts, published overall plans for a computer network, which led to the foundation of Advanced Research Projects Agency Network (ARPANet). The ARPANet was the first packet switching computer network. Its first node was installed at UCLA in September 1969 with three more installed later that same year at Stanford Research Institute (SRI), UC Santa Barbara, and University of Utah. By 1972, the ARPANet had grown to 15 nodes. In the mid 1970's, other networks based on the ARPANet's network-control protocol (NCP) were springing up including the ALOHANet, Telenet, Tymnet, and Transpac. The next major advance in networking was achieved in 1976 by Metcalfe and Boggs, who designed Ethernet, a baseband mode local area network, at the Xerox Palo Alto Research Center.

In the early 1970's, there were approximately 200 nodes connected to the ARPANet. By the end of the 1980's, nearly 100,000 nodes were connected as the modern day Internet started taking shape. In 1983, TCP/IP officially became the standard networking protocol of the ARPANet, replacing the legacy NCP. The World Wide Web was conceived sometime between 1989 and 1991 at CERN by Berners-Lee; though in 2000, ex-Vice-President Al Gore would have told you that he is the father of today's Internet (SNL, October 2000). Berners-Lee based the web on ideas stemming from discussions on hypertext in the 1940's by Bush and in the 1960's by Nelson. By 1992, there were some 200 Web servers.

B. RESEARCH IN MOBILE ENTITIES, AGENT-BASED CODE, AND LIQUID SOFTWARE

The goal of using mobile entities, agent-based code and liquid software is to enable distributed software systems with critical nodes to recover from failure of the critical nodes. The functionality of the critical nodes needs to be able to reside on multiple hosts in the network. In this approach, the critical service of a system is provided by one or more agents that moves around the system to maintain a stealthy appearance and remain incognito to an intrusive, malicious entity. In order to maneuver around the system without tying up valuable bandwidth, each agent has to maintain the smallest footprint possible.

The following sections describe past work in distributed systems: mobile operating systems, mobile applications, and mobile code.

1. Related Work

Mobile agents, mobile code, distributed operating systems have been around for several decades in one form or another. In its infancy, the Internet was chiefly a vehicle to distribute text and computer programs. As the Internet concept matured, its use as a means to provide a more fluid software environment took hold and evolved. These fluid software environments can be categorized into three distinct areas: 1) distributed programming libraries and packages, 2) distributed operating systems, and 3) distributed programming languages. [WUD98] More detail of each area will be covered in the following paragraphs.

2. Distributed Programming Systems

Of all the aforementioned fluid software environments, distributed programming systems have probably been around the longest. This is due to fact that they are the simplest form of distributed programming support. Some examples of distributed programming systems include RPC[1984], DCE[1992], CORBA[1996], DCOM[1997], RMI[1996], and HORB[1996]. While RMI and HORB are Java-specific, RPC, DCE, CORBA, and DCOM are not language, operating system, or hardware specific. [WUD98] Each of these systems have their respective advantages and disadvantages. Their advantages lie mainly in the fact that they require minimal effort to infuse their strategies into the existing software systems. Their disadvantages are found in their "limited form of interoperability and object migration that they support, although CORBA, DCE, and DCOM seek to ameliorate this by providing a rich set of standard services for creating, locating, registering, and storing an object." [WUD98]

3. Distributed Operating Systems

Distributed operating systems bring a different capability to the relatively new Internet environment. They allow the network to provide for more efficient computation through the additional support of fault-tolerance, distributed processes, multi-tasking, and flexibility. Some examples of distributed operating systems include Sprite[1987], V[1988], and Locus[1986]. The main focus of these systems is to "provide OS support to freeze the run-time computation of a process, migrate the process to a remote site, and unfreeze the process." [WUD98] Their contributions to computer science include the ability to track and utilize the resources in a multi-processor environment, the use of memory in multiple locations, the use of a distributed data/file storage system, and the use of the newly conceived concept of fine-grained process mobility.

4. Distributed Languages

While distributed languages are not recent technological development, they have taken shape and played a more productive role in the past several years. Some examples of currently studied/used distributed languages include Obliq[1995], Emerald[1988], and Distributed Oz[1997], and adaptive mobile code languages such as TACOMA[1995], Telescript[1996], Sumatra[1997], Agent TCL[1996], Odyssey[1996], Voyager[1996], and the Liquid Software project at University of Arizona[1996]. The first three are languages whose semantics were designed to support distributed scope and access, to provide a shared memory abstraction, and to extend ordinary language operations to manage replicated objects and data. [WUD98] The latter category comprises agent development languages and mobile code frameworks which provide object mobility within a distributed environment. [WUD98]

One of the most recent advantages in distributed languages is that of a program design known as "Liquid Software" invented at the University of Arizona. This concept explores the possibility to provide computing resources the ability to move throughout a network utilizing mobile agents. Consisting of an array of retargetable compilers, customizable client/server interfaces, and operating system support library, the Liquid Software infrastructure can be used to develop software agents for browsing, searching, and retrieval. [WUD98] This allows for a more robust system.

The MAGMA© software takes a similar approach as Liquid Software in that it uses mobile agents to move its functionality from host to host. In MAGMA©, SAAM server code is distributed to each host and placed in a dormant state. When the MAGMA© enhanced server moves to another host, a MAGMA© message is sent to the new host, which activates the new object functionality. MAGMA© provides SAAM with the ability to provide computation of routing tables, Quality of Service queue manipulation, link quality state monitoring, and flow tracking in a mobile environment.

5. Different Types of Mobile Code

Mobile code paradigms can be broken down into 4 major areas: Client/Server (CS), Remote Evaluation (REV), Code On Demand (COD), and Mobile Agents (MA). To understand Table 1, there is a need to discuss the abbreviations. A and B refer to the computation components of the code, also known as the Thread of Execution. S_A and S_B refer to the two hosting computers. *D* and *C* refer to the *data* and *code* resources, respectfully.

In the CS model, the data, code, and threads of execution remain stationary at each of the hosts with only the data resource from S_A moving to S_B. (See Figure 1.) In REV, the data, code, and computation components remain at both hosts, yet host A sends its data and code to host B for processing and execution. (See Figure 2.) "Although mobile code scenarios CS and REV are very similar in nature, they differ in one key aspect; the ability to transfer processing from one site to another by transferring the executable code." [WUD98] COD is just a reversal of REV, where processing and execution occur at host A. However, in COD, S_A sends its code and data to S_B for execution; where in REV, S_A requests code and data from S_B. (See Figure 3.)

Paradigm	*Before Migration*		*After Migration*	
	S_A	S_B	S_A	S_B
CS	A,D_A,C_A	B,D_B,C_B	A,D_A,C_A	B,D_B,C_B,D_A
REV	A,D_A,C_A	B,D_B,C_B	A,D_A,C_A	B,D_B,C_B,D_A,C_A
COD	A,D_A,C_A	B,D_B,C_B	A,D_A,C_A,C_B,D_B	B,D_B,C_B
MA	A,D_A,C_A	--,D_B,C_B	--,D_A,C_A	A,D_B,C_B,D_A,C_A

Table 1. Mobility and Remote Execution Paradigms[WUD98]

A very close representation of MAGMA© is MA. In MA, a dynamic and autonomous scenario is achieved through the movement of code, data, and computation components from one host to the next. (See Figure 4.) In CS, REV, and COD, the data and code are moved to another host, but the computation thread remains stationary at its original location. Also, S_A initiates the interaction with S_B with the results being achieved under separate threads within the two sites. To provide for execution of MA, processing has to be suspend during the movement and standing up of a new thread at the new host. During transfer of control, state information is maintained for commencement of the new controlling component.

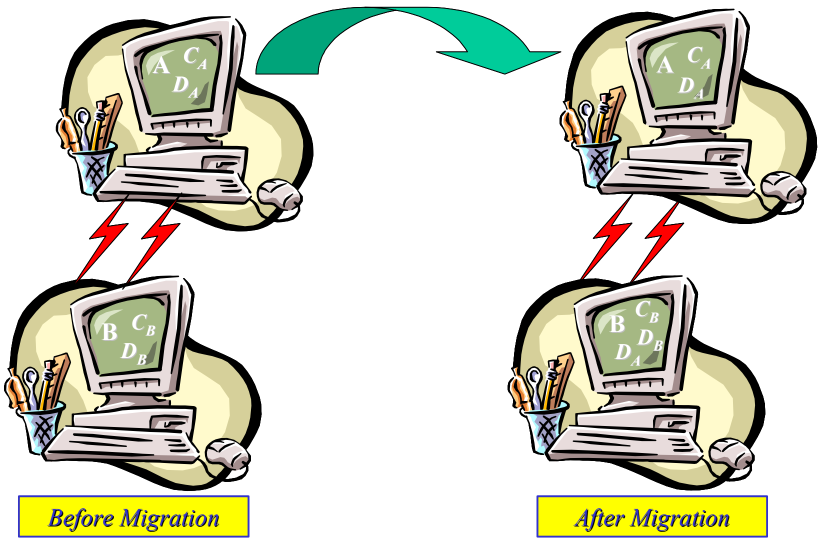


Figure 1. Client/Server Functionality

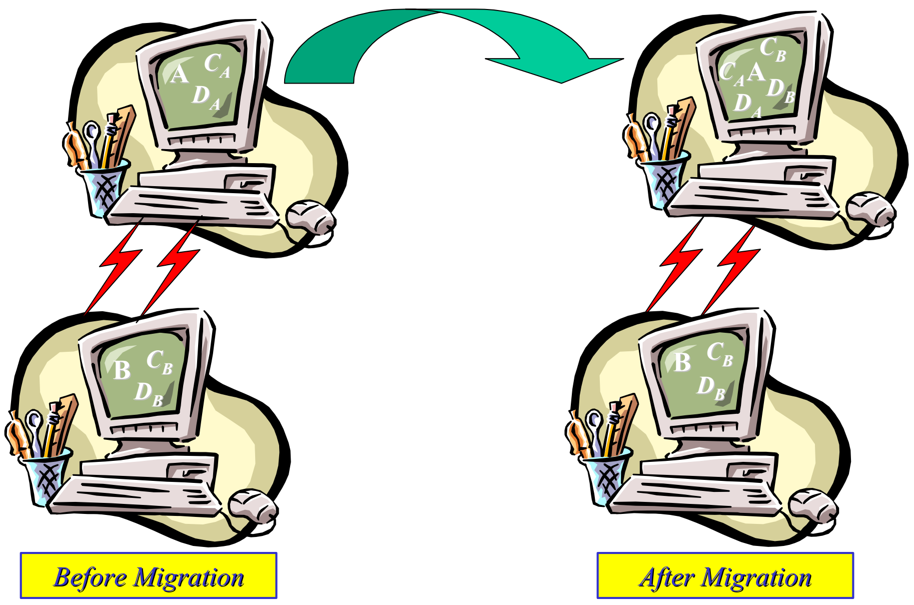


Figure 2. Remote Evaluation Functionality

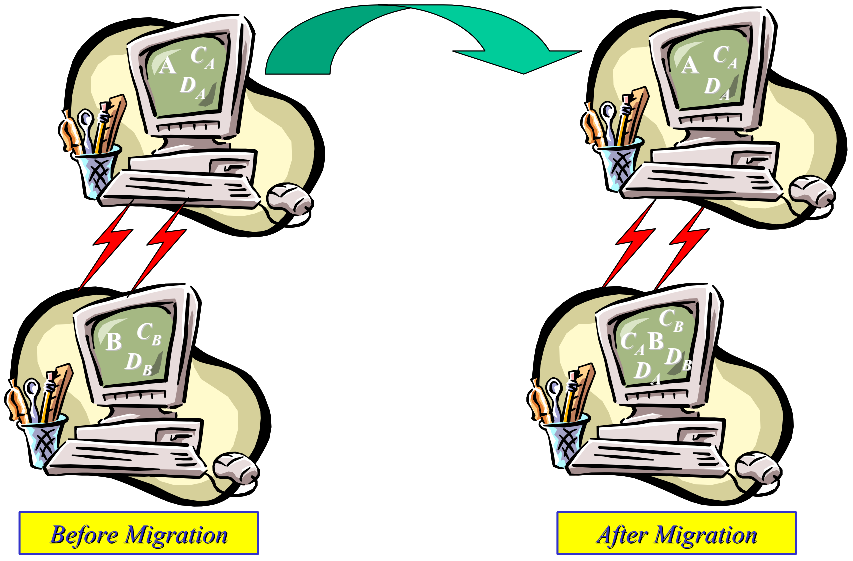


Figure 3. Code On Demand Functionality

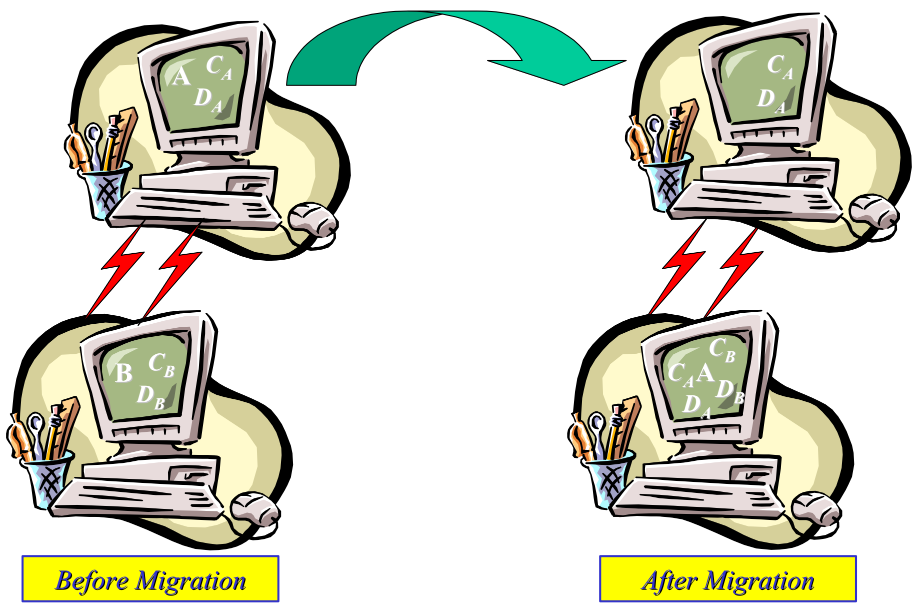


Figure 4. Mobile Agents Functionality

C. HISTORY OF SAAM

SAAM's genesis was the vision of Professor Geoffrey Xie in 1998. The initial goal was to address the Next Generation Internet's needs (also known as Internet Protocol version 6 {IPv6}) and the need to provide guaranteed Quality of Service (QoS). SAAM addresses these needs through the use of a master-slave like hierarchical approach where a SAAM server monitors the overall network and makes all QoS routing decisions while the clients, or SAAM routers, merely forward QoS sensitive packets according to instructions from the server. These routers may handle the best effort packets in the same way as before.

Professor Xie's first thesis students in this topic, Capt Dean Vrable and Capt John Yarger, United States Marine Corps, laid the foundation and built the infrastructure for a prototype SAAM network. This included the communication channels, administrative messaging formats, the host's simulation containers, and crude data structures to maintain the state information necessary to run this new system. Their joint thesis was entitled "The SAAM Architecture: Enabling Integrated Services."

At about the same time as Capt's Vrable and Yarger were laying the ground work for a SAAM prototype, several other students were already taking a more in-depth look at several aspects of the system. They included Major Brian Tiefert, United States Marine

Corps; Captain Fred Ludden, United States Army; Major Katrina Hensley, United States Marine Corps; and Lieutenant Namik Kaplan, Turkish Navy. Maj Tiefert's thesis, "Modeling Control Channel Dynamics of SAAM using NS Network Simulation", helped lay the ground work for the SAAM's administration signaling channels and allowed the SAAM team to develop simulation tools to evaluate the complex system that SAAM was evolving into. Capt's Ludden and Hensley completed a joint thesis, "ATM Security by 'Stargate' Solutions." LT Kaplan's work was entitled "SAAM Active Server Probing using PLAN and ANEP."

Next, Lieutenant Mustafa Altinkaya, Turkish Navy; 1st Lieutenant Hasan Akkoc, Turkish Army; 1st Lieutenant Huseyin Uysal, Turkish Army; 1st Lieutenant Efraim Kati, Turkish Army; and Mr. Henry Quek, Ministry of Defense, Republic of Singapore, turned the prototype into a more resilient system through several enhancements, to include agent deployment, routing protocols to assemble signaling channels, link state information, and Quality of Service management.

LT Kati, added robustness to the system by addressing the need to provide server fault tolerance. His thesis, "Fault-Tolerant Approach for Deploying Server Agent-based Active network Management (SAAM) in Windows NT Environment to Provide Uninterrupted Services to Routers in Case of Server Failure(s)," added a backup server to the SAAM environment, much like the use of a Windows NT Backup Domain Controller (BDC).

This was followed by another joint thesis by Captain Luis Velazquez, United States Marine Corps and Lieutenant Peter Szczepankiewicz, United States Navy. Capt Velazquez and LT Szczepankiewicz's thesis, "Authentication in SAAM Routers", provided SAAM a mechanism to allow for SAAM packet information assurance. This thesis is not integrated into the current SAAM version. Mr. Gary Stone, Ph.D. candidate, National Security Agency, provided a foundation for policy-based precise network management in his dissertation, "Path-based Network Policy Language."

Subsequently, Lieutenant Colonel Kuo Dao-Cheng, Republic of China Army, and Lieutenant Colonel John H. Gibson, US Airforce, addressed the need to maintain the state information in a more organized data structure. They gave the SAAM server some more flexibility in their thesis entitled, “Design of a Dynamic Management Capability for the Server Agent-based Active network Management (SAAM) System to Support Requests for Guaranteed Quality of Service Traffic Routing and Recovery.” They designed a data structure, which they called the Base Path Information Base (Base PIB).

Lieutenant Mohammad Ababneh, Royal Jordanian Air Force, provided the SAAM team a way to vary their SAAM Region’s architecture and to deploy SAAM agents through the use of Extensible Markup Language (XML) files. His thesis was entitled, “(SAAM) Network Configuration Using XML.” Lieutenant Faitah Turksoyu, Turkish Navy, provided a way to demonstrate SAAM’s ability to handle real world data packet flows in his thesis, “Realistic Traffic Generation Capability for SAAM Testbed.” LT Turksoyu was able to model Constant Bit Rate (CBR) packets (which simulate uncompressed video/audio bit streaming), train packets (which demonstrates compressed video and voice like traffic), Poisson distribution packets (which act like bursty email and web page traffic), and self-similar (which provides a more realistic packet model for total traffic generation and loads of a network segement [an aggregate of numerous protocols including FTP, HTTP, STMP, etc.]).

Recent work in SAAM includes the thesis by Lieutenant Commander Paulo Silva, Portuguese Navy, and the thesis by Captain Troy Wright, United States Marine Corps. LCDR Silva’s work, “Advanced Quality of Service Management for Next Generation Internet,” added an additional level to the QoS feature of SAAM to increase network utilization through a protocol, which allows borrowing of resources between Integrated and Differentiated Services. Capt Wright’s thesis, “Fault Tolerance in the Server and Agent Based Networking Management (SAAM) System” provided the foundation for expeditious re-routing of SAAM packets in the event of a link or router failure.

THIS PAGE INTENTIONALLY LEFT BLANK

III. NETWORK FAULT-TOLERANCE AND SECURITY

A. BASIC CONCEPTS AND DEFINITIONS

So why is fault tolerance and security so important? Why should we incorporate these features into our program development? Is it really worth the added development time and extra lines of code? Does it improve system performance or just add to the network traffic overhead? The answers to these questions are addressed below.

With the advent of computer networking, comes the harsh reality that there are many computer users whose goal is to intercept, interrupt, modify and/or fabricate objects on other computer systems. This can be accomplished through the use of software modification applications such as logic bombs, Trojan horses, trapdoors, information leaks, and computer viruses. In 1986, there was only one known case of computer virus. [FRE00] This figure has grown rapidly to over 42,000 reported computer virus infections in 1999. Between 10 and 15 new viruses are being discovered daily. While most people believe that attackers are usually from people outside the organization that is being attacked, almost 70% of all attacks come from insiders.

One term for these malicious computer users is "Hackers". There are two different types of hackers, nicknamed Black Hat and White Hat. A White Hat is someone who breaks into another system and reports the programming flaw of the system to all users, free of charge, to prevent unauthorized malicious attack on that system. The Black Hat uses the vulnerability of the system for his personal gain, regardless of the effect to others. Black Hat hackers can be further broken down into three main categories: 1) "Elites" who write their own code, 2) "Script Kiddies" who use other people's code in a creative way to exploit computer systems, and 3) "Lamers" who use other hackers' tools in an unsophisticated manner and leave lots of evidence of their presence.

The goals of these hackers are to gain access to a shell, to gain root access to an operating system, to be destructive (or non-destructive), to gain further access into other sections of the computer system, or to "Trojanize" the system. Their approaches can be stealthy or "loud and ugly". Most Hackers take a six-step approach to their hacking.

Their hack begins with reconnaissance/intelligence gathering of the entity that they wish to access. Next, they usually scan the system to find any open ports to exploit. This is followed by defining their goals, formulating their game plan and picking their type of hack. Fourth, the hackers will usually test their hack on a simulated system. The hacker will then perform the hack and finally, cover his tracks.

These hacks can inflict significant cost in terms of time and money. The Internet Worm attack in 1988 affected between 2100 to 2600 Sun and VAX systems, costing an estimated 2 million machine hours of loss network access, 8.3 million user hours of loss network access, $41.5 million in lost machine time costs and $25 million in lost access time. [NPS00]

B. FAULT TOLERANCE CONCEPTS

1. General Fault Tolerance

"Fault tolerance" is defined as the use of multiple systems or components in parallel to provide redundancy in case of a system/component failure. Often, there is confusion over the definitions of robustness and fault tolerance of a system. The difference between the two lies in whether an event was expected or unexpected. In a robust system, the system handles expected problems, and is able to maintain stable performance. A fault tolerant system is able to handle unexpected events.

The International Federation for Information Processing Working Group 10.4 and the IEEE Computer Society Technical Committee on Fault Tolerant Computing provide definitions and standard terminology for fault tolerance. They include definitions of failure, error, fault, and error latency. Failure is defined as "a deviation of the actual behavior from the specified behavior." [SIE91] Error is defined as "a defect in the module." [SIE91] An error is the by-product of a fault. The difference in time from the detection of an error and the detection of a failure from that error is known as error latency. Other significant definitions used are the Mean Time To Failure (MTTF), which is the mean time between known failures of a system, and the Mean Time To Repair (MTTR), which is the mean time to repair failures.

Fault tolerance design concepts include fail-fast designs, independent failure modes, and redundancy and repair designs. Fail-fast designs provide the system with an operating component that will stop working upon failure. With independent failure modes, the failure of one component will not effect operations of the remaining components. Finally, a redundancy and repair design requires advanced, built-in system spare components, in case one component fails, a second component will be able to seamlessly take over. The failed component can then be removed from the system, repaired, and returned to operation.

In general, maximizing the MTTF and minimizing the MTTR, while maintaining the associated cost within acceptable levels, is the bottom line in designing a fault tolerant system.

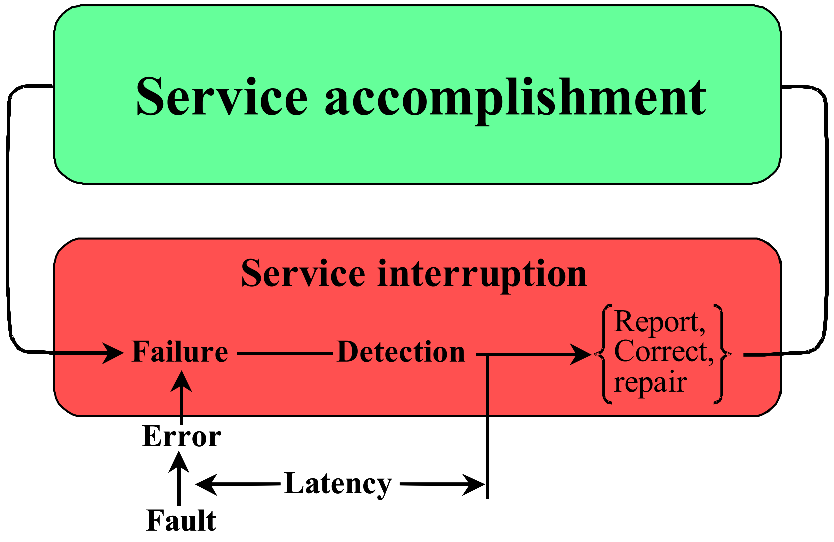


Figure 5. Depiction of Fault Tolerance Handling

Figure 5 is a duplicated graphical representation of the aforementioned fault tolerant design concepts. In most cases, the behavior of a module or system component mimics its designed specifications and is in its “service accomplishment state”. However, this does not always hold true and, on occasion, error/fault can occur in this module. “Then the module enters the service interruption state. After the failure is detected, reported, and corrected or repaired, the module returns to the service accomplishment state.” [SIE91]

2. Redundancy

When considering robustness in a computer system, there are four different ways in which a system can implement redundancy. These areas are hardware, software, information and time. In hardware redundancy, extra hardware is added for error detection or fault tolerance. Software redundancy, on the other hand, is designed through extra software coding for error detection or fault tolerance. Information redundancy uses extra data or codes and time redundancy uses extra processing for error detection or fault tolerance. All four areas will be discussed in detail below.

a. Hardware Redundancy

Hardware redundancy can be broken down into two types, passive and active. In passive hardware redundancy (also known as static techniques), fault masking is used to hide occurrence of faults. No actions from the system are required to provide this functionality. Examples of this type of system include Triple Module Redundancy (TMR) (See Figure 6 and Figure 7), N-Modular Redundancy (NMR), and replicate voters. (See Figure 8.) TMR can be accomplished through multiple components (e.g. CPU's, Hard Disk Drives (HDD's), Network Interface Cards (NIC's), etc.) with each component being from the same or different manufacture. NMR is currently being used on most space systems where N equals the number of different active components that make up the modular redundancy (i.e. on the Apollo space missions, N = 5 which also included 2 brands, 2 OS, 2 languages and 2 algorithms.) Hardware replicate voters are designed using "and" and "or" gates to provide redundancy.

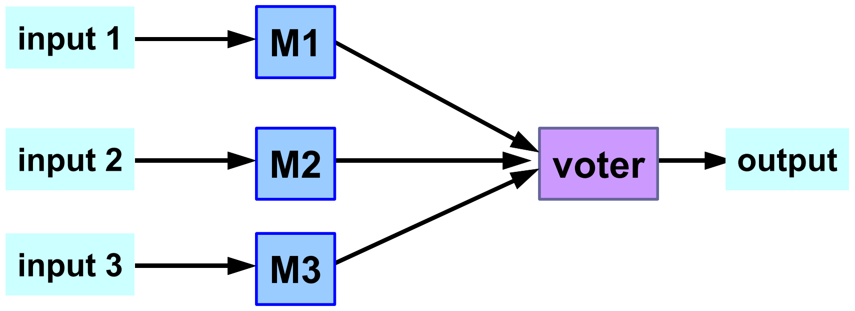


Figure 6. Triple Modular Redundancy before error detection [DEC99]

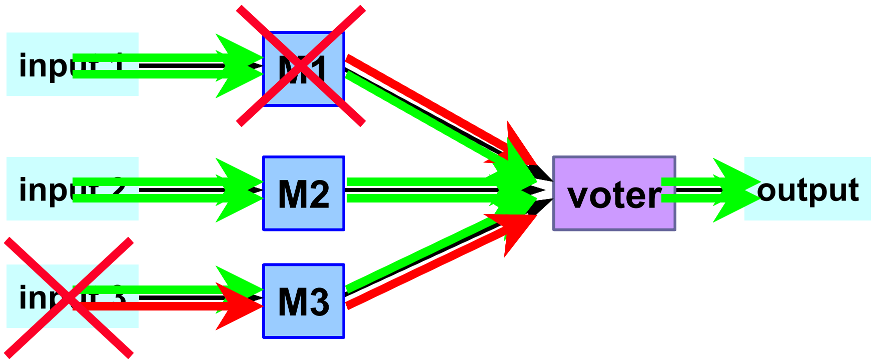


Figure 7. Triple Modular Redundancy after error detection [DEC99]

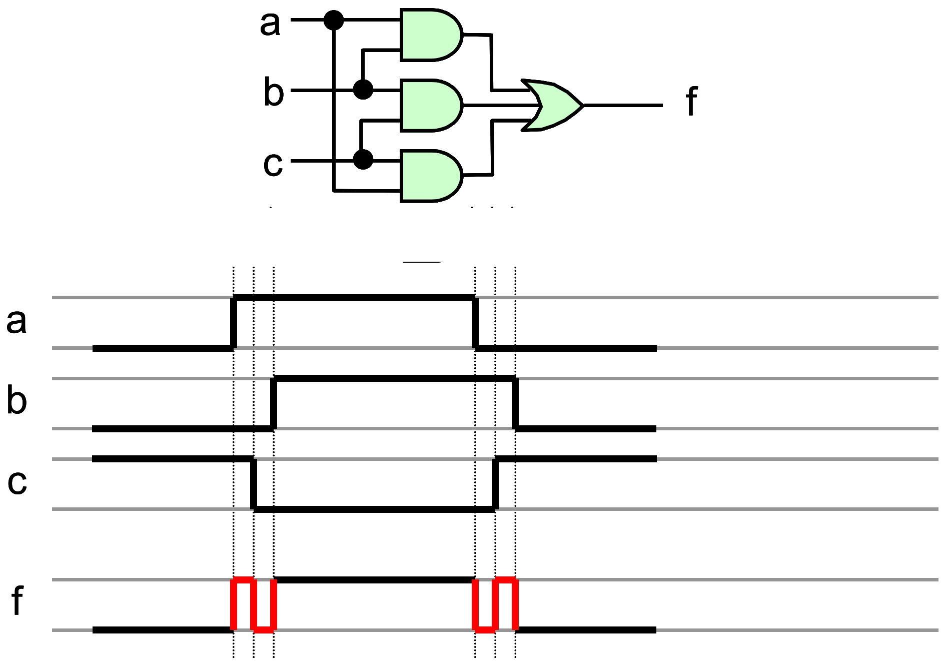


Figure 8. Hardware Voting [DEC99]

Active hardware redundancy (also known as dynamic hardware redundancy) involves techniques such as detection, localization, containment and recovery for correct results. It uses comparison for detection and/or diagnosis and removes faulty hardware from the system or reconfigures the system such that the faulty component is no longer in the decision loop. Dynamic redundancy can be broken into two types, duplication without comparison and standby sparing (backups). (See Figure 9.) In duplication without comparison, two active components are used to detect many different types of faults without localization of them. Standby sparing has two components in parallel with one of the components inactive until the other goes into a failure mode. An example of standby sparing is a duplicate battery cell on a satellite that is dormant until another battery cell fails.

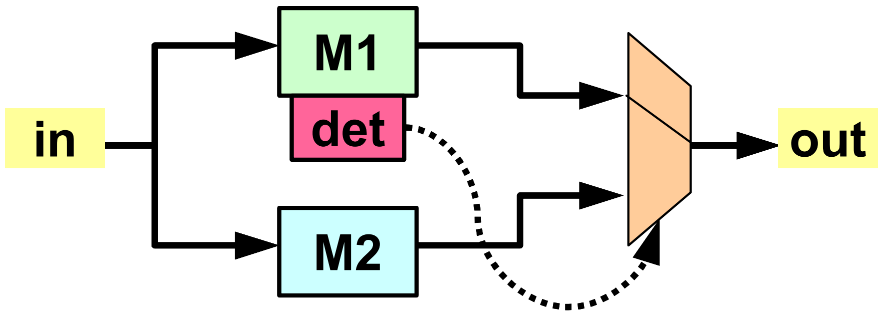


Figure 9. Standby Sparing dynamic redundancy [DEC99]

b. Software Redundancy

In general, software redundancy uses extra software checks to detect and correct system faults. The major software redundancy implementations are consistency checks, capability checks, N-Version Programming, and recovery blocks. Consistency checks perform an acceptability assessment between multiple inputs, redundant information, and any invariants checking for the range, history, etc. of the output in order to evaluate the system. Capability checks look over the systems to determine if its components are appropriate to allow the system to operate effectively. N-Version programming uses different development teams, languages, compilers, methodologies, etc., for the same specifications and functionalities to provide for diverse failure modes. Recovery blocks are a software equivalency to standby sparing. (See Figure 10.)

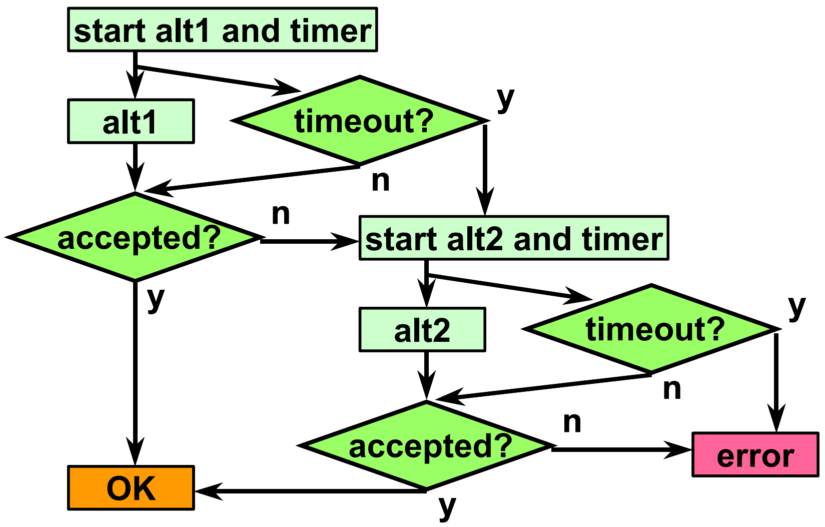


Figure 10. Recovery Block Software Redundancy [DEC99]

c. Information Redundancy

Information redundancy uses added information in the data stream to help detect error and, in some cases, correct these errors. Some areas where information redundancy can be found are Error Detecting Code (EDC), Error Correcting Code (ECC) such as Hamming code, Redundant Array of Inexpensive Disks (RAID 5), and Cyclic Redundancy Checksum (CRC).

d. Time Redundancy

Timing redundancy refers to the repetition (re-execution) of an algorithm and comparing the outcome. (See Figure 11.) Then, a voting mechanism is put into place to determine the correct outcome. This method is only efficient for transient faults. Permanent faults need to be corrected through dynamic hardware redundancy.

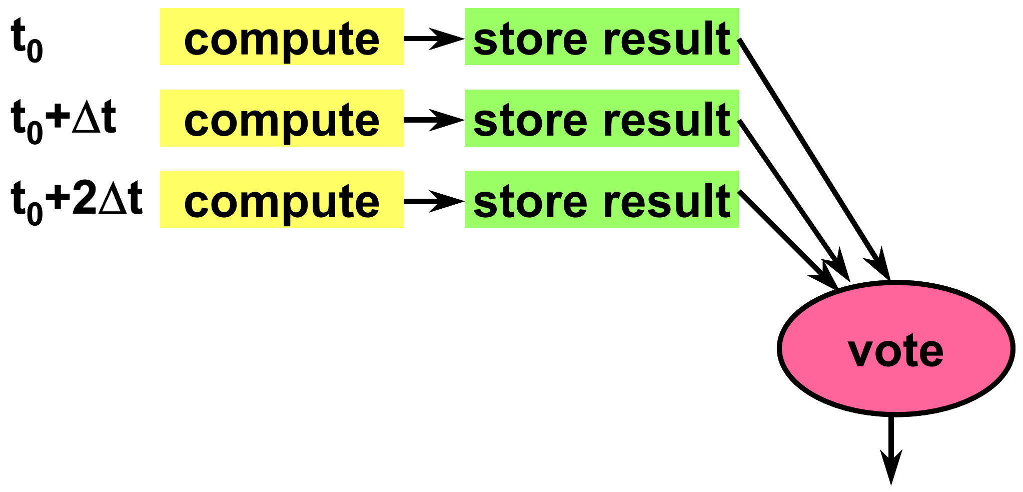


Figure 11. Time Redundancy [DEC99]

C. DENIAL OF SERVICE

1. SYN Attacks

In a TCP connection, the two hosts engaged in a handshake connection before they create a session and start transmitting data. The session is set-up in the following manner [BRE00]:

a. Source sends a packet to the destination with SYN=1. (SYN stands for synchronization. When SYN=1, then this flag is set to TRUE and synchronization has occurred. When SYN=0, synchronization has NOT occurred.)

b. Destination replies to the source with SYN=1, ACK=1. (ACK is the Boolean flag for acknowledgement.)

c. Source sends a packet to the destination with ACK=1.

d. Source starts transmitting data.

When the destination host receives the first SYN, it stores the connection request in a small queue in anticipation that the subsequent connecting handshake will occur quickly and the session will be transferred to a larger queue. The SYN queue is deliberately small in order to optimize memory utilization, assuming that there will be a low number of pending connection requests.

In a SYN attack, the hacker floods the destination host with a lot of TCP connection requests but does not perform step c for any of them, hoping to overload the SYN queue. The requests remain in the queue until a timeout occurs. The hacker maintains a steady flow of such connection requests, which denies use of the destination host to other hosts. If a firewall is in place which can detect this kind of attack, a rule can be added to the firewall to deny any TCP connection requests from a suspected "flooding" host.

2. Smurf Attacks

The Smurf attack was named after its exploit program. It uses the Internet Control Message Protocol (ICMP), which was designed to handle errors and exchange control messages. ICMP can also be used to determine if a host on the Internet is responding to requests. One example of this is the ping command where the system sends a signal to a host and that host responds by sending a packet back to the originator, letting it know that the host is a live.

In the Smurf attack, the hacker will send several ICMP messages to a target. The target can be a host or a broadcast address. Unlike a host which only delivers a single reply to the originating host, a broadcast address will bounce the ICMP to all the computers on that network which in turn reply. This increases the traffic on the network exponentially.

The hacker does not use his computer's IP address as the host originating the ICMP request. Instead, he uses the targeted computer's IP address. Therefore, all replies to the ICMP go to the targeted computer. This is where the denial of service (DoS) occurs. Sophisticated hackers generate automated tools that bounce ICMP packets off several unsuspecting broadcast hosts towards the same victim, intensifying the attack.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. MAGMA©'S DESIGN OF SECURITY, FAULT TOLERANCE, AND SURVIVABLE NETWORKING

A. PURPOSE

MAGMA© was designed out of concern that SAAM's architecture was leaving a gapping hole in its ability to be secure and fault tolerant. The SAAM server is the lynch pin that holds the SAAM architecture together. If there were to be degradation in SAAM server performance, either due to a failure of the SAAM server or due to a malicious attack, the lack of an operational SAAM server would be catastrophic, completely shutting down the SAAM Region from processing Quality of Service (QoS) sensitive traffic.

There needed to be policy put into place to shore up this shortfall in contingency planning. In order to do this effectively, MAGMA© leveraged off of preexisting agent deployment techniques and past research in liquid software. Upon initial research to determine if any related work had been done in this area, it was surmised that liquid software's existence began around 1996 at the University of Arizona in a project called "Liquid Software." However, the University of Arizona's project only dealt with providing code with the ability to maneuver around a network and then standup and run on the new host. Past work in liquid software did not handle maintaining state information about the program while maneuvering from one host to the next. With the use of this information and the previous thesis work completed on SAAM, MAGMA© came into fruition. Its ultimate goal was to provide survivable networking through fault tolerance and computer security.

The SAAM server is such a critical component of this system that it is imperative to the network that the server remains operative. Therefore, there is a need to build a more robust server. Liquid software is a programming concept similar to mobile code. Basically, the idea is to create a software agent, which will be running in resident memory. This agent will remain inactivate until it is stimulated by a message. Upon activation, the agent will begin running a new instance of the target application (SAAM server). The starting point of this application will be determined from the recent state information of the prioir instance of the application or a pre-determined starting point if that state information is not available. Liquid software will provide SAAM with an increased level of fault tolerance as well as security against malicious attacks.

B. DEVELOPMENT

1. Pre-MAGMA© SAAM

SAAM acts like a helicopter reporting on rush-hour traffic. When a routing request comes in from a computer, the SAAM server (like the helicopter) looks over the network, determines the best route with the least traffic/resistance, and assembles the routing path. (See Figure 12.) It has been determined in prior thesis projects that the SAAM Region could be optimized with fewer than 40 hosts in a mesh topology. This provides the SAAM server paths with no more than a 9-hop average. Since the SAAM server makes the routing decisions, the "light-weight" routers are freed-up to provide faster, more reliable, forwarding services. This allows these streamlined routers to provide a better Quality of Service through utilization of the router resources as data forwarders, instead of requiring the routers to calculate routing tables. The SAAM server overlooks an autonomous system called the SAAM Region. In order to provide SAAM services across several Autonomous Systems (AS) (also known as ISP's), a super SAAM would be deployed to provide recommendations for routing between AS's.

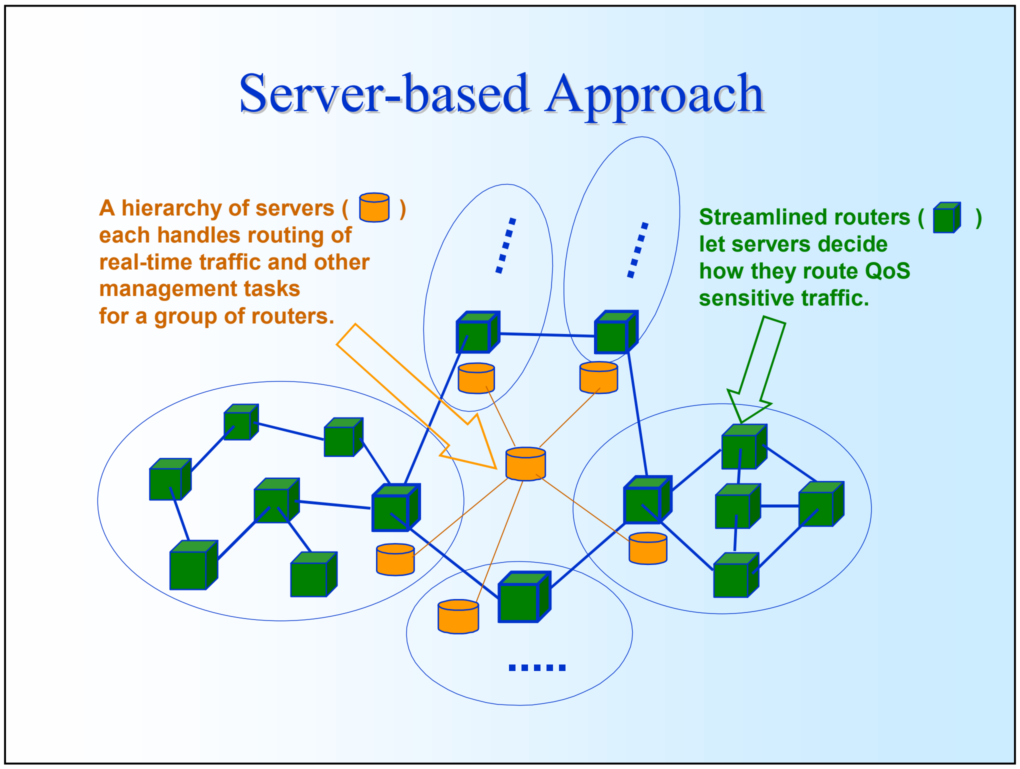

Server-based Approach
A hierarchy of servers ( ) each handles routing of real-time traffic and other management tasks for a group of routers.
Streamlined routers ( ) let servers decide how they route QoS sensitive traffic.

Figure 12. SAAM Server's Responsiblity

The SAAM Server updates its PIB through a series of administrative messages. (See Figure 13.) A Downward Configuration Message (DCM) is broadcast to its routers to determine the most efficient data paths (signaling channels) for server-router communications. When a router receives a DCM, it sends a message (known as a Parent Notification Message or PN) to the first router who passed it the DCM. The PN message lets the upstream router know that it is some router's quickest path to the SAAM server. In doing so, this also lets the upstream router know that it has children. When the leaf (or lowest level child) receives a DCM, it responds with a PN and an Upward Configuration Message (UCM). The parent takes this UCM, attaches its name and information to the message and sends it up to its parent. When all the UCM's arrive at the SAAM server, the BasePIB is updated. These events occur every 500 milliseconds so that the server can quickly learn changes in toplogy due to network faults or addition of hardware.

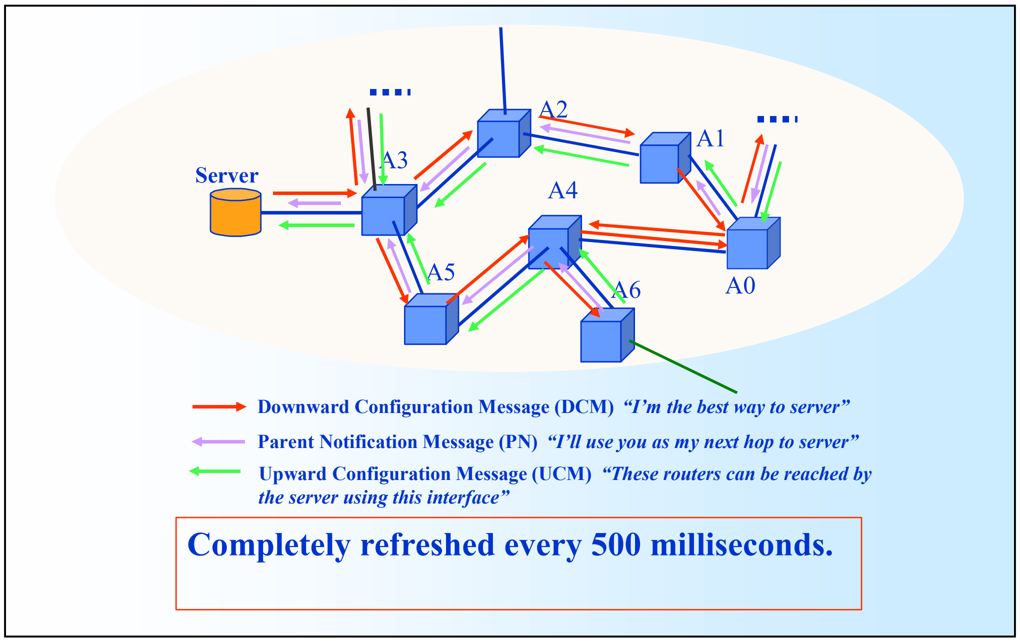

Server
A3
A2
A1
A4
A0
A5
A6
Downward Configuration Message (DCM) "I'm the best way to server"
Parent Notification Message (PN) "I'll use you as my next hop to server"
Upward Configuration Message (UCM) "These routers can be reached by the server using this interface"
Completely refreshed every 500 milliseconds.

Figure 13. SAAM's Auto Configuration Protocol

Currently, SAAM's only server fault tolerance is provided through a static SAAM server and a designated back up server, much like the Primary Domain Controller (PDC) and the Backup Domain Controller (BDC) of a Windows NT local area network. (See Figure 14.) The fault tolerance occurs through a protocol similar to a Hot Synch Reservation Protocol. The backup server sends out a SAAM message called a "Heartbeat Query" every 250 milliseconds (msec) to see if the SAAM server is still operating. If the backup server receives a reply, then it does nothing. Otherwise, it sends out another Heartbeat Query after the 250 msec wait timer expires, this time only waiting 125 msec for a reply. The wait timer is halved every time it expires before a reply is received from the primary server. This continues until a threshold is met, at which time the backup takes over as the primary SAAM server.

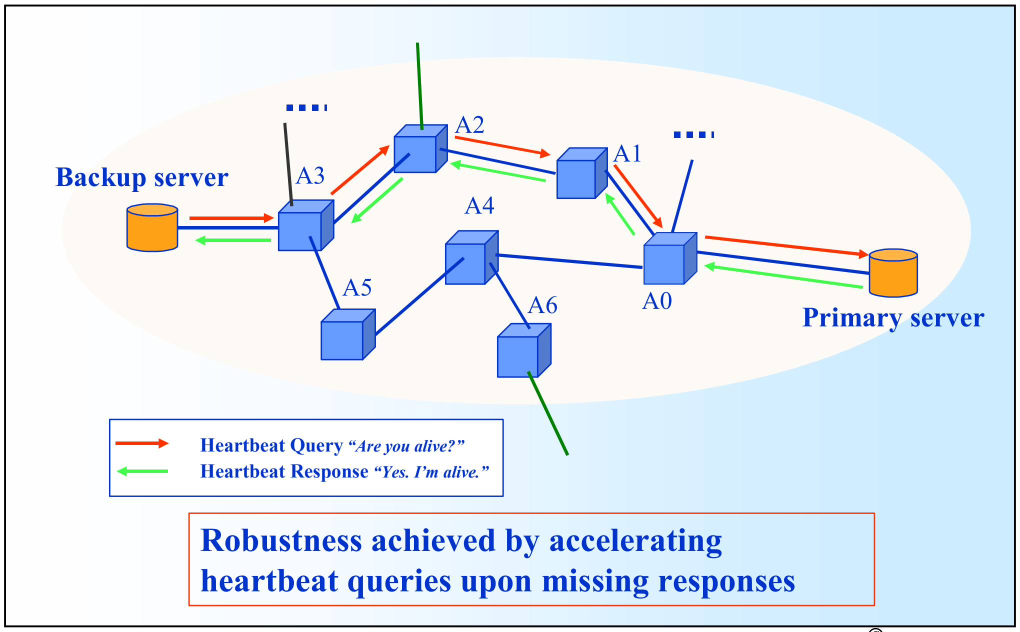

Backup server
A3
A2
A1
A4
A5
A6
A0
Primary server
Heartbeat Query "Are you alive?"
Heartbeat Response "Yes. I'm alive."
Robustness achieved by accelerating heartbeat queries upon missing responses

Figure 14. SAAM's Fault Tolerance before MAGMA©

2. MAGMA©'s Introduction to SAAM

This is where MAGMA© is introduced to the SAAM architecture. Currently, the primary server and backup servers are static, making them a "sitting duck" for all kinds of attacks and misfortunes. The SAAM server is a critical node in the SAAM Region. If a hacker disables it, the entire SAAM Region is affected. With MAGMA©, the mobile code will act like a shell game to potential attackers. (See Figure 15.) If a hacker tries to disrupt the system through a SYN or Smurf Denial of Services (DoS) attack, he will first have to find out where the SAAM server is located in the SAAM Region. By the time he launches his attack, the server will have already moved to a new location. With a little magic and slight of hand, the system will be safe against most attacks.

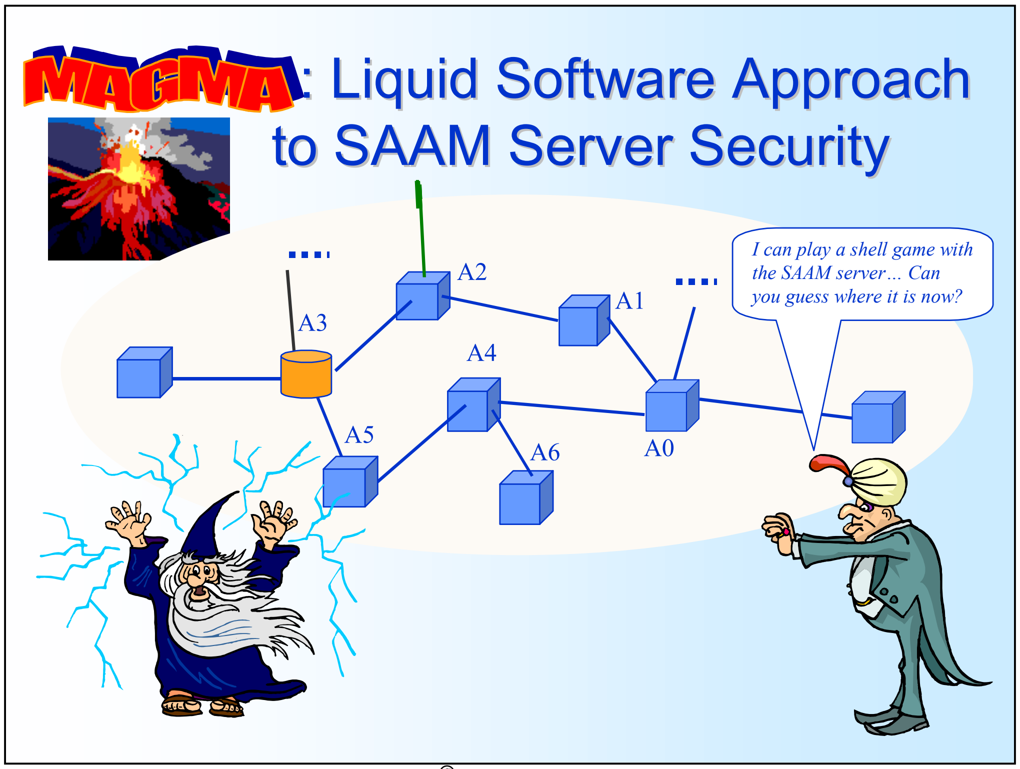

MAGMA: Liquid Software Approach to SAAM Server Security
....
A2
A1
....
A3
A4
A5
A6
A0
I can play a shell game with the SAAM server... Can you guess where it is now?

Figure 15. MAGMA©'s Security Addition to SAAM

MAGMA©'s ability to move the server across the network will allow SAAM to provide for a robust system with more built in fault tolerance. Each router will have the ability to act as either a light-weight router and/or as the SAAM server. (See Figure 16.) A Token will be held by the router/computer that is hosting the SAAM server. If the routers in the SAAM Region do not receive a DCM from the server within a prescribed period of time, the token will be regenerated and one of the remaining routers will take on the role of the SAAM server. This concept will also aid in periodic maintenance of a server, as the movement of the server can be manually activated.

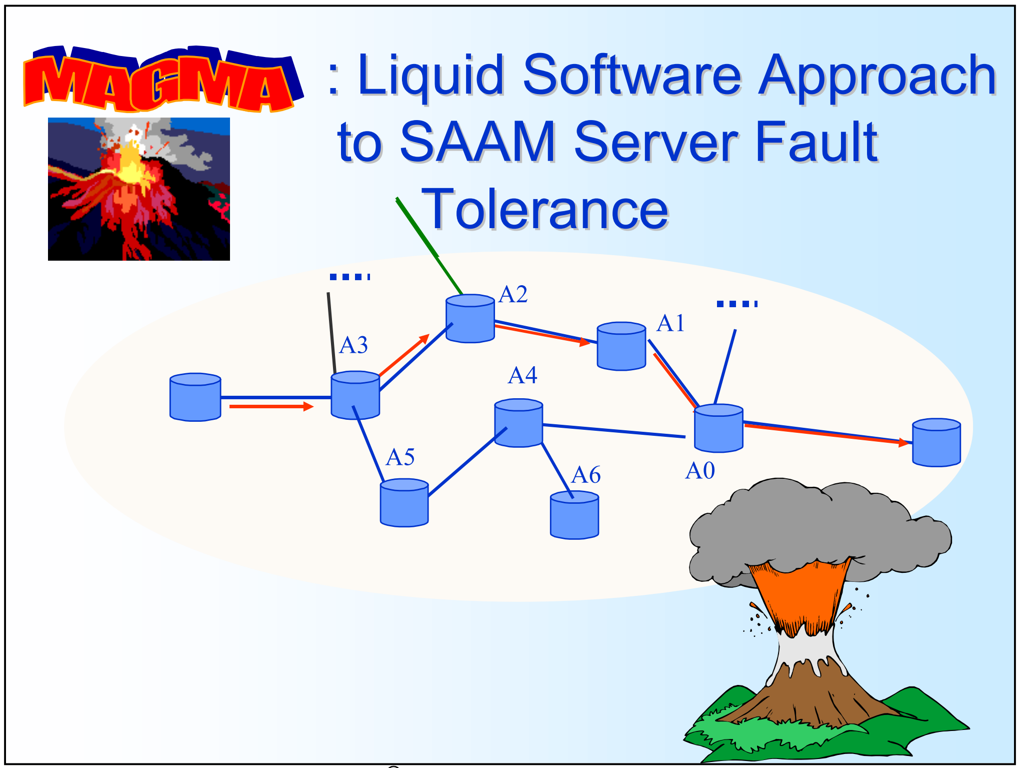


Figure 16. MAGMA©'s Fault-Tolerance Addition to SAAM

The SAAM server keeps track of autonomous system state information through signaling channels between the server and the routers. (See Figure 17.) This link state information is periodically sent to the SAAM server who uses this information to update its Path Information Base (PIB). The PIB is SAAM's master routing table. When an edge router receives a request for information to pass through the SAAM Region, it sends a flow allocation request message ("Call setup") to the server. The server then looks at its PIB for the best route and sends resource allocation messages to all the routers in that route. These routers take this information, reserve the required queues, and update their flow routing tables. SAAM also is designed to send the routers "agents" (or Snap-in) to handle special case routing requirements. MAGMA©'s deployment and setup features take advantage of SAAM's pre-existing capability to send agents.

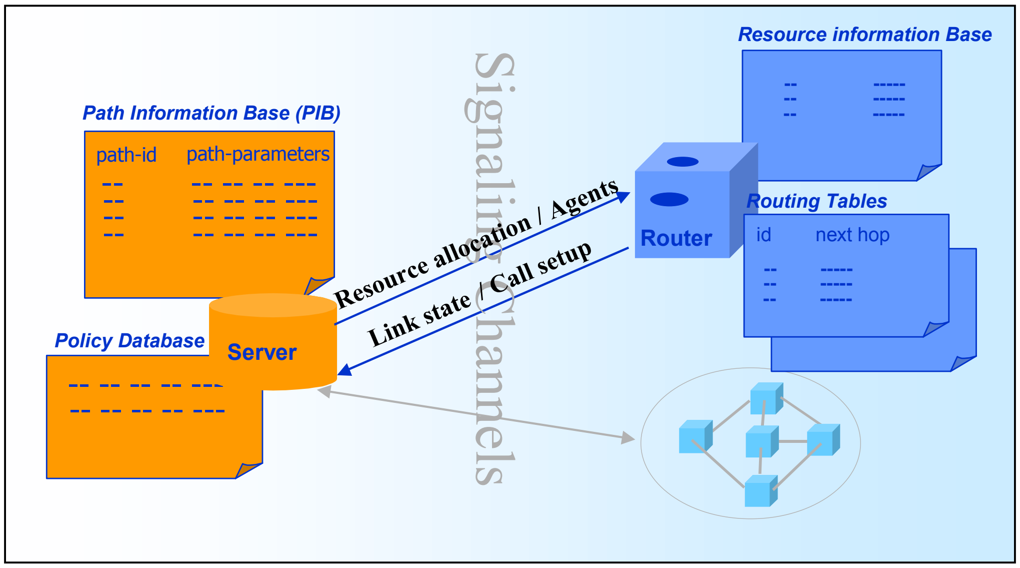


Figure 17. SAAM's Server Router Communications

C. INTEGRATION WITH SAAM SERVICES

In order to first tackle the problem of mobile code, a mechanism had to be designed which would allow for state information to be retained and distributed throughout the SAAM Region to all routers. Before this thesis, SAAM maintained its state information in a data structure known as the Base Path Information Base (*BasePIB*). With a six node simulated SAAM architecture, PIB size was calculated at approximately 310 kB. This size grows rapidly as the number of nodes increases. If MAGMA© were to send the whole PIB information over the network to each host, the links would be slowed down by this excessive overhead traffic.

MAGMA© took the need to diminish the overhead traffic into account through development of a device which will allow state information to be passed from one host to the next. This mechanism needed to be a container that would hold only the key essential volatile information stored in the PIB. With this information, a new host would be able to stand-up and operate expeditiously, efficiently, and seamlessly. Its minute footprint would allow timely updates to all potential hosts with minimal bandwidth effects. This instrument has been deemed Digest Path Information Base, or *DigestPIB*. Figure 18 shows the information in the *BasePIB* (in red, bold, italics) that was determined volatile and placed as part of the *DigestPIB*.

Base PIB	*Digest PIB*
● ***aPI[][][][]***	● ***digestAPI[][][]***
● htRouterIDtoNodeID	● ***digestHtInterfaces***
● htNodeIDtoRouterID	● ***digestHtPaths***
● htRouterInterfaceMap	
● ***htInterfaces***	
● ***htPaths***	
● htUserSLSs	

Figure 18. Base Path Information Base vs. Digest Path Information Base

In order to further reduce the amount of overhead from the state information stored in the *BasePIB*, these dynamic data structures were pulled out of the *BasePIB*, placed into the *DigestPIB*, and then shrunk with a java standard compression algorithm to help further combat the potentially huge amount of data stored in the *DigestPIB*. (See Figure 19.) Compression allows for transferring data in a smaller form across a network with lower bandwidth utilization. This is exactly the reason why compression is being used within the SAAM project. It provides smaller state information messages, allowing servers expeditious configuration and setup. The details of DigestPIB design and data compression are presented in Chapter V.

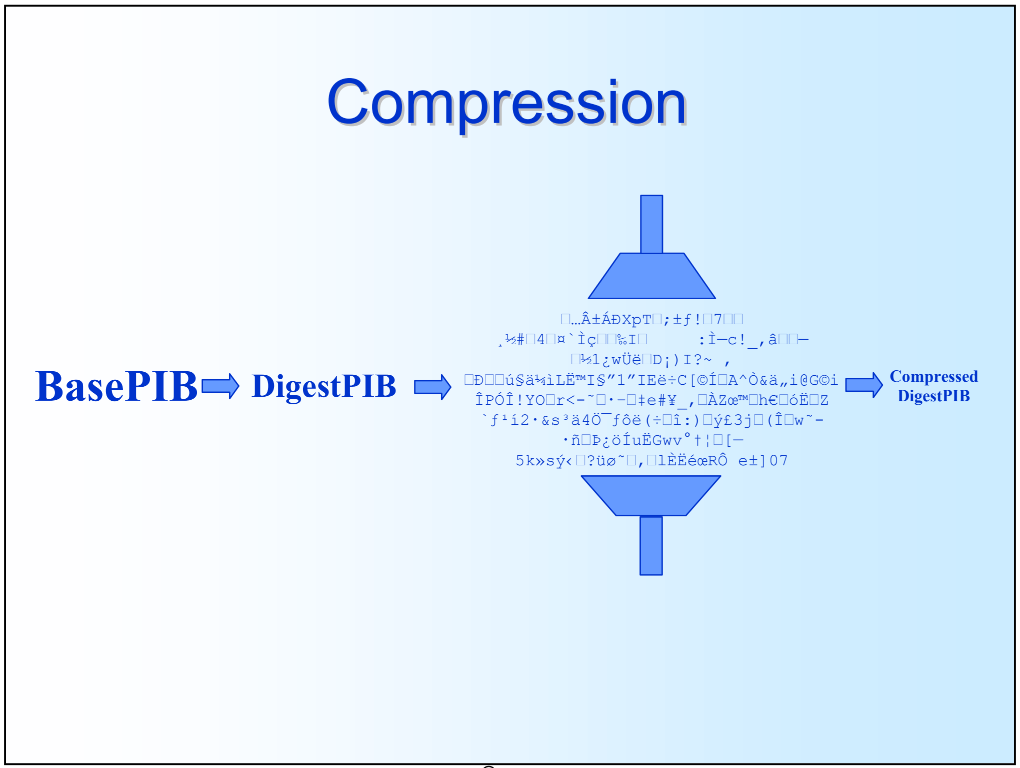
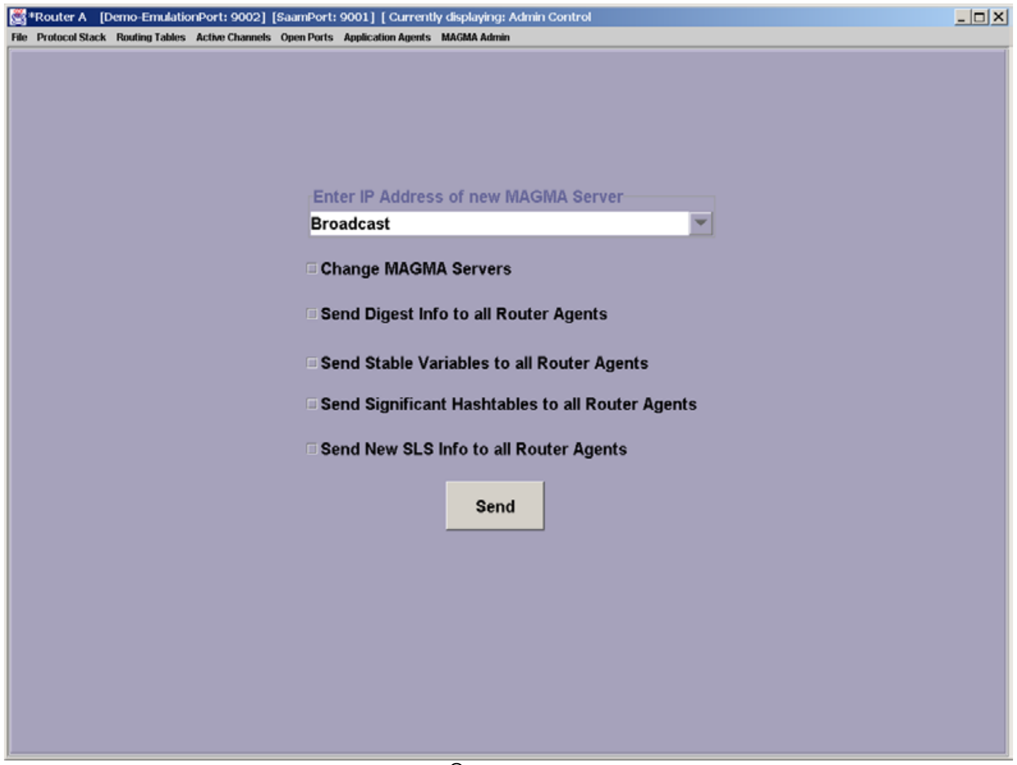


Figure 19. MAGMA$^{©}$'s Compression Routine

The next step in designing MAGMA$^{©}$ was to create a controller interface. This Graphical User Interface (GUI) had to be integrated with the existing SAAM *mainGui*. (See Figure 20.) Two new classes, *MAGMAAdminGui* and *MAGMAContorlGui*, were developed which leveraged off the pre-existing *SAAMRouterGui* class. The *MAGMAContolGui* provided the window which allowed a SAAM administrator to control the MAGMA$^{©}$ related tasks, giving the ability to manually transfer the control of server to another host, send the *digestPIB* of the current server to other hosts, update the relatively static data structures (those that are part of the BasePIB, but not included in the DigestPIB) at the other hosts, and send updated Service Level Agreements (SLAs) to other hosts. The MAGMAAdminGui was designed to monitor the *MAGMAControlGui* for process selection, host IP addresses, and selection of the action (send) button.


*Router A [Demo-EmulationPort: 9002] [SaamPort: 9001] [Currently displaying: Admin Control
File Protocol Stack Routing Tables Active Channels Open Ports Application Agents MAGMA Admin
Send

Enter IP Address of new MAGMA Server

Broadcast

- Change MAGMA Servers
- Send Digest Info to all Router Agents
- Send Stable Variables to all Router Agents
- Send Significant Hashtables to all Router Agents
- Send New SLS Info to all Router Agents

Figure 20. MAGMA© Admin Control Screen Shot

Once it was proven that the send button, selection boxes, and combo boxes worked correctly, a message had to be designed to pass the required information to the appropriate host(s). This *MAGMATokens* message extended SAAM's *message* class and utilized many of the same fields that were found in SAAM's *interfaceSA* message class. The message make-up can be seen in Figure 21 on page 36. The *MAGMATokens* (message) had to incorporate the *ByteArrayOutputStream* from the *squeezeObject( )* method to allow for transportability over the SAAM control channels. This message provides MAGMA© the ability to send a *magmaAgent* the information needed to configure itself to begin running the SAAM *serverAgent* on that host.

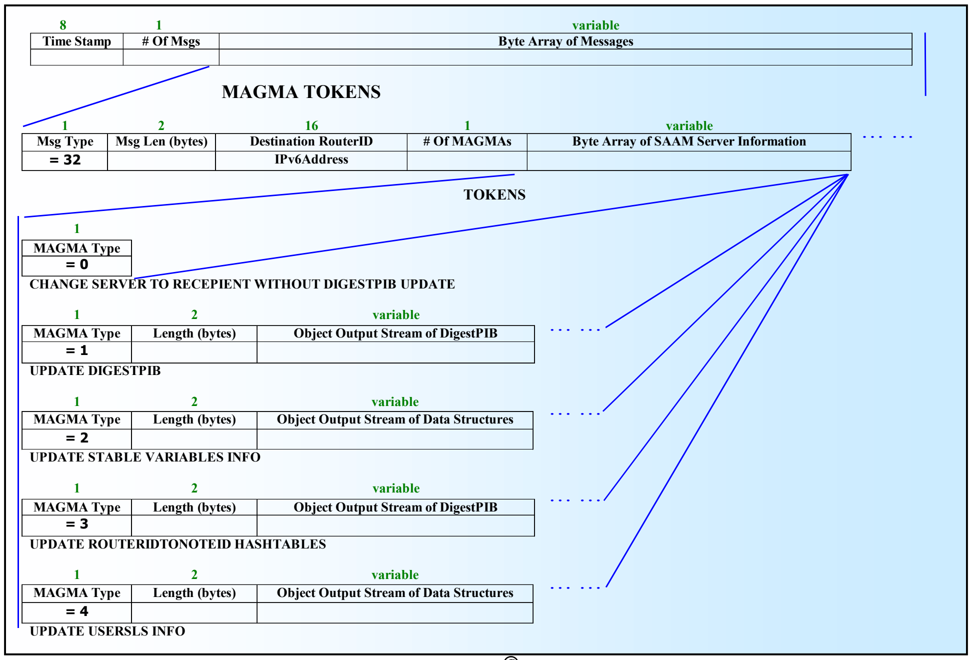


Figure 21. MAGMA© Message Format

After the *MAGMATokens* message was developed, it was incorporated into the *MAGMAAdminGui*, giving the required functionality to the *MAGMAControlGui* interface. A method called *processMagmaController* was developed as a handle to the *MAGMAAdminGui* and placed in the *Server* class, providing the server host control of MAGMA©'s functionalities.

Next, a new class was created called "*magmaAgent*," which removed the *SqueezeObject* and *ExpandObject* methods from the *Server* class and placed them in a central container (This was a preliminary design which was found to be inefficient and later changed. More details to the final design will be provided later in this chapter). *SqueezeObjectToFile( )* was rename to *squeezeObject( )* method to reflect that the data was no longer going to a file, just getting compressed for TCP/IP transfer. Upon completion of this challenge, the next daunting task was to determine how to send the message over SAAM's existing control channels and ensure reception by the proper host. This proved to be a very difficult issue to complete as SAAM's code had several ways to send messages over several different channels. It was found that the *send( )* method found in SAAM's *controlExecutive* class was used as the vehicle to place the required information on SAAM's Control Channel.

Still, there needed to be way to add a handle from the *Server* object to the *magmaControlGui*. This handle was devised, allowing the MAGMA$^{©}$ controller to associate the MAGMA$^{©}$ message with its *Server* object. Debug statements were added to ensure that proper information was being retrieve from the *magmaControlGui*. The new MAGMA$^{©}$ server's IPv6Address needed retrieval from the *magmaContolGui*. This address is sent to the MAGMA$^{©}$ message for building a SAAM Packet. This message packet is sent to the receiving host, which then processes it and takes over the responsibilities of the MAGMA$^{©}$ server.

Now that the server had the ability to select a message, compress it, and put it on a SAAM channel to be sent to a new host, the receiving host's inbound packet factory had to be altered to accept the new *MAGMATokens* message. Once the inbound *packetFactory* received the message, the host's *controlExecutive* class had to process the message and provide it the ability to manipulate the newly received message. It was found that the *controlExecutive's parseMessage( )* and *getMessage( )* methods, along with the *magmaAgent's processMagmaMessage( )*, needed to be altered to give this functionality. With the new host receiving the message, a method was devised to kill the old SAAM server's "Router x currently hosts the Primary Server" banner to show that the server was moving to a new host. At first, a new *KillBannerMessage* was developed. It was thought the host receiving the CHANGEMagma message would return the *KillBannerMessage* to the original SAAM server, thus killing the server banner and providing the ability to shut it down once its functions were moved to the new host. However, after making an attempt to get this method to work, it was determined that an easier way of handling this problem was to have the original *Server* class kill its own server banner after sending its CHANGEMagma message.

With the server banner killed and the CHANGEMagma message beginning received, the new host needed to stand up and take control of the server functions. This required adding several new methods and changing many existing methods. Modeled after the current way that the *DemoInitInfo* class of SAAM stands up the original SAAM server, *standupMagmaServerAgent( )* method was added to the *magmaAgent* class to aid in this functionality. Furthermore, the translator objects, which simulate each of the hosts, needed to reflect their change in functionality. The *updateDisplay( )* and *updateMagmaAdministrator( )* methods were changed to allow for this translator update, providing visual confirmation of the MAGMA© server shift.

It was later determined that the best way to provide software liquidity of the MAGMA© server was to install a *serverAgent* on each of the hosts at startup and then placing the *serverAgents* in a dormant state until awakened by a *MAGMATokens* message. With this discovery, code was written that allowed the MAGMA© server to stand up, show that it was functioning properly by forwarding simulated network traffic, move to another host and begin operating from its pre-existing state, and continue this process over several hosts. Furthermore, recent development of MAGMA© has shown that the most efficient placement of the *squeezeObject( )* and *expandObject( )* methods is in a new class called *MAGMA©* , which is located in SAAM's server folder. Additionally, added were two methods in the *Server* class, *standUpAutoConfigThread( )* and *killAutoConfigThread( )*, which provided MAGMA© its fluidity. In order to avoid confusion and remove old entries in the server tables, flow routing tables, and router bound control channel table, *refreshServerTable( )* method in the *serverTable* class was re-written to remove old table entries.

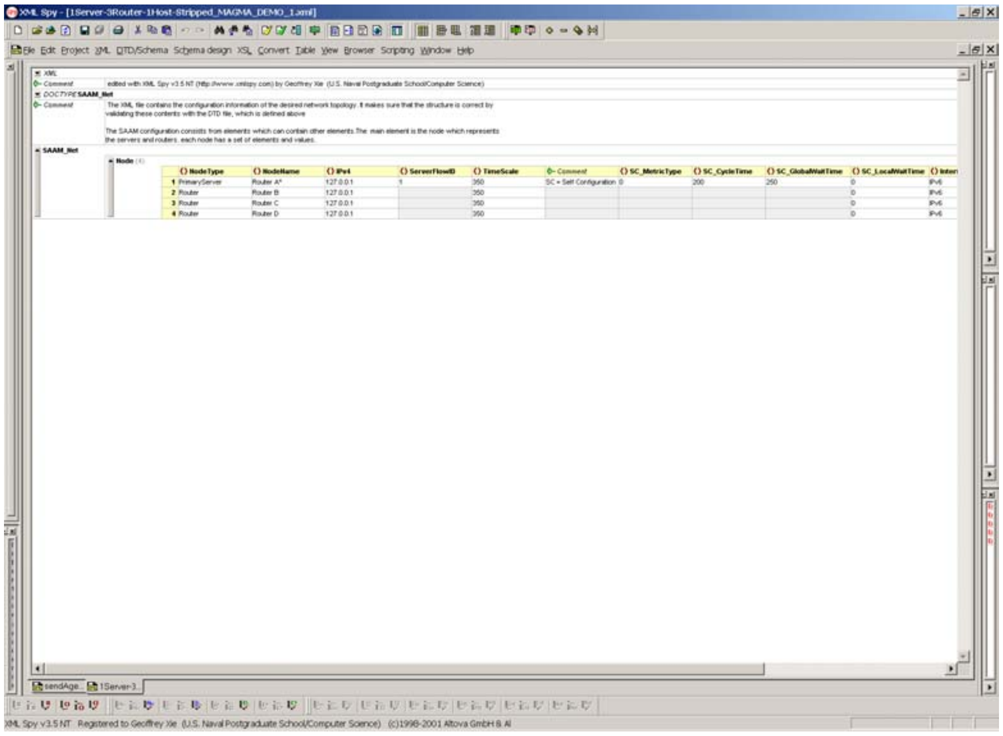
Alas, due to time constraints and the many obstacles that were overcome in executing this thesis project, MAGMA©'s liquid software did not achieve 100 percent of its original goal. This provided for many new investigations in future thesis work. These new concepts are further discussed in Chapter VI (Conclusions and Recommended Future Work). In the end, new ground was broken in the exploration of mobile agents and survivable networking.

D. HOW IT WORKS

In order to stand up the SAAM environment, an XML file needed to be created to provide each of the simulated hosts their individual behaviors (either as a router or a server). A generic object that SAAM refers to as a *translator* is used to simulate the router/server hosts. (See Figure 22.) These are stood up by executing the *main( )* method of the *multiTransTest* class. Once the required amount of *translator's* are created, they receive their behavior from the XML files through the *SAXParserDemo* class. The XML file contains information such as the translator's node type (server or router), its node name, IPv4Address, IPv6Address, serverFlowID (only for the server), time scale, self configuration (sc) metric type, sc cycle time, sc global wait time, sc local wait time, interface address type, interface information such as address, mask, and bandwidth, and any agents associated the host (either a source or sink agent). (See Figure 23.)

Translator [Demo-EmulationPort: 9008] [SaamPort: 9007] [SAAM Router Prototype]
File
Protocol Stack
Routing Tables
Active Channels
Open Ports
Application Agents
ONE FOR ALL, ALL FOR ONE
SAAM
SERVER And AGENT based network MANAGEMENT
NPS
DARPA
NASA

Figure 22. SAAM's Translator Object


XML Spy - [1Server-3Router-1Host-Stripped_MAGMA_DEMO_1.xml]

File Edit Project XML DTD/Schema Schema design XSL Convert Table View Browser Scripting Window Help

XML

Comment: edited with XML Spy v3.5 NT (http://www.xmlspy.com) by Geoffrey Xie (U.S. Naval Postgraduate School/Computer Science)

DOCTYPE SAAM_Net

Comment: The XML file contains the configuration information of the desired network topology. It makes sure that the structure is correct by validating these contents with the DTD file, which is defined above

The SAAM configuration consists from elements which can contain other elements.The main element is the node which represents the servers and routers. each node has a set of elements and values.

SAAM_Net

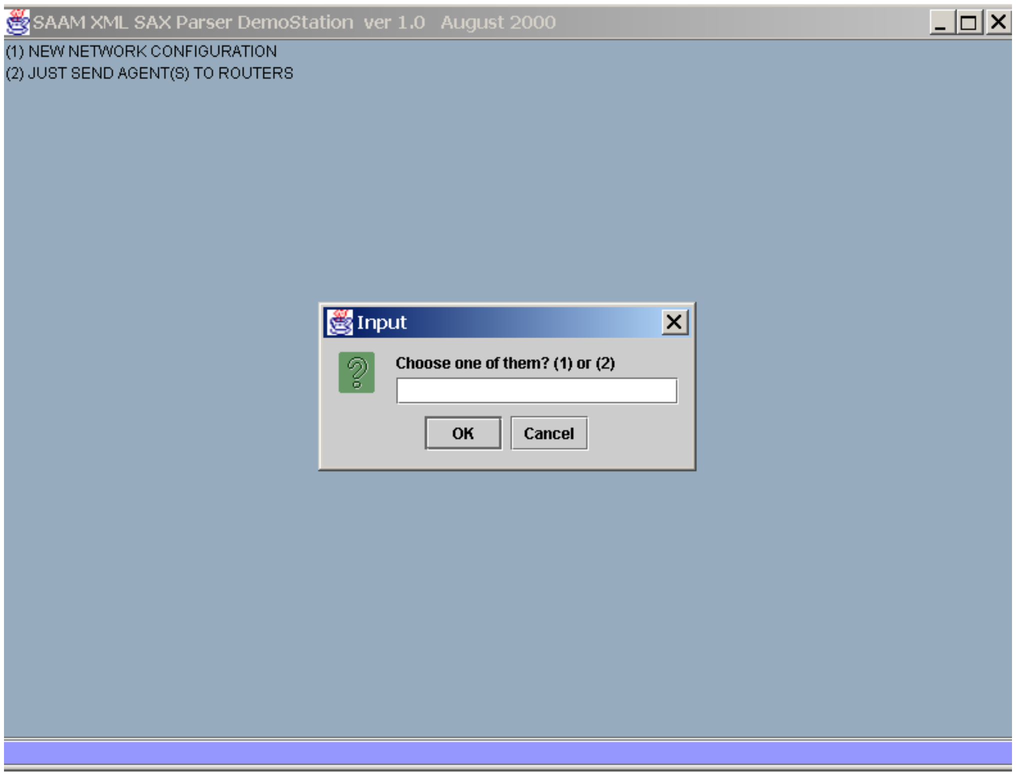
Node (4)

	NodeType	NodeName	IPv4	ServerFlowID	TimeScale	Comment	SC_MetricType	SC_CycleTime	SC_GlobalWaitTime	SC_LocalWaitTime	Inter
1	PrimaryServer	Router A*	127.0.0.1	1	350	SC = Self Configuration	0	200	250	0	IPv6
2	Router	Router B	127.0.0.1		350					0	IPv6
3	Router	Router C	127.0.0.1		350					0	IPv6
4	Router	Router D	127.0.0.1		350					0	IPv6

XML Spy v3.5 NT Registered to Geoffrey Xie (U.S. Naval Postgraduate School/Computer Science) (c)1998-2001 Altova GmbH & Al

Figure 23. XML Network Configuration Format

SAAM's innate agents either produce or absorb simulated Internet traffic using a Constant Bit Rate (simulates streaming audio or video traffic), Poisson distribution (simulating mail traffic), packet train (simulating web page traffic) or self-similar (simulating a more realistic demonstration of combined network traffic) form. The *SAXParserDemo* presents the initiator with a text box and choices of either loading a "New Network Configuration" or "Just Send Agent(s) to Routers." (See Figure 24.) If "New Network Configuration" is chosen, another text box pops up with a list of all the XML files available to set up a simulated SAAM Region. (See Figure 25.) Once the SAAM Region is stood up, the flow agents begin sending traffic over the network, downward configuration messages are sent by the server, upward configuration messages are generated by the routers, traffic flows are tracked and added to the routing tables, and the path information base state information data types are updated as necessary.

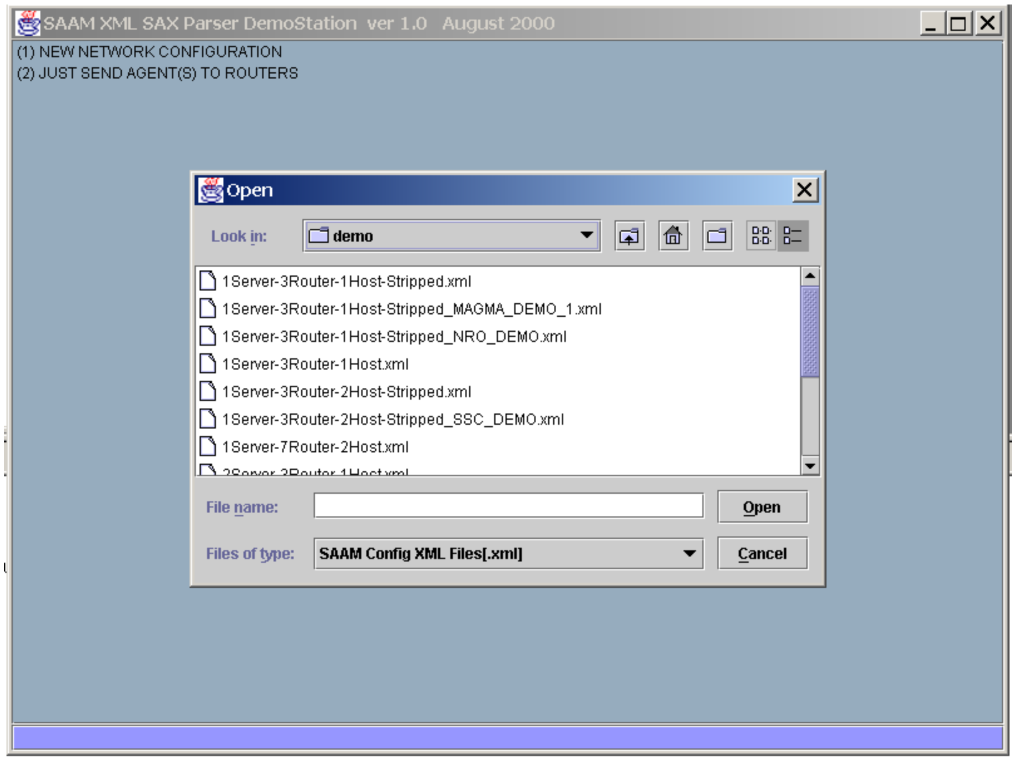

SAAM XML SAX Parser DemoStation ver 1.0 August 2000

(1) NEW NETWORK CONFIGURATION
(2) JUST SEND AGENT(S) TO ROUTERS

Input

Choose one of them? (1) or (2)

Figure 24. SAX Parser DemoStation Text Box Gui


SAAM XML SAX Parser DemoStation ver 1.0 August 2000

(1) NEW NETWORK CONFIGURATION
(2) JUST SEND AGENT(S) TO ROUTERS

Open

Look in: demo

1Server-3Router-1Host-Stripped.xml
1Server-3Router-1Host-Stripped_MAGMA_DEMO_1.xml
1Server-3Router-1Host-Stripped_NRO_DEMO.xml
1Server-3Router-1Host.xml
1Server-3Router-2Host-Stripped.xml
1Server-3Router-2Host-Stripped_SSC_DEMO.xml
1Server-7Router-2Host.xml
2Server-3Router-1Host.xml

File name:
Files of type: SAAM Config XML Files[.xml]

Figure 25. SAAM Configuration XML Files

The *MAGMAAdminController* GUI only appears on the *translator* that takes on the behavior of the MAGMA© server. The server is signified by an asterisk (*) before the router name at the top of the *translator* object. (See Figure 26.) The server *BannerFrame* GUI is another way that the server is specified. (See Figure 27.) The ellipse in Figure 28 highlights the pull down tab which will display the *magmaAdminController* GUI. From this pull down tab, there are two choices: DigestPIB or MAGMAController.

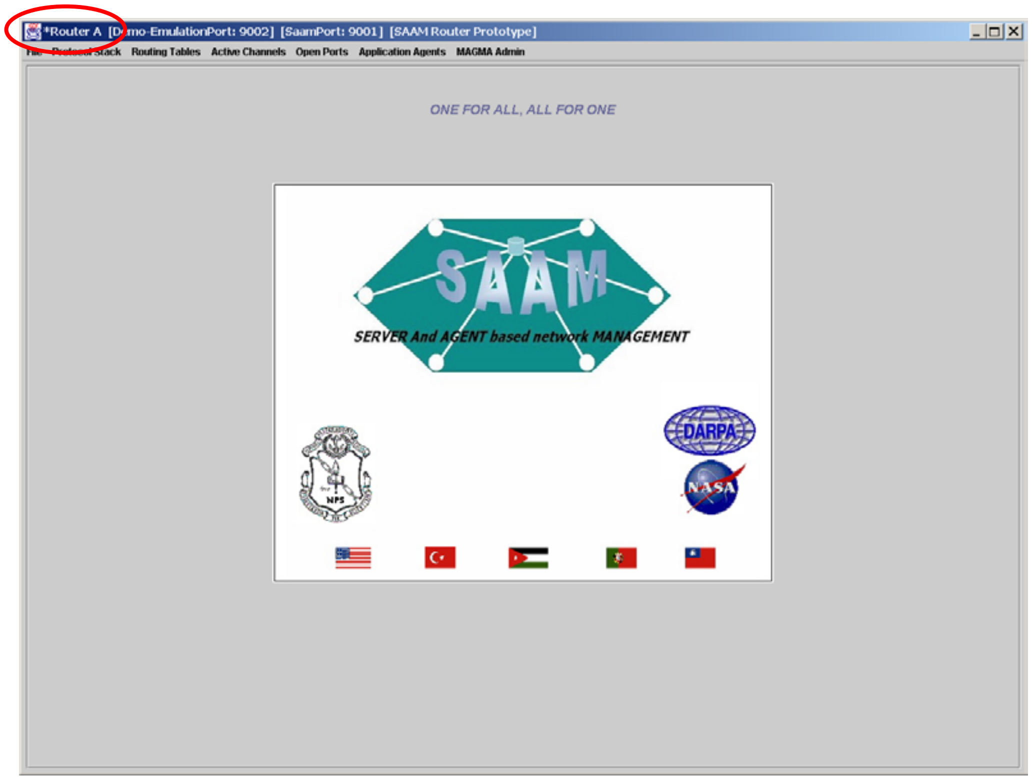
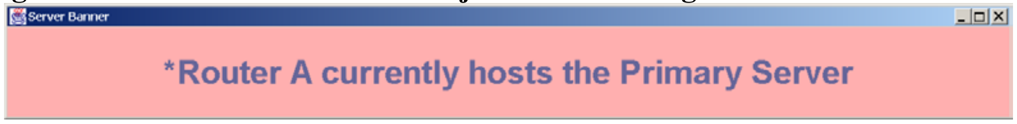


Figure 26. SAAM's Translator Object After Taking the SAAM Server Form



*Router A currently hosts the Primary Server

Figure 27. SAAM's Server Banner

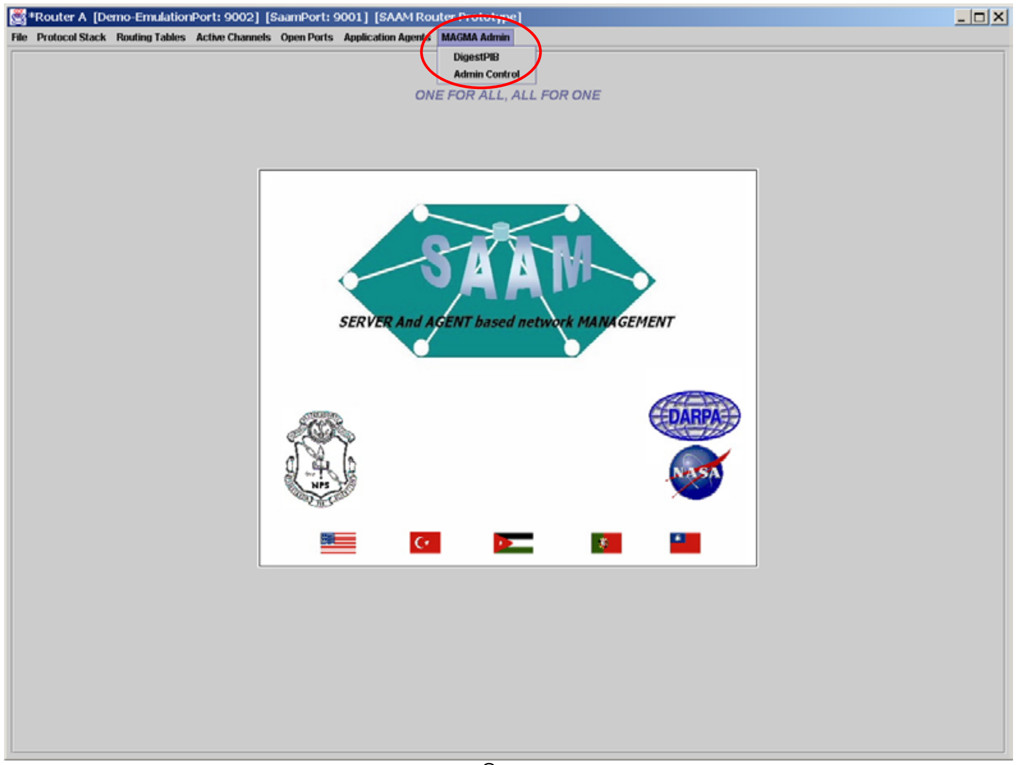


Figure 28. MAGMA$^{©}$ Menu's Pull Down Tab

The DigestPIB button gives the SAAM administrator the ability to observe what information is being placed in the DigestPIB. This button was originally placed in the code to allow the programmer to see that the information in the DigestPIB was the same before and after the compression/inflation methods. The Admin Control button in the MAGMA$^{©}$ Admin pull down tab gives the Administrator the control window used to send *MAGMATokens* messages to the other hosts. (See Figure 20.)

Once the SAAM Region is stood up, configured and running, the Admin Control button, along with the *magmaControlGui*, gives the SAAM administrator the capability to update the dormant MAGMA$^{©}$ agents with the static and dynamic state information or to manually move the MAGMA$^{©}$ server. In order to do this, the administrator needs to select the desired option by placing the mouse curse in the box next to the command and checking the box with a click of the mouse's left button. Next, the address of the target router must be selected in the combination box. Selecting the pull down arrow and maneuvering the mouse cursor over the desired IPv6 address can accomplish this. Also, if the desired IPv6 address is not available in the pull down menu, the administrator can type the required address in the text box. Once the correct IPv6 address is showing in the text box and the desired option(s) is(are) selected, the send button is depressed with the mouse cursor to deploy the *MAGMAToken* message.

E. EXPERIMENTAL RESULTS

In order to prove that this thesis produced code that supports its original intent, an experimental simulated architecture (See Figure 29) is prepared using an XML file simliar to the one shown in Figure 23. Next, the MAGMA© server is transferred from one host to the next and back again. In this simulation, the server moves from router A to router B to router C to router D, then back to router A. (See Figure 30.)

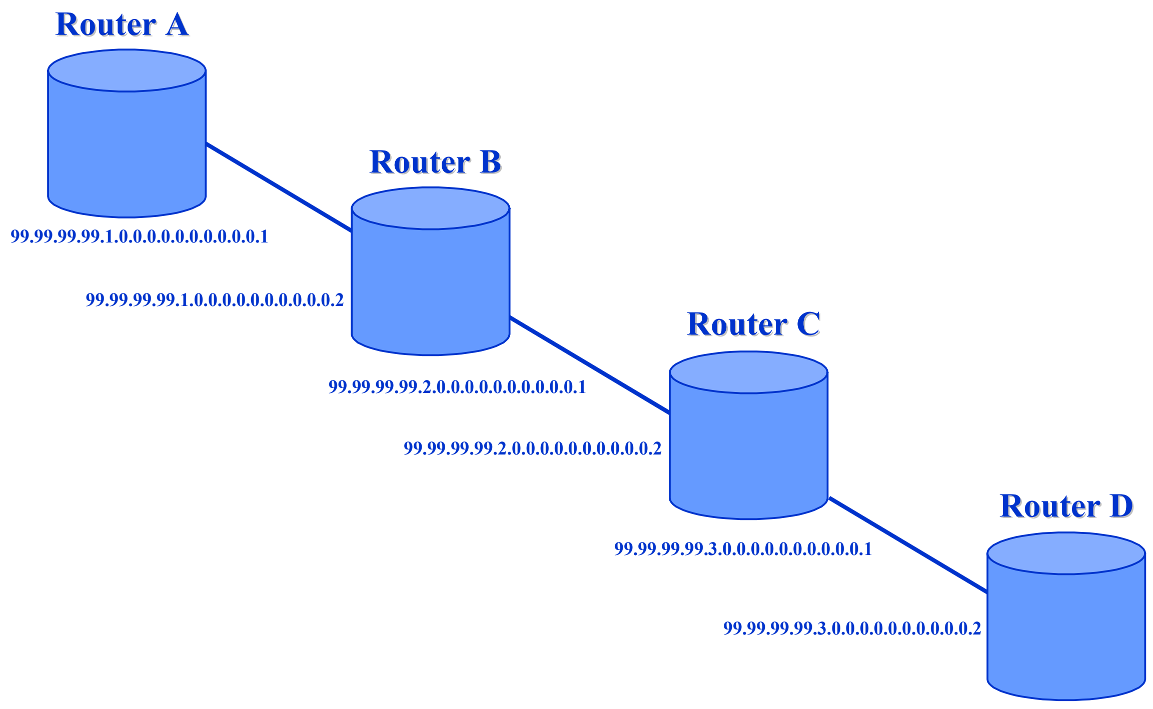


Figure 29. Experimental Simulation Architecture

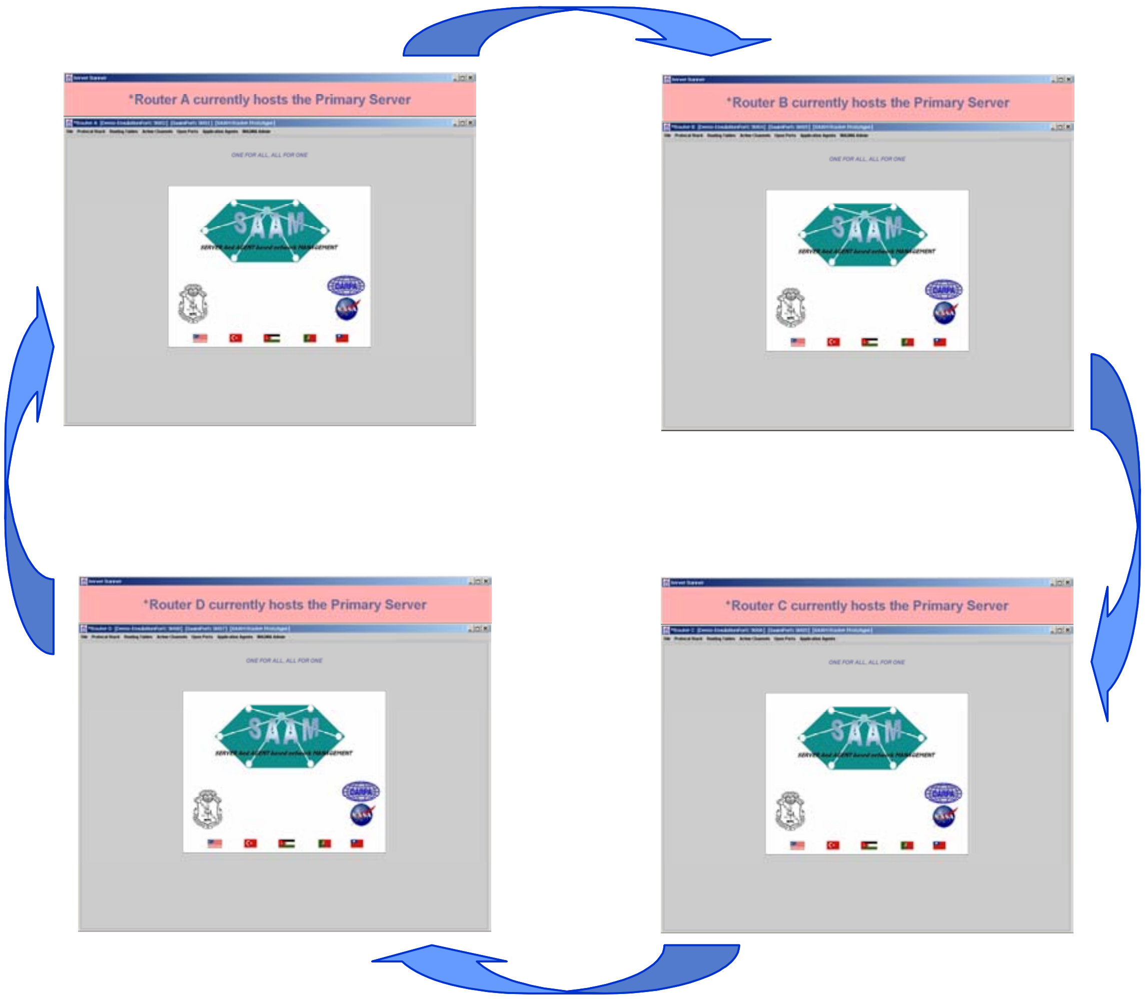

*Router A currently hosts the Primary Server
*Router B currently hosts the Primary Server
*Router C currently hosts the Primary Server
*Router D currently hosts the Primary Server
SAAM

Figure 30. MAMGA©Server Movement

In the following figures, you will see further demonstrations of MAGMA©'s movement through the SAAM Region. In Figure 31, the pull down menu circled shows the routing tables that are visable at each of the hosts. As the MAMGA© Server moves from one host to the next, a new Router-bound Channel Control Table is generated and placed in the pull down menu tab. If a table entry exists for the host receiving the new MAMGA©, the host will use its pre-existing table. The Router-bound Channel Control Table (See Figure 32) contains the next hop's interface number for each of the destination hosts. The next hop's interface number is listed in the left-most column. The "Goodness" column lets the observer know the cycle number of the last UCM. If 2 cycles go by and the entry does not get updated, that entry will be removed from the table.

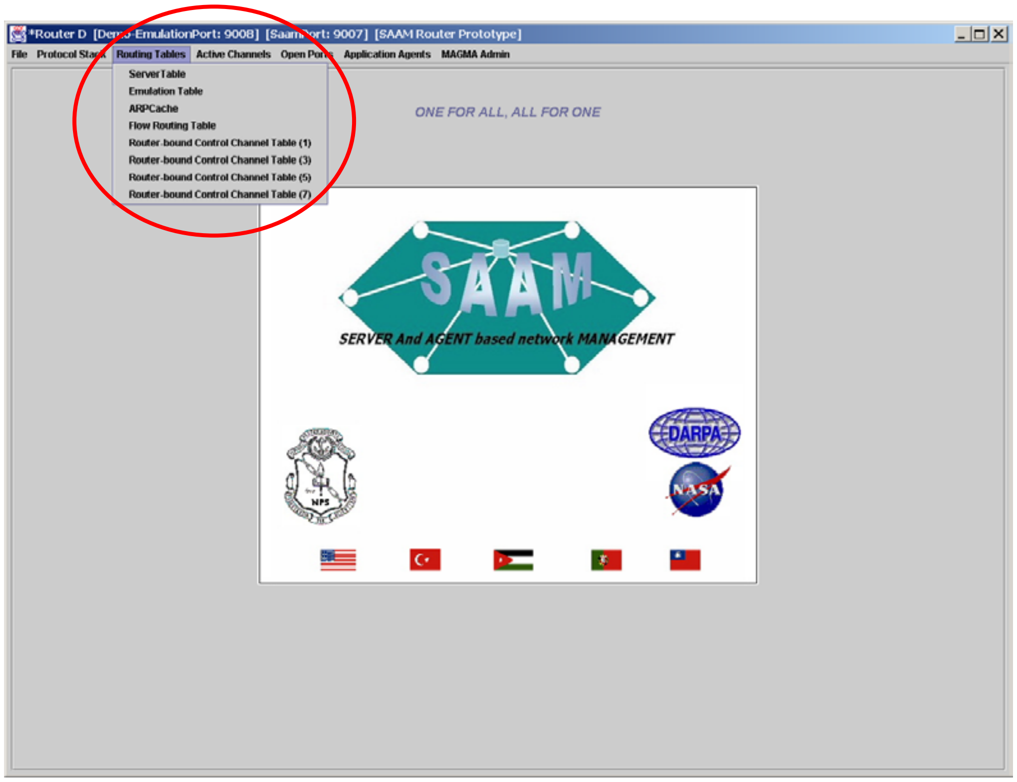
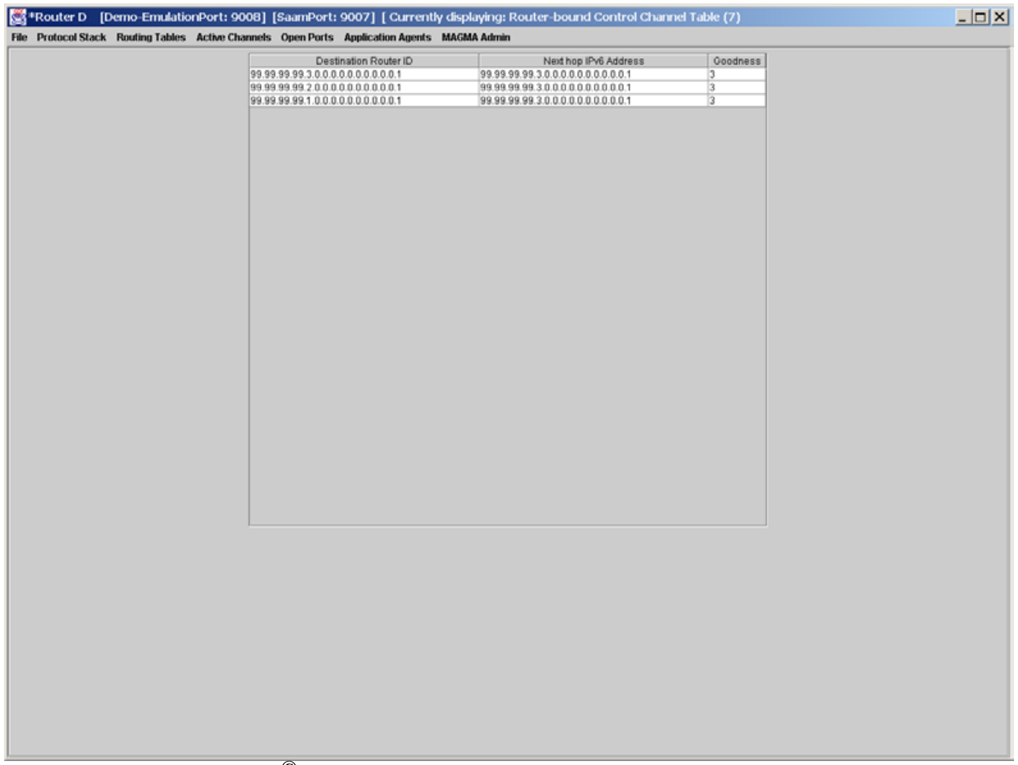

*Router D [Demo-EmulationPort: 9008] [SaamPort: 9007] [SAAM Router Prototype]
File
Protocol Stack
Routing Tables
Active Channels
Open Ports
Application Agents
MAGMA Admin
ServerTable
Emulation Table
ARPCache
Flow Routing Table
Router-bound Control Channel Table (1)
Router-bound Control Channel Table (3)
Router-bound Control Channel Table (5)
Router-bound Control Channel Table (7)
ONE FOR ALL, ALL FOR ONE
SAAM
SERVER And AGENT based network MANAGEMENT
DARPA
NASA

Figure 31. Routing Tables


*Router D [Demo-EmulationPort: 9008] [SaamPort: 9007] [Currently displaying: Router-bound Control Channel Table (7)
File
Protocol Stack
Routing Tables
Active Channels
Open Ports
Application Agents
MAGMA Admin

Destination Router ID	Next hop IPv6 Address	Goodness
99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1	3
99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1	3
99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1	99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1	3

Figure 32. MAMGA© Router-bound Control Channel Table for Router D

V. DIGEST PIB AND DATA COMPRESSION FOR REDUCING OVERHEAD

A. COMPRESSION TECHNIQUES

1. Overview Of Compression

Data compression algorithms were developed to help combat the huge amount of data storage needs in today's computers. Compression alleviates this problem and allows us to transfer data in a smaller form across a network with lower bandwidth. This is exactly the reason compression is being used within the SAAM project, providing condensed messages to efficiently allow servers the ability to configure themselves across the network with their state information. The following section provides an overview of possible compression algorithms that were considered for this research.

Though there are many algorithms available, they can be broken down into two types: lossless, which allows for perfect recall of data; and lossy, which allows for an approximate recall of data.

a. Lossless Algorithms

(1) Run Length Encoding (RLE). This is the easiest algorithm to implement with probably the worst compression around. The technique has its most famous implementation in some bitmap image files on the Window's platform. It takes runs of bytes and encodes them as a pair of bytes. In the worse case scenario, where each byte used to take just one byte, now each byte requires two. However, in very redundant files, it can use the space of two bytes to encode an arbitrary run of identical bytes.

(2) Huffman Coding. Huffman coding offers a very nice way to save space with very little effort. Huffman codes use either a tree or a lookup-table to find the codes used in place of the original data. The tree or lookup table can be either fixed in the compression algorithm, or dynamic. A dynamic algorithm will give better results for the current data set than a generic one. However, the dictionary of values used in this dynamic algorithm must also be sent with the message. Therefore, it is only useful for longer messages.

(3) LZ77. The most well-known lossless data compression techniques are based on LZ77, an algorithm invented by Lempel and Ziv in 1977. This

family of algorithms performs very well in compressing general data. A sliding window is used when compressing and decompressing the data. A 32kB window slides along behind the current byte, and if any repetition is noticed within the window of more than three bytes, a pair is output. The bytes need not be three or more of the same bytes, but rather identical sequences of bytes. When it encounters a sequence of identical bytes, the decompressor travels backwards through the stream bytes, outputs bytes, and then advances to the next value in the decoding stream.

(4) LZ78. LZ78 uses a dictionary and outputs nothing but indices into the dictionary. This can be implemented very quickly using a hash-table, but does not compress as well as LZ77. Every value must be stored in the dictionary. This is not as bad as it sounds though, since the algorithm that builds the dictionary is very deterministic. The dictionary does not have to be transferred; it refers back to previous values in the stream. LZ78 is most famous for its implementation in the (poor) Unix program Compress, known as LZC.

(5) Image Compression. The most popular lossless algorithms include ".gif" (pronounced "jif", per the ".png" spec) and ".png". ".gif" was developed years ago by CompuServe for transferring images. The current standard, ".bmp", had no compression. ".gif" uses an algorithm patented by Unisys to compress the data. The data was compressed, to begin with, by using a pallet; basically a dictionary. The pallet stores up to 256 RGB values. Each pixel points to one of the entries in the pallet. This offers decent size savings, but very little quality. There are no provisions for larger pallet sizes. The patented algorithm cannot be replaced with one in the public domain without breaking a lot of software licensing issues. The solution to these problems is ".png" (pronounced "ping").

".png" uses a better variant of LZ77 than ".gif", allowing for 48-bit color and levels of transparency. (".gif" offers two levels of transparency -- on or off. ".jpeg" offers no transparency. ".png" offers between 1 and 64k levels of transparency, though applications beyond 254 levels are not expected to be popular.) ".png" also offers better error checking than ".gif". This error checking allows software to determine if the file was corrupted while in transit, without requiring human intervention.

".png" was written as a response to the Unisys/CompuServe patent fiasco. All the algorithms used in ".png" are in the public domain and available for use without fear of lawsuits. It was developed by a team of roughly twenty individuals in comp.graphics, all experts in the field.

Acceptance of ".png" has been slow. Only recently has Microsoft Office and Adobe Photoshop offered support for ".png", while Netscape's Navigator has supported it for years. Also, GIMP has supported it for the past year or so. The one thing that has perhaps saved ".gif" from extinction is its ability to store multiple images in one file -- when viewed with the right application, it offers video! ".png" does not support this option, but a team is working on a version of ".png" called ".mng" which will support this functionality.

b. Lossy Algorithms

(1) JPEG. JPEG is the best-known lossy picture-compression algorithm. It throws away the values of some pixels, being content to calculate their values from the remaining pixels. It also depends on the photo having self-similarities. Most real-life photos have redundant color schemes and the subtle color gradients needed to make JPEG really shine. Usually, computer generated pictures do not compress well with JPEG. Since JPEG is lossy, it introduces *artifacts* into the final picture. These artifacts usually are not noticeable unless there are spaces of solid color in the picture. Photos do not have these spaces, so the artifacts are difficult to see unless the compression is turned up fairly high.

(2) MPEG. MPEG motion video heavily depends on JPEG. One thing to notice with motion video is that the picture changes very little from frame to frame. Most of the picture stays the same, especially when the video is very smooth. So, to capitalize on this fact, MPEG, and most other video compressors, store "key frames" every few frames that are pure pictures. These "key frames" are stored as JPEGs (in the case of MPEG), or other pure pictures. The next few frames are stored as differences to the key frame. The frequency depends on the compressor being used. Sometimes the key frame is fixed, for example every fifth frame. While other times, it is variable based on how far frame x has changed from the key frame.

The difference frames can be stored either with RLE encoded frames or with some other encoded format to help save on space and redundancy. MPEG offers more than just video compression. MPEG version 2 layer 3 audio (also known as MP3) offers 12:1 compression of audio. Audio compression has long been a "thorn in our side". One minute of CD-quality audio requires over one megabyte of storage, after compression. There are two popular ways to compress audio and both depend very heavily on the nature of the data.

MP3 took years of research. The researchers found many interesting things about how the human ear hears sound. They found an algorithm to describe what we can and cannot hear from a given input, then threw away everything we are not capable of hearing. Audiophiles may shudder at the thought; however, the audio quality is equivalent to CD quality and the size is about right -- one megabyte per minute of music. Lower qualities of sound can offer more impressive ratios.

(3) Wavelets. The other audio compression format currently being researched is *wavelets*. Wavelets allow for good modeling of the human voice by storing equations to describe little wavelets that approximate the full wave of audio. We can throw away a lot of data that is nonessential to understanding the content of this wavelet. The wavelet-based products to date are good ONLY for voice. Music wavelets are unrecognizable. This product is a fairly vertical application, useful for those who need to store nothing but voice, and a lot of it.

2. Java Compression Libraries and Classes

a. ZLIB

The ZLIB compression library was initially developed as part of the ".png" graphics standard (a variant LZ77 compression) and is not protected by patents. Java's goals in picking a suitable compression technique are as follows:

To define a lossless compressed data format that:

Is independent of CPU type, operating system, file system, and character set, and hence can be used for interchange;

Can be produced or consumed, even for an arbitrarily long sequentially presented input data stream, using only an a priori bounded amount of intermediate storage, and hence can be used in data communications or similar structures such as Unix filters;

Can use a number of different compression methods;

Can be implemented readily in a manner not covered by patents, and hence can be practiced freely; and

The data format defined by this specification does not attempt to allow random access to compressed data.

***b. Detailed ZLIB specification (Reference:* ZLIB Compressed Data Format Specification version 3.3 *RFC 1950*)**

(1) Byte Ordering. The following represents one byte; a box like this:

```
+==============+
|              |
+==============+
```

Bytes stored within a computer do not have a "bit order", since they are always treated as a unit. However, a byte considered as an integer between 0 and 255 does have a most- and least-significant bit, and since we write *numbers* with the most significant digit on the left, we also write *bytes* with the most-significant bit on the left. In the diagrams, we number the bits of a byte so that bit 0 is the least-significant bit:

```
+--------+
|76543210|
+--------+
```

Within a computer, a number may occupy multiple bytes. All multi-byte numbers in the format described here are stored with the MOST-significant byte first (at the lower memory address). For example, the decimal number 520 is stored as:

```
    0        1
+--------+--------+
|00000010|00001000|
+--------+--------+
 ^        ^
 |        |
 |        + less significant byte = 8
 + more significant byte = 2 x 256
```

(2) Data format. A ZLIB stream has the following structure:

```
  0   1
+---+---+
|CMF|FLG|   (more-->)
+---+---+
```

(if FLG.FDICT set)

```
  0   1   2   3
+---+---+---+---+
|     DICTID    |   (more-->)
+---+---+---+---+

+=====================+---+---+---+---+
|...compressed data...|    ADLER32    |
+=====================+---+---+---+---+
```

Any data, which may appear after ADLER32, are not part of the ZLIB stream.

(3) CMF (Compression Method and Flags). This byte is divided into a 4-bit compression method and a 4-bit information field, depending on the compression method.

bits 0 to 3	CM	Compression method
bits 4 to 7	CINFO	Compression information

(4) CM (Compression Method). This identifies the compression method used in the file. CM = 8 denotes the "deflate" compression method with a window size up to 32kB. This is the method used by GZIP and “.png”.

(5) CINFO (Compression INFOrmation). For CM = 8, CINFO is the base-2 logarithm of the LZ77 window size, minus eight (CINFO=7 indicates a 32kB window size). Values of CINFO above 7 are not allowed in this version of the specification. CINFO is not defined in this specification for CM not equal to 8.

(6) FLG (FLaGs). This flag byte is divided as follows:

bits 0 to 4	FCHECK	(check bits for CMF and FLG)
bit 5	FDICT	(preset dictionary)
bits 6 to 7	FLEVEL	(compression level)

(7) FCHECK. The FCHECK value must be such that CMF and FLG, when viewed as a 16-bit unsigned integer stored in MSB order (CMF*256 + FLG), is a multiple of 31.

(8) FDICT (Preset dictionary). If FDICT is set, a DICT dictionary identifier is present immediately after the FLG byte. The dictionary is a sequence of bytes which are initially fed to the compressor without producing any compressed output. DICT is the Adler-32 checksum of this sequence of bytes (see the definition of ADLER32 below). The decompressor can use this identifier to determine which dictionary the compressor has used.

(9) FLEVEL (Compression level). These flags are available for use by specific compression methods. The "deflate" method (CM = 8) sets these flags as follows:

0 - compressor used fastest algorithm
1 - compressor used fast algorithm
2 - compressor used default algorithm
3 - compressor used maximum compression, slowest algorithm

The information in FLEVEL is not needed for decompression. It is there to indicate if recompression might be worthwhile.

(10) ADLER32 (Adler-32 checksum). This contains a checksum value of the uncompressed data (excluding any dictionary data) computed according to the Adler-32 algorithm. This algorithm is a 32-bit extension and improvement of the Fletcher algorithm, used in the ITU-T X.224 / ISO 8073 standard. Adler-32 is composed of two sums accumulated per byte: s1 is the sum of all bytes; s2 is the sum of all s1 values. Both sums are done modulo 65521. s1 is initialized to 1, s2 to zero. The Adler-32 checksum is stored as s2*65536 + s1, with most significant byte first (network) order.

B. MAGMA© COMPRESSION

1. SAAM Compression Goals

One thing needed to support liquid software functionality is a suitable method to compress the DigestPIB class which provides the liquid server critical state information. It was determined that the Java libraries were the best place to get the compression algorithm, vice third party software, due to copyright infringement laws. ZLIB, a variant of LZ77, was used to further reduce the size of DigestPIB, beyond the logical compression described earlier.

Java also offers the GZIP library to perform compression and PKWARE's ".zip" file compression; but, to avoid the proprietary concerns, ZLIB was selected. Data collected during this section of coding research was each of the objects' size, compression ratio and compression rates achieved with the "Best Speed" and with the "Best Compression" techniques offered by the *Deflator* class within Java. The results are given and described later in Table 2 on page 64.

2. Java Compression Classes Used

a. Deflator

This class provides support for general-purpose compression using the popular ZLIB compression library. It is used to set the compression level via Deflator.Best_Speed or Deflator.Best_Compression static variables. Next, it is passed to the Deflator output stream for compression setup. The *Inflator* class is the reverse class of *Deflator*.

b. DeflatorOutputStream

This class implements an output stream filter for compressing data in the "deflate" compression format. The stream can be wrapped by a FileOutputStream or ObjectOutputStream to provide extra functionality. The InflatorInputStream is used to inflate and recover the data.

3. SAAM Code Modifications

The SAAM code was modified to set up streams, allowing the BasePIB and DigestPIB to be written to a file. This was done to obtain the objects' size, compression rates, and compression ratios. This information was used to determine the advantages of compressing BasePIB into a more compact form (DigestPIB) for later use in the MAGMA©. The following describes some insight that was gained during code development.

a. Serialization Issues

Implementing Serialization. Any object written to a stream must implement the Java.IO.Serializable interface. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable. In order to allow compression, to write to a file or to just write that object to a socket stream, the class and all of its inner classes must implement this interface.

Overriding *writeObject( )*. The ObjectOutputStream that was wrapped around the DeflatorOutputStream and FileOutputStream has a method available called *writeObject( )* that writes BasePIB or DigestPIB to a file. This method can be called by placing a *writeObject(ObjectOutputStream out) throws IOException{}* method within that class and its inner classes. This allows customization of the serialization. Placing *writeObject( )* in BasePIB was required to determine its size. Yet, BasePIB contained a *Server* object and a *SAAMGUI* object which did not need to be included in the state information. All non-static data in a class that implements *Serializable* should be included within the *writeObject( )* method that is inserted in that class.

Example within BasePIB:

```
BasePIB implements Serializable{…

private void writeObject(ObjectOutputStream out) throws IOException{
  ObjectOutputStream os = new ObjectOutputStream (out);

  os.writeObject(this.aPI);
  os.writeObject(this.htRouterIDtoNodeID);
  os.writeObject(this.htRouterIDtoNodeID);
  os.writeObject(this.htRouterInterfaceMap);
  os.writeObject(this.htInterfaces);
  os.writeObject(this.htPaths);
          os.writeObject(this.htUserSLSs);
  os.flush();

 }}
```

b. Server Object

The server object provides the *processLSA(LinkStateAdvertisement LSA)* and *processFlowRequest(FlowRequest flowRequest)* method calls. This is where the Link State Advertisements (LSAs) and flow requests update the BasePIB and DigestPIB and where the squeeze method was placed. The squeeze method is called after the BasePIB and DigestPIB are updated.

c. Creating a squeezeObjectToFile( ) method

The method signature below shows that this method will take any serialized object with its corresponding *Deflator* and squeeze it to the file name given.

```
public double squeezeObjectToFile(Object o, Deflater df, String fileName, int fileNo, double uncompressedFileSize)
```

d. Wrapping the DeflatorOutputStream within a ObjectOutputStream

The following shows how the streams are wrapped to create a object compressor stream that sends this object to a file:

```
fosC = new FileOutputStream(fileC.getName());
dosC = new DeflaterOutputStream(fosC,df);
oosC = new ObjectOutputStream(dosC);
```

e. Setup of Write Object to a File:

The following shows the *writeObject( )* and the *dosC.close* necessary for the deflator stream to write the appropriate tailing and checksum data at the end of this file:

```
oosC.writeObject(o);
dosC.close();
```

f. Calling the Squeeze method and setting up the Deflator:

The following shows the code snippet that sets up the deflators used and squeeze method calls after updating the BasePIB and the DigestPIB. This also occurs in the process flow request within the server object.

```
public void processLSA(LinkStateAdvertisement LSA){
  Deflater dfPIB = new Deflater(Deflater.NO_COMPRESSION);
  Deflater df0 = new Deflater(Deflater.NO_COMPRESSION);
  Deflater df1 = new Deflater(Deflater.BEST_SPEED);
  Deflater df2 = new Deflater(Deflater.BEST_COMPRESSION);

  basePIB.processLSA(LSA);
  digestPIB = newPIB.updateDigestPIB();
```

```
            squeezeObjectToFile(basePIB, dfPIB,"BasePIB_No_Compression", fileNo++,
uncompressedFileSize);
            squeezeObjectToFile(digestPIB, dfPIB,"BasePIB_No_Compression", fileNo++,
uncompressedFileSize);
            squeezeObjectToFile(newPIB, df1, "BasePIB_Best_Speed", fileBSNo++ ,
uncompressedFileSize);
            squeezeObjectToFile(newPIB,df2,"BasePIB_Best_Compression", fileBCNo++,
uncompressedFileSize);

            }
```

C. DIGEST PATH INFORMATION BASE

For liquid software to truly work, there needs to be a mechanism which will allow state information to be passed from one host to the next with minimal overhead. This mechanism needs to involve a container that will hold only the key essential volatile information. With this information, a new server will be able to stand-up and operate expeditiously, efficiently, and seamlessly. Its minute footprint will allow timely updates to all potential hosts with minimal bandwidth effects. This instrument has been deemed Digest Path Information Base, or Digest PIB.

1. Updating Base PIB and Digest PIB

To determine what information is critical to Digest PIB, it is important to understand how the Base PIB gets its information and how this information is then processed by the Digest PIB methods. Incoming data streams are received and handled by what is called SAAM's *Packet Factory*. These streams come in many forms, but Base PIB is only concerned with flow requests and link state advertisements (LSAs). When the *Packet Factory* receives this data, it sends it to the *Control Executive* for handling. The *Control Executive* takes these flow requests/LSAs and sends them to the Server-Agent for processing. They are further forwarded to the *Server* class which instantiates the Base PIB and Digest PIB objects. The Base PIB and Digest PIB objects are updated by one of two methods in the Server class, *processLSA( )* or *processFlowRequest( )*. It is in these methods where the *updateDigestPIB( )* method is called and the Base PIB is restructured. After creating a Digest PIB object and reorganizing the Base PIB, the dynamic state information forms the DigestPIB data structure and is placed into an Object Output Stream and Compressed. The *updateDigestPIB( )* method follows:

```
public synchronized DigestPIB updateDigestPIB() //updateDigestPIB added by Scott Margulis,
                                                //George Stavritis, and Bill Wilkins in
                                                //March 2001
{
  Integer[][]  digestAPI;  // A two-dimensional array of integers containing Path ID for
                    // the maximum bandwidth available between a source and destination router
                    // pair.  The array is set up as follows:
                    // digestAPI[sourceNode][destinationNode].

  DigestPIB digestPIB = new DigestPIB();

  digestAPI = new Integer[MAX_NODE_NUMBER][MAX_NODE_NUMBER];

  for (int i = 0; i < aPI.length; i++)     // From source router (i)
  {
   for (int j = 0; j < aPI[i].length; j++)     // To destination router (j)
   {

    if(i != j){
              digestAPI[i][j] = getWidestPath(i,j,aPI);// local method to find the path from the
                                               // source (i) to the destination (j) which
                                               // contains the largest available bandwidth.
                                               // The method returns the PathID as an Integer
     } // End if
    else {
      digestAPI[i][j] = new Integer(0);
    } // End else

   } // end of j loop
  } // end of i loop

  digestPIB.setDigestAPI(digestAPI);
  digestPIB.setDigestHtInterfaces(htInterfaces);
  digestPIB.setDigestHtPaths(getReducedHtPaths(htPaths));

  return digestPIB;

}//end updateDigestPIB
```

2. Path Information Base vs. Digest PIB

Currently, the Server and Agent-based Active network Manager (SAAM) server agent keeps state information in a java class called Base PIB (Path Information Base). Kuo Dao-Cheng and John H. Gibson designed the Base PIB in a September 2000 joint thesis entitled, "Design of a Dynamic Management Capability for the Server and Agent-based Active network Management (SAAM) System to Support Requests for Guaranteed Quality of Service Traffic Routing and Recovery." This Base PIB class contains several variables, hashtables, vectors, arrays and inner-classes including the following:

aPI[source][destination][hop count], a three-dimensional array of hashtables which contains all the *PathIDs* from each source router to each destination router segmented by hop count; *htRouterIDtoNodeID*, a hashtable keyed by the router's largest IPv6 address which associates the routers with a unique node ID for the life of the Base PIB while the router's ID is an IPv6 address that could change; *htNodeIDtoRouterID*, a reverse look-up table of the *htRouterIDtoNodeID* which is keyed by *NodeID*, an Integer Object; *htRouterInterfaceMap*, a hashtable, keyed by the router's IPv6 address which contains all the active interfaces for the corresponding router and a hashtable keyed by the interface's IPv6 address which maps to the IPv6 address byte array, allowing rapid search, insert, and removal of interfaces from a router; *htInterfaces*, a hashtable keyed by the interface's IPv6 address which holds an *InterfaceInformationObject*; *htPaths*, a hashtable keyed by *iPathID*, an Integer Object, which enables rapid access to all paths assigned to the SAAM Region; and *htUserSLSs*, a hashtable used for Differentiated Services which is keyed by the *userID* and stores an SLS object (the customer's service level agreement contract).

"Most volatile" is one criterion for evaluating and choosing which data should be included in the Digest PIB. Another one is "most urgent"; that is, whether the information is immediately needed by a new server to handle flow requests. Other less urgent information could be built/collected by the new server in a less timely manner without interrupting and slowing down the functionality of the new server.

Of all the aforementioned tables/variables, it was determined that only those, which are most volatile, should be included in the Digest PIB. From the Base PIB format, a variation of the three-dimensional *aPI* (array of Path Information), the *htInterface* hashtable, and the *htPaths*' hashtable were used. Algorithms were written in Java to place only the critical information from these three structures into the Digest PIB. These algorithms include *getWidestPath( )* for digestAPI, a two-dimensional array of *PathIDs* (which are Integer Objects); and *getReducedHtPaths( )*, a hashtable that is keyed by Integer Objects of *PathIDs* and contain critical paths in the SAAM Region.

3. getWidestPath( )

This algorithm was designed to store only the path from a source router to a destination router that has the largest available bandwidth. In Base PIB, a three-dimensional array of hashtables stores all the available paths from a source to destination router containing from 1 to 8 hops. For a 30 node SAAM Region, this could equate to 7,200 hashtable entries in *aPI[][][]* (array of Path Information). There is also a variable called *htPaths*, a hashtable that is keyed by the *iPathID* and contains valuable path information, including an inner class called *PathQoS* (Path Quality of Service), which holds the available bandwidth for that given path. The available bandwidth, for the purpose of *PathQoS*, is defined as the smallest available bandwidth found on each of the links that make up the path from a particular source router to its corresponding destination router. *getWidestPath( )* takes each of these hashtables, for all hop counts, from a source to destination router and searches for the path with the largest available bandwidth, or the "biggest pipe." This java method returns the *iPathID* of the path with the biggest pipe and stores it in a two-dimensional array of Integer Objects. *getWidestPath( )* method follows:

```
public Integer getWidestPath(int source, int dest, Hashtable aPI[][][]) //getWidestPath added by Scott Margulis,
                                                                        //George Stavritis, and Bill Wilkins in
                                                                        //March 2001
{
    Hashtable table = new Hashtable();   // Stores the Hashtable for a
                                         // given source/dest/hop count from
                                         // the aPI Hashtable

    Path currentPath = null;             // Stores a Path Object from the htPaths
                                         // Hashtable.

    Integer bestPath = new Integer(0);   // Keeps track of the PathID of the path
                                         // with the largest available bandwidth

    int maxBandwidthAvailable = 0;       // A temporary storage bin for the largest
                                         // bandwidth.  This number is compared with
                                         // each path's bandwidth to determine which
                                         // path has the largest available bandwidth

     for (int k = 1; k < aPI[source][dest].length; k++)  // search each path for all hop counts to
                                                         // determine which one has the biggest pipe,
                                                         // regardless of hop count
      {

      table = aPI[source][dest][k];        // create a copy of the aPI hashtable
     Enumeration enum = table.elements( );   // enumerate the hashtable aPI copy

        // For each of the PathIDs, get the reference path to determine the max bandwidth available
        while (enum.hasMoreElements( ))  // search through each path
        {
          Integer currentPathID = (Integer) enum.nextElement();

          currentPath = (Path) htPaths.get(currentPathID);  // get the Path object for the currentPathID
          if (currentPath != null){
          if (currentPath.objPathQoS[INT_SERV].availableBandwidth( ) > maxBandwidthAvailable) {
                  maxBandwidthAvailable = currentPath.objPathQoS[INT_SERV].availableBandwidth( );
                  bestPath = currentPathID;}
          }
        } // end while loop

      }// end of k loop
```

```
        return bestPath;   // looking for maximum bandwidth available
                           // once maximum bandwidth available is found,
                           // store only that PathID.

    } // End of getWidestPath
```

4. getReducedHtPaths( )

This algorithm was designed to reduce the number of paths stored in Base PIB's *htPaths*' hashtable. When Base PIB creates its *htPaths* hashtable, it stores all the path information for all paths within the SAAM Region. For a SAAM Region containing 30 nodes, this equates to 870 paths with all their information stored in a hashtable. Not all of these paths that traverse the SAAM Region contain data flows. These inactive paths are not critical to expeditiously standing up a new SAAM server-agent. Therefore, the *getReducedHtPaths( )* method enumerates the *htPaths* hashtable and searches through each of the paths, assigning only those paths that contain an *iNewFoldID* (the number of flows that transverse the path) greater than 0, thus removing all inactive paths. *getReducedHtPaths( )* method follows:

```
public Hashtable getReducedHtPaths( Hashtable htPaths) // getReducedHtPaths added by Scott Margulis,
                                                        // George Stavritis, and Bill Wilkins in March 2001
{
   Hashtable table = new Hashtable();     // creates hashtable to store
                                          // a copy of the htPaths hashtable


   Enumeration enum = htPaths.keys();      // enumerates the hashtable to observe its contents



   if (!enum.hasMoreElements()){
           // For each of the PathIDs, get the iNewFlowID number to determine if the path
           // starts and ends at an edge router.
   while (enum.hasMoreElements())  // search through each path
     {
       Integer currentPathID = (Integer) enum.nextElement();


       Path currentPath = (Path) htPaths.get(currentPathID);  // gets the Path object for a
                                                               // given PathID number


       if (currentPath.getFlowID( ) != 0) //== 0)  // looks at the iNewFlowID number
                                                   // to determine if the path begins
                                                   // and ends at an edge router
         {
                 table.put(currentPathID, currentPath);
         } // End if

     } // end while loop


   return table;      // removes those paths that do not begin or end with an
                      // edge router

 } // End of getReducedHtPaths
```

D. TEST BED TOPOLOGY

One topology of SAAM network was used to test the new code written for the DigestPIB and compressed DigestPIB. The topology is defined in Figure 33 on page 63. This model contains 6 nodes, one server and five routers. Routers A, B, and C are generating (sources) and consuming traffic (sinks); D is a sink only; and E is neither a source nor a sink.

The original SAAM code placed an artifical limitation on the number of nodes allowed to operate at a time. This was due to resource allocation issues. Therefore, only 6 nodes were used for this testbed topology. In this simulation, all 6 instances of SAAM routers were running on the same machine, a highly resource intensive process. Every run processed differently and each machine that this was tested on required a different time amount of time to observe the topology building up. At times, it took more than 10 minutes for the topology test bed to build.

An Extensible Markup Language (XML) file was used to insert the topology information into the SAAM model. The only detail that caused some delays in making SAAM understand the topology was the fact that the current SAAM server code was designed to recognize a router by its interfaces IP with the greatest IPv6 address. This allowed different port numbers to distinguish different traffic being sent to the same destination IPv6 address, not different interfaces. To compose the XML file, the XML editor "XML-SPY" was used.

Agents were used to simulate the network traffic. The following shows the different flow patterns. The letter to the left of the arrow signifies the router name of the traffic source, whereas the letter to the left of the arrow specifies the traffic's sink.

A → B
A → C
B → D
B → A
B → C
C → B
C → A

F has no agent.

Links 2 and 6 were omitted.

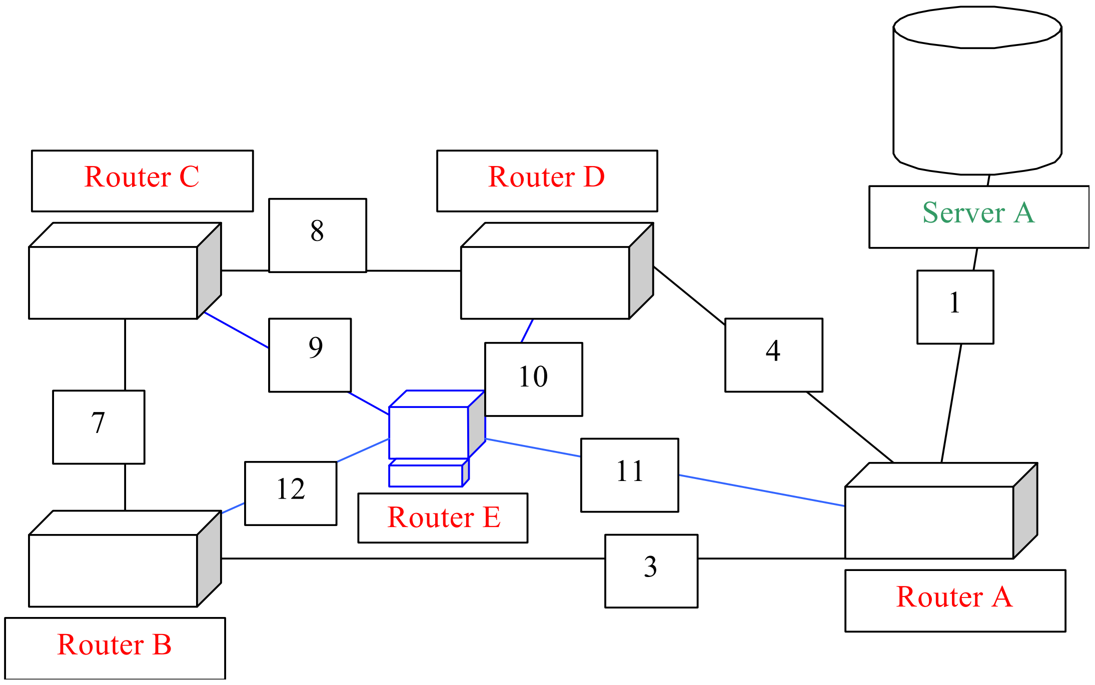


Figure 33. Test Bed Topology to test the Digest PIB and Compression Algorithms.

E. EXPERIMENTAL RESULTS

Table 2 on page 64 shows the results of the logical reduction gained by pulling out the important data from BasePIB. This results is a 11:1 reduction for the uncompressed DigestPIB. Java "Best Compression" further shrinks this file to a 126:1 compression, making it feasible for DigestPIB's to travel the network without overstressing network bandwidth resources.

Object	Base PIB	Digest PIB		
Compression Type	Uncompressed	Uncompressed	Best Speed	Best Compression
Size (Bytes)	310,037	27,324	2982	2454
Compression Rate(Bytes/Sec)	815,886	683,100	99,400	49,080
Overall Compression Ratio	1:1	11:1	104:1	126:1

Table 2. Test Bed Compression Results.

Table 2 data is an average of the Path Information Base (PIB) and the Digest PIB obtained when running the *updateDigestPIB( )*, *getWidestPath( )*, *getReducedHtPaths( )*, and *squeezeObjectToFile( )* methods with a 5 router and 1 server test bed topology; using CBR, Poisson, packet train and self-similar packet transmittal. From these results, it was determined that "Best Speed" compression was the best compression algorithm to use. Where as "Best Compression" provided less network overhead, "Best Speed" only provided a 21.5% increase in overhead over "Best Compression", while reducing the time of compression by 50%.

VI. CONCLUSIONS AND RECOMMENDED FUTURE WORK

A. SYNOPSIS AND CONCLUSION

This thesis' goal was to provide a more robust, survivable SAAM architecture, protecting the SAAM Region from system degradation due to system faults or malicious attacks. This was to be accomplished through the use of mobile agents. The concept was to play a shell game with the SAAM server by altering the position of the server from one router to the next. This movement was to take place in a random fashion, such that if a hacker wanted to disrupt SAAM service, the server would move to another hosts. Also, if a hacker was successful in shutting down the SAAM server or if an operating server had a malfunction, like the Token Ring protocol, a new server would be generated and take over the server functions.

So what was accomplished in this thesis? The many advances in this field of liquid software were covered in previous chapters. MAGMA was able to provide a method to stand up a SAAM server and maneuver this server around a SAAM Region. This was accomplished through instantiation of a server agent on each of the routers during initial configuration. These server agents were placed in a dormant state until stimulated by a *MAGMATokens* message with the CHANGEMagma flag set. Once the server was stood up on the new router, it was able to continue operating and processing network traffic flows. It was demonstrated that this server could be transferred from one router to the next and then back to the original router.

Due to time constraints and the many obstacles that were overcome in executing this thesis project, MAGMA©'s liquid software did not achieve 100 percent of its original goals. However, this opens the door for many new investigations in future thesis work. These new concepts are further discussed below. In the end, new ground was broken in the exploration of mobile agents and survivable networking.

B. FUTURE WORK

Due to the limited time afforded at this institution, many of the potential tasks that MAGMA© can performed must be left up to further theses. The following sections briefly touch on each of these determined areas. However, this is only a sample of potential areas of research and not an all-inclusive listing. The topics are only limited by the creative minds of future researchers.

1. Create Code to Move the Server Automatically, in a Random Motion.

The original goal of this thesis was to provide for a random walk or shell game with the MAGMA© server to provide security to the critical node, the MAGMA© server. The idea was to create a table, which contained all of the potential future MAGMA© servers. Then, at a random time interval, use some sort of random number generator or possibly one of the digits from the computer's clock to pick a new server, using the number to select a hashtable's key to provide the next server host. Another concept is to devise a new protocol, which provides a random movement MAGMA© server.

2. Clean-up the MAGMA© GUI and provide a way to change font sizes, font color, font style, background color, etc.

The current MAGMA© Admin Controller is very plain. There is no mechanism in place to change the font size, style, or color. Furthermore, the addresses in the combination box are predefined at system startup and do not reflect the current routers in the system. Future thesis work may want to provide a GUI enhancement and also provide a more dynamic way of updating the combo box with only those nodal addresses available for MAGMA© server placement.

3. Provide the MAGMA© GUI with a separate container so that it can be processed as a Servlet, giving it the ability to be viewed from whatever workstation the SAAM Administrator is working at, while having the functionality to view the entire network.

This new found technology is designed to operate an entire SAAM routing Region. The SAAM Region is designed to contain up to 40 nodes. With each of these nodes spread out over a vast geographical area, there needs to be a centralized point to administer the server which will be residing on one of the routers. By moving the controlling GUI to a servelet, MAGMA© will have the ability to be controlled from any host which is connected to the SAAM Region.

4. Come Up With Automatic Sensors To Calculate And Quantify Gains and Overhead Added To System, In Fault Tolerance And In System Security

In order to provide the statistical data necessary to prove that MAGMA©'s mobile agent based approach is a viable solution to security and fault tolerance, there needs to be a way to quantify the legitimacy of this new technology. To provide these figures, there needs to be mobile agent probes which can deploy to each of the hosts, gather necessary information, and report back to a centralized collector database.

5. Build in an information assurance mechanism to ensure that all intercommunication messages between the MAGMA© server and each router host are not altered or replayed.

While an original goal of MAGMA© was to provide for security of a SAAM Region, it is based on an assumption that the messages that are being sent from one host to the next is, in itself, secure. If a hacker is able to determine the structure of messages or capture a CHANGEMagma message and retransmit a copy of it, he may be able to disrupt the SAAM Region or gain control of the next MAGMA© server's location. To prevent this from happening, all administrative traffic being sent from one host to the next should be encrypted with one form of hashing function such as Message Digest 5 (MD5), Triple Data Encryption Standard (DES), Rivest-Shamir-Adelman (RSA), or Secure Hash Algorithm (SHA) for information assurance purposes.

6. Provide a protocol to automatically handle a hacker's attack.

To further harden the security of the SAAM Region, a protocol should be put into place which will detect a malicious attack and thwart it by automatically moving the MAGMA© server out of harm's way.

7. BETA test MAGMA©.

The programmer was limited in his time and resources. Where as this software was tested for accuracy and functionality, there needs to be a more rigorous, structured BETA testing plan developed and executed to prove that the changes made by this thesis are precise and complete.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDICES

A. MAGMA© SOURCE CODE

1. saam\agent\router\LinkStateMonitor.java

```
//23Oct2001[Margulis] - removed firstTime class variable and its use in generateInterfaceSA method

                    /**Modified by Margulis [sam]
                     * This procedure was modified to provide a mechanism for the boolean, firstCycle, to
                     * be keep in scope for delivery to the private procedure, generateInterfaceSA.
                     * @param firstCycle boolean variable used to let the LinkStateMonitor know that
                     *           this is the initial cycle call to set up the interface and not
                     *           an interface that was added after the SAAM Region was stood up.
                     * @return myISA    an Interface State Advertisement
                     */
                    //added boolean parameters [sam]
                    public InterfaceSA getISA(boolean firstCycle)
                    {
                     generateInterfaceSA(firstCycle);
                     return myISA;
                    }//end getISA

                    /**Modified by Margulis [sam]
                     * Creates the interfaceLSA for this interface and send it to LSA generator.
                     * @param firstCycle boolean variable used to let the LinkStateMonitor know that
                     *           this is the initial cycle call to set up the interface and not
                     *           an interface that was added after the SAAM Region was stood up.
                     * @return void
                     */
                    //added boolean parameters [sam]
                    private synchronized void generateInterfaceSA(boolean firstCycle)
                    {
                     //gui.sendText("generating interface lsa");
                     //first append the header section of the InterfaceLSA


                     cycleCount++; //[TW] remove -- simulate interface failures
                      //byte ip[] = null;

                     //[sam]
                     //This disables Troy's Hack to shutdown an interface
                     byte ip[] = {99,99,99,99,99,99,99,99,99,99,99,99,99,99,99,99};

                     //byte ip[] = {99,99,99,99,2,0,0,0,0,0,0,0,0,0,0,2};
                      //byte ip[] = {99,99,99,99,4,0,0,0,0,0,0,0,0,0,0,2};
                      //byte ip[] = {99,99,99,99,4,0,0,0,0,0,0,0,0,0,0,1};

                     IPv6Address testAdd = null;
                     try
                     {
                      testAdd = new IPv6Address(ip);
                     }
                     catch (Exception e)
                     {
                      System.out.println(e);
                     }
                     if (cycleCount == 4 && parent.getID().getIPv6().equals(testAdd))
                     {
                      gui.sendText("Disabling interface " + parent.getID().getIPv6());
                      parent.shutdownInterface(control);
                     }
                     // end [TW]
```

```
   InterfaceInfo id = parent.getID();
   IPv6Address interfaceAddress = id.getIPv6();
   if (firstCycle)
   {
     //created InterfaceSA type "ADD"
     myISA = new InterfaceSA(interfaceAddress, id.getBandwidth(), id.getSubnetMask());

   }
   else
   {
     //check for packets received since last cycle
       boolean trafRcvd = this.parent.trafficReceived();
       if (trafRcvd) //tvw switch this to (trafRcvd) for normal
     {

       gui.sendText("Traffic received on " + parent.getID().getIPv6()); //[TW]
       //created InterfaceSA type "UPDATE"
       myISA = new InterfaceSA(interfaceAddress);

       Vector SSAVector = new Vector();

       //for every Service level queue we need to know the servicelevel data
       for (int sl = 0; sl < Interface.numberOfServiceLevels; sl++)
       {
         PriorityQueue slq = (PriorityQueue) parent.SLQueuesVector.get(sl);

         short delay = slq.getDelay();
         short lossRate = slq.getLossRate();
         byte utilization = slq.getUtilization();

         System.out.println("\nInterface " + parent.getInstanceNumber() +
                           " Service Level " + sl + " performance: " +
                          "\n  average utilization = " + utilization + " (%)" +
                          "\n  average packet delay = " + delay + " (milliseconds)" +
                          "\n  average loss rate = " + lossRate + " (%)\n");

         ServiceSA ssaUtilization = new ServiceSA((byte) sl, ServiceSA.UTILIZATION_TYPE,
                                                  utilization);
         SSAVector.add(ssaUtilization);
         ServiceSA ssaDelay = new ServiceSA((byte) sl, ServiceSA.DELAY_TYPE, delay);
         SSAVector.add(ssaDelay);
         ServiceSA ssaLossRate = new ServiceSA((byte) sl, ServiceSA.LOSSRATE_TYPE, lossRate);
         SSAVector.add(ssaLossRate);
       }//end for
       myISA.insertServiceSAs(SSAVector);
     }
     else
     {
       messageIndex++;
       gui.sendText("No traffic received on " + parent.getID().getIPv6()); //debug
       //notify Interface no traffic rcvd
       parent.silentInterface();
       //generate silent message
       myISA = new InterfaceSA(interfaceAddress, messageIndex);
     }

   }

   started = false;

 }//end method generateInterfaceLSA()
```

2. saam\agent\router\LsaGenerator.java

```
//23Oct2001[Margulis] - getLSA, PerformLSACycle, GenerateExtraLSA altered to fix previous bug

/**Modified by Margulis [sam]
 * This procedure was an entire re-write by Prof Xie and incorporates performLSACycle
 * called by AutoConfiguration Executive to pull LSA packets.
 * Also added was the boolean parameter. This method was altered to pass the boolean variable on
 * to the LinkStateMonitor.getISA method.
 * @param firstCycle boolean variable used to let the LinkStateMonitor know that
 *                 this is the initial cycle call to set up the interface and not
 *                 an interface that was added after the SAAM Region was stood up.
 * @return LinkStateAdvertisement  a status of the interface
 */
public LinkStateAdvertisement getLSA(boolean firstCycle)
{
  Vector isaVector = new Vector();
  Enumeration enum = interfaces.elements();
  while (enum.hasMoreElements())
  {
    Interface link = (Interface) enum.nextElement();
    LinkStateMonitor monitor = link.getMonitor();
    InterfaceSA isa = monitor.getISA(firstCycle);
    //check the status of the interface
    if (!link.getState())
    {
      //interface is down
      //check if the LSA is sent before
      if (!link.isISASent())
      {
        generateExtraLSA(link);
        isa.setInterfaceSAType(InterfaceSA.REMOVE);
        checkIdChange(isa.getInterfaceIP());
        link.setISASent();
      }
    }

    isaVector.add(isa);
  }//end while

  //construct the header of LinkStateAdvertisement for this router
  LinkStateAdvertisement lsa = new LinkStateAdvertisement(routerId);

  lsa.insertInterfaceSAs(isaVector);
  //gui.sendText("the number of interface is: " + );

  if (idChanged)
  {
    decideRouterID();
    idChanged = false;
  }

  // if the this is server LSA should also be sent to itself
  if (controlExec.isServer())
  {
    long ts = System.currentTimeMillis();
    byte[] data = lsa.getBytes();

    byte numbMsgs = 1;

    data = Array.concat(numbMsgs, data);
    data = Array.concat(PrimitiveConversions.getBytes(ts), data);

    int channelP = ProtocolStackEvent.TO_PACKETFACTORY_CHANNEL;

    ProtocolStackEvent pe = new ProtocolStackEvent(
        this.toString(),
        this,
        channelP,
        data);

    try{
      controlExec.talk(pe);
    }
    catch (ChannelException ce){
      System.out.println(ce);
    }
  }

  return lsa;

}//end getLSA()


/**Modified by Margulis [sam]
 *  This function sends an extra LSA
 *  This function was shorted and its functionality was added to getLSA method.
 *  Professor Xie removed several lines so this method will have to be closely
 *  compare in the next integration upon detecting an interface failure.
 */
private synchronized void performLSACycle()
{
  Enumeration enum = interfaces.elements();
  while (enum.hasMoreElements())
  {
    //check the status of the interface
    Interface link = (Interface) enum.nextElement();
    if (!link.getState())
    {
      //interface is down
      //check if the LSA is sent before
      if (!link.isISASent())
      {
        generateExtraLSA(link);
        link.setISASent();
      }
    }
  }//end while

}//end performLSACycle()
```

3. saam\agent\server\ServerAgent.java

```
//08Nov2001[Margulis]-Minor modifications of several methods

//[sam]
import saam.net.IPv6Address;
import java.lang.Byte;

                    //[sam]Modified by Margulis
                    /** -crcp
                     * parseMessage() receives inbound messages via its argument. It is called by a thread
                     * running in the ThreadQueuedMessageProcessor (TQMP) superclass.
                     *  This method is now required to implement call-back for the new MessageProcessor
                     * superclass. Inbound messages are buffered in a queue maintained by the separate
                     * thread. This method is called from the superclass's thread whenever messages are
                     * available and subsequently dequeued by the thread.
                     * Every instance of the new TQMP subclasses gets its own "hidden" threadedQeueue
                     * which is created, started, and managed by the TQMP superclass.  This common code
                     * is easily changed in the superclass, and affects all subclasses unless designers
                     * override with their own specializations. The (superclass) common code is declared
                     * final for now to prevent ACCIDENTAL override by designers unfamiliar with this
                     * mechanism.
                     *
                     *
                     * @param   obj  Generic Message object dequeued from superclass' thread
```

```
  */
 protected void parseMessage(Object obj)  //-crcp change from private to protected
 {
   //Note: Modified IBPF encodes these as generic "Message" objects -crcp
   gui.sendText(">>>>>>>>Entering parseMessage()"); //diag -crcp
   Message message = null; //-crcp

                                   //for modified messages
   gui.sendText("check if GENERIC MSG: name is " + obj.getClass().getName() +
      ", byte [0] = " + ((Message) obj).getBytes()[0]
      + ", getType = " + ((Message) obj).getType() +
      ", typeName= " + Message.lookUpMessageByType(((Message) obj).getType())); //diag
//    if (((Message) obj).getBytes()[0] == Message.LSA) //now we can check either way
   if (((Message) obj).getType() == Message.LSA)
   {
   // Construct an LSA based on byte array contained in generic Message object.
   // This has been loosely termed -- late message "binding" -crcp
   // -crcp    Actually, this is merely delayed
   // Message-subclass instantiation.  Dynamic binding is more automatic, and
   // requires message differentiation to occur in the IBPF, which we are trying
   // to get away from.  True late-binding would allow us to do something like:
   // myServer.processLSA((LinkStateAdvertisement) obj); But that would require
   // that the IBPF go back to differentiating messages and calling the NEW operator
   // against the constructors of the differentiated message types, and then sending
   // up these messages (the old way) to the appropriate MessageProcessor.

       gui.sendText("Inside new LSA processing" +
          "...Instantiating LSA message from byte []");

       message = new LinkStateAdvertisement(((Message) obj).getBytes());
        // First time we fully reconstruct LSA message (call LSA constructor)

       myServer.processLSA((LinkStateAdvertisement) message);


       if (!initHeartBeatRan)
       {
         myServer.initHeartbeat();
         initHeartBeatRan = true;
       }
       myServer.updateServerStrings();

   }
   else if (((Message) obj).getType() == Message.CONFIGURATION) //-crcp new way now
   {
     message = new Configuration(((Message) obj).getBytes());
     gui.sendText("Received Message: " + ((Configuration) message));
     gui.sendText("Calling Server method: processConfiguration()");
     message = new Configuration(((Message) obj).getBytes());
     myServer.processConfiguration((Configuration) message);
   }

   //[sam]
   // "&& controlExec.isServer()" was added to prevent server agents that are
   // not active servers from processing flow requests
   else if ((((Message) obj).getType() == Message.FLOW_REQUEST)&& controlExec.isServer())    {
     try
     {
       message = new FlowRequest(((Message) obj).getBytes()); //-crcp call its ctor
     }
     catch (Exception ex)
     {
       gui.sendText("ERROR instantiating FlowRequest:" +
          ex.getMessage());
     }
     gui.sendText("\nReceived Flow Request:\n" + ((FlowRequest) message));
     gui.sendText("Forward to server.processFlowRequest()");
     myServer.processFlowRequest((FlowRequest) message);
   }
```

```
    else    //-crcp  For unmodified messages
    {
      //[sam]
      // "&& controlExec.isServer()" was added to prevent server agents that are
      // not active servers from processing flow requests
      if ((((Message) obj).getType() == Message.FLOW_REQUEST)&& !controlExec.isServer()){

          gui.sendText("Received a flow request, but I am not the active Server so I am not processing it!!");
      }//end if

      gui.sendText("Must be old-style message");
      message = (Message) obj;
    }

//-crcp The following should all go away after all message conversions are complete:
/********-crcp The remaining messages will not work until made generic. TBD********/
    try
    {
      //the following two "else if" cases are added by Efraim KATI
      if (message.getType() == (byte) Message.HEARTBEAT_QUERY)
      {
        gui.sendText("Received Message: " + ((HeartbeatQuery) message));
        gui.sendText("Calling Server method: processHeartbeatQuery()");
        myServer.processHeartbeatQuery((HeartbeatQuery) message);
      }

      else if (message.getType() == (byte) Message.HEARTBEAT_RESPONSE)
      {
        gui.sendText("Received Message: " + ((HeartbeatResponse) message));
        gui.sendText("Calling Server method: processHeartbeatResponse()");
        myServer.processHeartbeatResponse((HeartbeatResponse) message);
      }

      //added altinkaya March 00
      else if (message.getType() == Message.PROBE_RESULT)
      {
            gui.sendText("Received Message: " + ((ProbeResult) message));
            gui.sendText("Calling Server method: processProbeResult() ");
            myServer.processProbeResult((ProbeResult) message);
         }
    }
    catch (Exception e)
    {
      message = null;
    }

  }//end parseMessage()

  //[sam]
  /**Added by Margulis
   * This method was added to allow for the Server banner to reappear after the Magma server
   * moved from one host to another then back to a host with had already had an operating
   * Magma server on it
   * @param boolean  setFlag allows the calling method specify whether or not the initial HeartBeat
   *                message was called.
   */
  public void setInitHeartBeatRan (boolean setFlag)
  {
    initHeartBeatRan = setFlag;
  }//end setInitHeartBeatRan()
```

4. saam\control\AutoConfigurationExecutive.java

```
//23Oct2001[Margulis] Altered createNewServerInformation and sendOneUCM methods
//        to fix previous bug

                    //[sam]
                     /**Modified by Margulis
                      * The call to setFirstCycle to true was added to account for newly stood up Magma
                      * servers.  This allows them to be added to the Router Bound Control Channel Tables.
                      *
                      * Learning a server from DCM this method places an entry in the servertable
                      * for that server and also creates a RouterBountControlChannel for this server
                      * @param int pid root path id of server
                      * @param IPv6Address serverID router id of server
                      * @return void
                      */
                      public synchronized void createNewServerInformation(int serverFlowLabel, IPv6Address serverId)
                      {
                        // Add it to table
                        RouterBoundCtrlChTable newTable =
                                    new RouterBoundCtrlChTable(serverFlowLabel, 1711, controlExec);

                        ServerTableEntry newEntry = new ServerTableEntry(serverFlowLabel, newTable, serverId);
                        controlExec.getServerTable().add(newEntry);
                        ServerInformation serverInfo = new ServerInformation(serverFlowLabel);
                        serverTable.put(new Integer(serverFlowLabel), serverInfo);

                        //[sam]
                        serverInfo.setFirstCycle(true);
                      }// end createNewServerInformation()

                    //[sam]
                     /**Modified by Margulis
                      * The call to getLSA was modified to incorporate the fact of whether this
                      * is the firstCycle of the DCM in order to know whether or not to added this
                      * MAGMA server to the Router Bound Control Channel Table. The flag is then set
                      * to false so that the MAGMA server will not be duplicated upon the next DCM.
                      *
                      * Method initiates sending a UCM message to the parent of the router
                      * @param int serverFlowLabel flow label used by server
                      * @return void
                      */
                      private void sendOneUCM(int serverFlowLabel)
                      {
                        ServerInformation sInfo =
                          (ServerInformation) serverTable.get(new Integer(serverFlowLabel));

                        try
                        {
                    //Xie-nov01:

                          Message message = (Message) (new FlowRoutingTableEntry(serverFlowLabel + 1));
                          FlowRoutingTableEntry entry = (FlowRoutingTableEntry)
                              controlExec.getRoutingAlgorithm().getFlowRoutingTable().query(message);

                          if (entry != null)
                          {

                          sInfo.stopGlobalWaitTimer();

                          byte [] bytes = null;
                          byte numOfMsgs = 0;

                            // This is not the server's host router; so forward a UCM to parent
                            IPv6Address nextHopToServer = entry.getNextHop();

                            Interface srcInterface = Interface.getMatchInterface(
                                        controlExec.getInterfaces(), nextHopToServer);
```

```
      IPv6Address srcInterfaceAddr = srcInterface.getID().getIPv6();

      // Finally add own router id to reachable router list
      sInfo.addToReachableRoutersBuffer(controlExec.getRouterId());

      UCM ucm = new UCM(serverFlowLabel + 1,
                        srcInterfaceAddr,
                        sInfo.getReachableRoutersBuffer().size(),
                        sInfo.getReachableRoutersBuffer(),
                        sInfo.getLastSeqNum());

      bytes = Array.concat(bytes, ucm.getBytes());
      numOfMsgs++;
      gui.sendText("A new UCM message is created");

      // Retrieve local LSA and piggyback it to UCM
      LsaGenerator generator = controlExec.getGenerator();
      bytes = Array.concat(bytes, generator.getLSA(sInfo.isFirstCycle()).getBytes()); //[sam]
      numOfMsgs++;
      gui.sendText("Local LSA is fetched and piggybacked to UCM");

      // Retrieve LSAs of children and piggyback them to UCM
      byte [] lsaBytes = sInfo.retreiveLSABytes();
      if (lsaBytes != null)
      {
        int index = 0;
        numOfMsgs += lsaBytes[index++];
        bytes = Array.concat(bytes,
                  Array.getSubArray(lsaBytes, index, lsaBytes.length));
        gui.sendText("LSAs of children are retrieved and piggybacked to UCM.");
      }

      controlExec.sendUCM(numOfMsgs, bytes, serverFlowLabel + 1, nextHopToServer);
      gui.sendText("The compound UCM/LSA message is sent to parent");

      sInfo.setDCMReceived(false);

    // Reset the reachable router list
    sInfo.getReachableRoutersBuffer().clear();
      }
     else
      {
      // Just deliver local LSA to server agent
      gui.sendText("This is host router of the target server; just deliver local LSA to server");

      LsaGenerator generator = controlExec.getGenerator();
      LinkStateAdvertisement lsaMessage = generator.getLSA(sInfo.isFirstCycle());

      MessageEvent mEvent = new MessageEvent(
                              this.toString(),
                              controlExec,
                              controlExec.SAAM_CONTROL_PORT,
                              (Message) lsaMessage);

      controlExec.receiveEvent(mEvent);
      }
    }
    catch (Exception e)
    {
     gui.sendText("***Problem sending UCM/LSAs " + e.toString());
    }

    sInfo.setFirstCycle(false);
  }//end sendOneUCM()
```

5. saam\control\ControlExecutive.java

```
//23Oct2001[Margulis] - Several methods altered to incorporate Magma
//21Sep2001[Margulis] - Added code to update MainGui for changing Server Host location
//17Jul2001[Margulis] - Added methods to incorporate Magma Messages and parsing of message.
//20Apr2001[Margulis] - Added MagmaAdminGui handle to controlExec

import saam.agent.server.*;
 import saam.util.MAGMAAdminGui;

                    /**
                     * The ControlExecutive registers with itself as a MessageProcessor capable of
                     * processing messages of the following types.
                     */
                    private static final String[] messageTypes = //-crcp No need. Remove dependencies/delete
                    {
                      "saam.message.InterfaceInfo",
                      "saam.message.ServerID",
                      "saam.message.FlowResponse",
                      "saam.message.DemoHello",
                      "saam.message.DCM",
                      "saam.message.UCM",
                      "saam.message.ParentNotification",
                      "saam.message.TimeScale",
                      "saam.message.ServiceLevelSpec",
                      "saam.message.ResourceAllocation",
                      "saam.message.InterfaceShutdown",
                      "saam.message.MAGMATokens"    //Margulis [sam]
                     };

                    // New message type "Message.MAGMA_TOKEN" was added to account for the MagmaToken Message
                    private static final byte [] myMsgs =  //-crcp new MP registration stuff
                    {
                   //    Message.INTERFACE_ID, //-crcp deleted ServerID.java
                      Message.FLOW_RESPONSE,
                      Message.DEMO_HELLO,
                      Message.DCM,
                      Message.UCM,
                      Message.PARENT_NOTIFICATION,
                      Message.TIME_SCALE,
                      Message.SLS_TABLE_ENTRY,
                      Message.RESOURCE_ALLOCATION,
                      Message.INTERFACE_SHUTDOWN,
                      Message.MAGMA_TOKEN   //Margulis [sam]
                    };

                    //[sam]
                    //This instantiates a new MAGMA serverAgent in each host upon initialization of the host.
                    private ServerAgent magmaAgent = new ServerAgent();

                    /**
                     * Instantiates and sets up communication with all Objects that are necessary
                     * to allow the ControlExecutive to start receiving ResidentAgents and Messages.
                     */
                    public ControlExecutive (EmulationTable emTable)
                    {
                     mainGui = new MainGui(this, "SAAM Router Prototype");
                     gui = new SAAMRouterGui(toString());

                     mainGui.addSAAMRouterGui(gui);

                     transportInterface = new TransportInterface(this);
                     //for receiving inbound packets
                     inputPacketFactory = new PacketFactory(this);
                     outputPacketFactory = new PacketFactory(mainGui);

                     this.emTable = emTable;

                     arpCache = new ARPCache();
```

```
  //[sam]
  //This installs the magmaAgent, but leaves it dorminant in resident memory.
  magmaAgent.install(this);

  serverTable = new ServerTable(this);
  autoConExec = new AutoConfigurationExecutive(this);

  //this should eventually become a ResidentAgent
  routingAlgorithm = new RoutingAlgorithm(this, arpCache);

  //Here is where the ControlExecutive registers itself as a MessageProcessor

  registerMessageProcessor(myMsgs,this); //-crcp new MP registration stuff

  playerId = "Translator"; // Makes uninitialized player easy to spot  -crc

  /* Enable Talking on the channel that the Translator is listening on for
     router status updates.*/


  try
  {
     addTalkerToChannel(this, ROUTER_STATUS_CHANNEL);
  }
  catch (ChannelException ce)
  {
     gui.sendText(ce.toString());
  }

  // added to send packets to Transport Interface
  int channel_ID =
      ProtocolStackEvent.FROM_APP_TO_TRANSPORTINTERFACE_CHANNEL;
  try
  {
    addTalkerToChannel(this, channel_ID);
    gui.sendText("Talk on " +
       (channel_ID <= MAX_PORT? "port: " : "channel: ") + channel_ID);
  }
  catch (ChannelException ce)
  {
    gui.sendText(ce.toString());
  }//try-catch

  try
  {
    //Get ownership of the SAAM_CONTROL_PORT so other applications
    //cannot.  When the TransportInterface sees a packet destined
    //for this port, it will not forward the packet directly on
    //the SAAM_CONTROL_PORT, rather, it will forward the packet
    //to the PacketFactory on the ProtocolStackEvent.TO_PACKETFACTORY_CHANNEL
    monitorPort(this, SAAM_CONTROL_PORT);
    gui.setTextField("Monitoring emulated port " + SAAM_CONTROL_PORT);
  }
  catch (PortAccessDeniedException pade)
  {
    gui.sendText(pade.toString());
  }

  mainGui.updateDisplay();

 }//ControlExecutive()

 //[sam]
/**Add by Margulis
 * Registers a MAGMAAdminGui object with the player's mainGui via
 * the player's controlExecutive.  This became a requirement when MIPH
 * design went single JVM-capable and non-final static members became verboten.
 * @param   newGui  A MAGMAAdmin Control Gui
 */
```

```
public void addMagmaGui(MAGMAAdminGui newGui)
{
  mainGui.addMAGMAAdminGui(newGui);
}//end addMagmaGui()

 //[sam]
 /**Added by Margulis on November 06, 2001
  * Removes the Magma pull down menu from the router MainGui
  * @param  none
  * @return void
  */
public void removeMagmaMenufromMainGui( )
{
  mainGui.updateDisplay();
  mainGui.resetCurrentDisplay();
} //end removeMagmaMenufromMainGui()


//[sam]
/**Added by Margulis
 * Renames the Player Identifier String signifying the programmed role of this player
 * according to the demoStation initializing message, DemoHello.
 * @param  newID the new playerID value
 * @return void
 */
public void setPlayerId(String newID)
{
  this.playerId = newID;
}//end setPlayerId()

//[sam]
/**Added by Margulis
 * This method was added to allow for the Server banner to reappear after the Magma server
 * moved from one host to another then back to a host with had already had an operating
 * Magma server on it
 * @param boolean  setFlag allows the calling method specify whether or not the initial HeartBeat
 *                message was called.
 */
public void setInitHeartBeat(boolean setFlag)
{
  magmaAgent.setInitHeartBeatRan(setFlag);
}//end setInitHeartBeat()

/**Margulis added this in October 2001
 * To indicate whether router is hosting active server
 * @param setFlag boolean variable to indicate whether router is hosting active server
 */
public void setIsServerFlag(boolean setFlag)
{
  isServer = setFlag;
}//end setIsServerFlag

//[sam]
/**Added by Margulis
 * Returns the message received in the processMessage method.
 * @return the message received in the processMessage method.
 */
public synchronized byte[] getMessage()
{
  return bytes;
}//getMessage()

//[sam]
 /**Modified by Margulis
  * This method was modified to incorporate the handling of inbound MAGMA Messages by the
  * control executive.
  *
  * This method contains the logic needed by the ControlExecutive
  * to process the Messages it is registered to process.
  * @param message The subclass of saam.message.Message to be processed.
  */
 //synchronized
public void processMessage(Message message)
{
  //[sam]- debug statement
  System.out.println("ControlExec.processMessage: In the Beginning");

  // -crcp    latest message binding.  DCM specificity is OK/required here.
  this.bytes = message.getBytes();

  switch (message.getBytes()[0])
  {
    case Message.DCM:
      message = new DCM(message.getBytes()); //convert generic into specific
      gui.sendText("\nGot a DCM.");
      DCM receivedDcm = (DCM) message;
      gui.sendText("  Forward the message to AutoConfigureExecutive.");
      autoConExec.processDCM(receivedDcm);
      break;

    case Message.PARENT_NOTIFICATION:
      message = new ParentNotification(message.getBytes());
      gui.sendText("\nGot a Parent Notification.");
      ParentNotification pn = (ParentNotification) message;
      gui.sendText("  Forward the message to AutoConfigureExecutive.");
      autoConExec.processPN(pn);
      break;

    case Message.UCM:
      message = new UCM(message.getBytes()); //convert generic to specific
      gui.sendText("\nGot a UCM.");
      UCM ucm = (UCM) message;
      gui.sendText("  Forward the message to AutoConfigureExecutive.");
      autoConExec.processUCM(ucm);
      break;

    case Message.DEMO_HELLO:
      message = new DemoHello(message.getBytes());
      gui.sendText("\nGot a demoHello message from demo station.");
      DemoHello hello = (DemoHello) message;
      byte numberOfInterfaceInfos = hello.getNumberOfInterfaceInfos();
      Vector helloInterfaces = hello.getInterfaceInfos();
      for (int i = 0; i < numberOfInterfaceInfos; i++)
      {
        try
        {
          InterfaceInfo id = (InterfaceInfo) helloInterfaces.get(i);
          standUpInterface(id);
        }
        catch (Exception e)
        {
          gui.sendText("Exception: " + e);
        }
      }

      helloMessageReceived = true;
      updateRouterStatus();
      mainGui.updateDisplay();

      //add code to extract 3 and update the players GUI -crc
      playerId = ((DemoHello) message).getPlayerId() + "  ";
      mainGui.updateFramePlayer(); //just signal mainGui to update frame
      break;

    case Message.TIME_SCALE:
      {
        byte [] bytes = message.getBytes();
        message = new TimeScale(message.getBytes());
        gui.sendText("\n received TimeScale Message.");
        TimeScale ts = (TimeScale) message;
        timeScale = ts.getTimeScale();
      gui.sendText("TS = " + timeScale);
     }
     break;

    case Message.FLOW_RESPONSE:
     {
      byte [] bytes = message.getBytes();
      gui.sendText("received FlowResponse Message.");
      FlowResponse response = null;
      try
      {
       response = new FlowResponse(bytes);
      }
      catch (UnknownHostException uhe)
      {
       gui.sendText("Error instantiating FlowResponse Msg:" +
         uhe.getMessage());
      }
      gui.sendText("\nGot a flow response.");
      long timeStamp = response.getTimeStamp();
      gui.sendText("  timeStamp = " + timeStamp);
      int flowLabel = response.getFlowLabel();
      gui.sendText("  flow label = " + flowLabel);
      ResidentAgent requestor =
          (ResidentAgent) transportInterface.getFlowRequestors()
            .get(new Long(timeStamp));
      gui.sendText("  forwarding it to Requester: " + requestor);
       // Add flow into assignedFlows list; [GX]: need to remove it later!
      if (requestor != null)
      {
       assignedFlows.put(new Integer(flowLabel), requestor);
       requestor.receiveFlowResponse(response);
      }//else do nothing
     }
     break;
   //[sam]
   //This code handles the MAGMA Token messages sent by a MAGMA Server by passing the message
     case Message.MAGMA_TOKEN:  // added by Margulis in July 2001
     {
      gui.sendText("In ControlExec.processMessage: case Message.MAGMA_TOKEN");
      System.out.println("In ControlExec.processMessage: case Message.MAGMA_TOKEN");

      server.processMagmaMessage(this, bytes);

      // This may not be looking at the right byte. Further tests need to be completed to determine
      // if this is the right byte to be looking at.
      if (bytes[20] == MAGMATokens.CHANGEMAGMA) {
      server.killServerBanner();  // removed to debug kill banner issue
      }//end if

     }
     break; //end case Message.MAGMA_TOKEN
    default:
     gui.sendText("***Unconverted, nongeneric Message processing**");
     break;

  }//end switch

  //unconverted messages


  if (message.getBytes()[0] == Message.SLS_TABLE_ENTRY) //ServiceLevelSpec
  {
    gui.sendText("\nGot a ServiceLevelSpec");
    //Router do nothing yet
    //Future work
  }
  else if (message.getBytes()[0] == Message.INTERFACE_SHUTDOWN) //Interface Shutdown
  {
    gui.sendText("\nGot an Interface Shutdown message");
```

```
      //find the interface that is failing
      Enumeration enum = interfaces.elements();
      InterfaceShutdown failure = (InterfaceShutdown) message;
      while (enum.hasMoreElements())
      {
        Interface iFace = (Interface) enum.nextElement();
        InterfaceInfo id = iFace.getID();
        IPv6Address ip = id.getIPv6();

        if (ip.equals(failure.getIP()))
        {
          iFace.shutdownInterface(this);
          gui.sendText("  Interface with address " +
                           ip.toString() + " has been shut down.");
          return;
        }
      }
      System.out.println("Shutdown failed:  There is no interface with IP "
                         + failure.getIP().toString());
    }

  }//end ProcessMessage

  //[sam]
  /**Added by Margulis, Modified by Xie in November 2001
   * An asterisk (*) will be appended to front of player id when the router
   * starts to host the primary server.
   *
   * @param  boolean status, indicates whether this router hosts the primary SAAM server
   * @return void
   */
  public void setPrimaryServerStatus(boolean status)
  {
    this.isPrimaryServer = status;
    if (status == true)
    {
      playerId = "*" + playerId;
    }
    else
    {
      // Need to remove the asterisk
      playerId = playerId.substring(1);
    }

    mainGui.updateFramePlayer();

  } //end setPrimaryServerStatus
```

6. saam\control\MainGui.java

```
//23Apr2001[Margulis] - Added the magmaAdministratorMenu and all things associated with it
//23Apr2001[Margulis] - Added Magma Administrator Option to Gui

  //[sam]
  JMenu magmaAdministratorMenu;
  ImageCanvas imagePanel;
  Container contentPane;

  Vector magmaAdministratorToDisplay = new Vector(); // Margulis [sam]

  private Hashtable instancesOfMAGui = new Hashtable(); // [sam]
  private Vector titlesOfMAGuis = new Vector(); // [sam]
```

```
 //[sam]
 /**Added by Margulis
  * This method adds the required titles to the MAGMAAdmin pull down tab
  * @param MAGMAAdminGui  The MAGMAAdmin Controller Gui
  */
 public void addMAGMAAdminGui(MAGMAAdminGui gui)
 {
   titlesOfMAGuis.add(gui.toString());
   instancesOfMAGui.put(gui.toString(), gui);
 }//end addMAGMAAdminGui()

 //[sam]
/**Margulis added on November 6, 2001 ...Scott's Hack
 * This was added to remove the Magma JPanel on the old server
 * @param none
 */
 public void resetCurrentDisplay( )
 {

   //  Need to clean up dangling guis for garbage collection

       setTitle(controlExec.getPlayerId() + " " + frameStatusMessage +
         " Currently displaying: Resetting Router Display" ); // mod for status -crc
       contentPane.setLayout(new FlowLayout());
       contentPane.add(imagePanel);
       setContentPane(new JScrollPane(imagePanel));
       imagePanel.setVisible(true);
       imagePanel.validate();
       validate();

//    displaySAAMLogo(); Xie-nov01
//    setVisible(true);

 }//end resetCurrentDisplay( )

 //[sam]
 /**Modified by Margulis to incorporate MAGMAAdmin Gui's and Panels
  * This method creates the file menu for all the pull down menus.
  */
 void createFileMenu()
 {
   menubar = new JMenuBar();
   fileMenu = new JMenu("File");
   exit = new JMenuItem("Exit");
   exit.addActionListener(
                         new ActionListener()
                         {
                           public void actionPerformed(ActionEvent ae)
                           {
                             System.exit(0);
                           }
                         }
                        );
   fileMenu.add(exit);
   protocolStackMenu = new JMenu("Protocol Stack");
   routingTableMenu = new JMenu("Routing Tables");
   openChannelMenu = new JMenu("Open Channels");
   activePortMenu = new JMenu("Active Ports");
   slsTableMenu = new JMenu("SLSTable");  //Henry
   flowTableMenu = new JMenu("Flow Tables");  //Henry
   appAgentsMenu = new JMenu("Application Agents");  //Fatih
   magmaAdministratorMenu = new JMenu("MAGMA Admin"); //Margulis[sam]

   menubar.add(fileMenu);
   menubar.add(protocolStackMenu);
   menubar.add(routingTableMenu);
   menubar.add(openChannelMenu);
   menubar.add(activePortMenu);
   menubar.add(slsTableMenu);  //Henry
   menubar.add(flowTableMenu);  //Henry
   menubar.add(appAgentsMenu );  //Fatih

   setJMenuBar(menubar);
 }//end createFileMenu()

 //[sam]
 /**Modified by Margulis
  * This method was modified to ensure that only the simulated server host
  * receives the pull down menu with the MAGMAAdmin Controller.
  *
  * This checks to see if any of the menu items need to be changed and
  * updates any of these changes, such as new agents.
  */
 synchronized void updateDisplay()
 {
   updateRoutingTables();
   updateProtocolStackObjects();
   updateChannels();
   updateRouterSLSTable();  //Henry
   updateApplicationAgents(); // Fatih

   //[sam]
   try
   {
     //add this menu option for the server only
     if (controlExec.isServer())  //add this menu option for the server only
     {
       menubar.add(magmaAdministratorMenu);  // Margulis
       setJMenuBar(menubar);
     } //end if
     else
     {
       menubar.remove(magmaAdministratorMenu);
       setJMenuBar(menubar);
     }//end else
   }//end try
   catch (Exception e)
   {
     System.out.println(e);
   }//end catch

   //[sam]
   updateMagmaAdministrator();

 }//end updateDisplay


 //[sam]
 /**Added by Margulis
  * This method was modified to ensure that only the simulated server host
  * receives the pull down menu with the MAGMAAdmin Controller.
  *
  * This checks to see if any of the menu items need to be changed and
  * updates any of these changes, such as new agents.
  */
 void updateMagmaAdministrator()
 {

   for (int i = 0; i < titlesOfMAGuis.size(); i++){
     String thisTitle = (String)titlesOfMAGuis.get(i);
     if(!magmaAdministratorToDisplay.contains(thisTitle)){

       JMenuItem item = new JMenuItem(thisTitle);
       magmaAdministratorMenu.add(item);
       item.addActionListener(new ActionListener(){
                           public void actionPerformed(ActionEvent ae){

                             setCurrentDisplay(
                              (MAGMAAdminGui) instancesOfMAGui.get(ae.getActionCommand()) ); //-crcy
```

```
                              }  //end new ActionListner
                             } //end item.addActionListner
                            );
      magmaAdministratorToDisplay.add(thisTitle);
    }  //end if
   } //end for
  } //end updateMagmaAdministrator
```

7. saam\control\PacketFactory.java

```
//04Jun2001[Margulis] - altered several lines of code in parseMessage to allow for MAGMAToken and
//        KILL_BANNER messages


    /**Modified by Margulis [sam]
   * This method is used to extract the individual Class
   * Objects that are represented in the packet.  These Class
   * Objects are either of type 0 (ResidentAgent) or 1 (Message).<p>
   * If a ResidentAgent is received, a Class Object is created
   * that represents the agent.  That Class Object is then sent to
   * the ControlExecutive for screening and agent instantiation.<p>
   * If a Message is received, that Message is instantiated and sent
   * to the ControlExecutive for further processing.
   */
  private void processPacket(byte [] packet)
  {
   //[sam]
   //Altered by Margulis on June 4, 2001 to incorporate MAGMA
   //Tokens.

   //see saam.util for PrimitiveConversions and Array classes
   long timeStamp = PrimitiveConversions.getLong(
                           Array.getSubArray(packet, 0, 8));
   byte numberOfInputMsgs = packet[8];

   gui.sendText("\nProcess packet: " +
     "\n  size:         " + packet.length +
     "\n  # of Messages: " + numberOfInputMsgs +
     "\n  timeStamp:    " + timeStamp);

   //now we trim the packet by removing the header.
   packet = Array.getSubArray(packet, 9, packet.length);

   //used to track the current position in the array.
   int index = 0;
   short length = 0;


   String elementName = "";
   Class message = null;

   //extract and process each atomic element of the packet
   //separately.  Here we assume the packet is a properly
   //formatted SAAMPacket when it arrives, and that the
   //length is less than the max allowed.

   for (int i = 1; i <= numberOfInputMsgs; i++)
   {

     gui.sendText("\n  Processing Element["+i+"]:");
     System.out.println("\n  Processing Element["+i+"]:");
     byte type = packet[index++];

     gui.sendText("    type:  " + type +
       ", typeName= " + Message.lookUpMessageByType(type));

     System.out.println("    type:  " + type +
       ", typeName= " + Message.lookUpMessageByType(type));
```

```
      byte [] bytes;

/************Newstuff for Converted Type-1 messages    -crcp ***/

      if ((type >= Message.ARPCACHE && type <= Message.RBCCTBL_ENTRY)
          || (type == Message.FLOW_RESPONSE)
          || (type == Message.FLOW_REQUEST)
          )
      { // temp while transitioning -crcp
        gui.sendText("    This is a new Message of type: " + type +
            ", typeName= " + Message.lookUpMessageByType(type));
        short messageLength = PrimitiveConversions
                            .getShort(Array.getSubArray(packet, index, index + 2));

        gui.sendText("%%%%%%%Instance NEW msg event: messageLen = "+ messageLength +
                  " index= " + index);
//    I have removed classname and classNameLen from the msg format for these new types.
        MessageEvent me = new MessageEvent( //-crcp do this like LSA
                    this.toString(),
                    this,
                    ControlExecutive.SAAM_CONTROL_PORT,
                    new Message(Array.getSubArray      // box as generic message
//                          (packet, index - 1, index - 1 + messageLength))
                       (packet, index - 1, index + 2 + messageLength))
                    );

//        index += messageLength - 1;
        index += messageLength + 2; //this means the LSA indexing should be changed!-crcp


        try
        {
          gui.sendText("    Forwarding on channel " +
                            ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(me);
          gui.sendText(" Succeeded  talking OK");
        }
        catch (Exception e)
        {
          gui.sendText(e.toString());
        }

        continue;
      }
/******************/


      switch (type)
      {
        case Message.RESIDENT_AGENT:
        case Message.INTERFACE_SHUTDOWN:
          // Will this go away?  -crcp**********
          //retrieve the number of bytes the class name occupies
          byte nameLength = packet[index++];

          //extract the name of the class file as a byte array
          byte [] elementNameArray = Array.getSubArray(packet, index,
                                          index + nameLength);
          index += nameLength;

          //convert the name back into a String
          elementName = new String(elementNameArray);
          gui.sendText("    Name: " + elementName);
          // End of Will this go away? -crcp********  for now, assume not.


          length = PrimitiveConversions.getShort(Array.getSubArray(packet, index,
                                                index + 2));
          index += 2;
          gui.sendText("    Length:       " + length);
```

```
bytes = Array.getSubArray(packet, index, index + length);
index += length;

if (type == Message.RESIDENT_AGENT)
{
 gui.sendText("    This is a ResidentAgent");
 //Assume this class is of type ResidentAgent
 try
 {
  //Attempt to define the class using the current
  //class loader.
  loader.defClass(elementName, bytes);

 }
 catch (LinkageError le)
 {
  //If the loader already has a definition for the class
  //a LinkageError will be thrown.  If this happens, we
  //need to instantiate a new class loader and use it to
  //define the class.  A nice little trick we learned from
  //page 55 of Jason Hunter's "Java Servlet Programming" book.

  gui.sendText(le.toString());
  gui.sendText("Class was previously loaded...");
  gui.sendText("Replacing old ClassLoader...");
  Loader newLoader = new Loader();
  newLoader.defClass(elementName, bytes);
 }
 try
 {
  //message is of type Class.
  message = Class.forName(elementName, true, loader);
 }
 catch (ClassNotFoundException cnfe)
 {
  gui.sendText(cnfe.toString());
 }
 gui.sendText(message.toString());
 ResidentAgentEvent rae = new ResidentAgentEvent(
    this.toString(),
    this,
    ControlExecutive.SAAM_CONTROL_PORT,
    message);
 try
 {
  gui.sendText("    Forwarding on channel " +
                ControlExecutive.SAAM_CONTROL_PORT);
  controlExec.talk(rae);
 }
 catch (ChannelException tde)
 {
  gui.sendText(tde.toString());
 }


}//end ResidentAgent



//Hasan UYSAL
else if (type == Message.INTERFACE_SHUTDOWN)
{
 gui.sendText("    This is an Interface Shutdown message.");

 InterfaceShutdown failure = new InterfaceShutdown(bytes);
 MessageEvent failMes = new MessageEvent(
    this.toString(),
    this,
    ControlExecutive.SAAM_CONTROL_PORT,
    failure);
    try
    {
      controlExec.talk(failMes);
    }
    catch (Exception ex)
    {
      gui.sendText("Problem with interface shutdown message.");
      continue;
    }
  }
  break;


//THIS CASE PROCESSES THE HEARBEATQUERY MESSAGE
//added by Efraim Kati
case Message.HEARTBEAT_QUERY:

  //retrieve the bytecode of the Object
  bytes = Array.getSubArray(packet, 1, packet.length);
  gui.sendText("    This is a  HeartbeatQuery message");

  //Create the instance of this Message
  HeartbeatQuery hbq = new HeartbeatQuery(bytes);
  gui.sendText(hbq.toString());

  MessageEvent hbqMe = new MessageEvent(
                              this.toString(),
                              this,
                              ControlExecutive.SAAM_CONTROL_PORT,
                              hbq);

  //send this MessageEvent on the Control port.
  try
  {
    gui.sendText("    Forwarding on channel " +
                              ControlExecutive.SAAM_CONTROL_PORT);
    controlExec.talk(hbqMe);
  }
  catch (ChannelException tde)
  {
    gui.sendText(tde.toString());
  }
  break;

//THIS CASE PROCESSES THE HEARBEATRESPONSE MESSAGE
//added by Efraim Kati
case Message.HEARTBEAT_RESPONSE:

  //retrieve the bytecode of the Object
  bytes = Array.getSubArray(packet, 1, packet.length);
  gui.sendText("    This is a  HeartbeatResponse message");

  //Create the instance of this Message
  HeartbeatResponse hbr = new HeartbeatResponse(bytes);
  gui.sendText(hbr.toString());

  MessageEvent hbrMe = new MessageEvent(
                              this.toString(),
                              this,
                              ControlExecutive.SAAM_CONTROL_PORT,
                              hbr);

  //send this MessageEvent on the Control port.
  try
  {
    gui.sendText("    Forwarding on channel " +
                                ControlExecutive.SAAM_CONTROL_PORT);
    controlExec.talk(hbrMe);
  }
```

```
          catch (ChannelException tde)
          {
            gui.sendText(tde.toString());
          }
          break;

        //Henry
//        case Message.FLOW_REQUEST:  //-crcp outcomment
//        case Message.FLOW_RESPONSE: //-crcp outcomment
        case Message.RESOURCE_ALLOCATION:
        case Message.SLS_TABLE_ENTRY:
//Note:  within type, these messages can vary in length!  -crcm
          //retrieve the length of the Object
          length = PrimitiveConversions.getShort(Array.getSubArray(packet, index,
                                                       index + 2));
            index += 2;

//          gui.sendText("    Length:       " + length);
          bytes = Array.getSubArray(packet, index, index + length);
          index += length;


/* As yet unconverted.  See ce.processMessage(byte []) -crcp ******
          if (type == Message.RESOURCE_ALLOCATION)
          {
            gui.sendText("    This is a ResourceAllocation Message");
            controlExec.processMessage(bytes, "ResourceAllocation");
          }//end resource allocation

          else if (type == Message.SLS_TABLE_ENTRY)
          {
            gui.sendText("    This is a SLSTableEntry Message");
            if (bytes.length == SLSTableEntry.REMOVE_SLS_TYPE)
            {
              controlExec.processMessage(bytes, "SLSTableEntry");
            }
            else
            {
              processMessage(bytes, this.toString(), "SLSTableEntry");
            }
          }//end slstableentry
          break;

        case Message.FLOW_TERMINATION:
          length = 4;

//            gui.sendText("  Length:       " + length);
          bytes = Array.getSubArray(packet,index,index + length);
          index += length;
          gui.sendText("    This is a FlowTermination Message");
          controlExec.processMessage(bytes, "FlowTermination");
******************/


          break;

      // [sam]
      // this case statement is added by MARGULIS in JUNE 2001
      case Message.MAGMA_TOKEN:
        //I got a magma message so I need to process accordingly
        //Assume this class is of type Message. -crcp

        gui.sendText("Received Magma Token");
        System.out.println("Received Magma Token");


        length = PrimitiveConversions.getShort(Array.getSubArray(packet, index, index + 2));

        gui.sendText("%%%%%%%Instance LSA msg event: messageLen = " + length +
          " index= " + index);
        // Check this length number...  Should concide with payload

        System.out.println("IBPF:processPacket: case Message.MAGMA_TOKEN:length = "+ packet.length);

        //altered this call to match new baseline code [sam]
        controlExec.processMessage(new Message(packet));

        index += length;

        break; // end of Message.MAGMA_TOKEN case


         //Hasan AKKOC
         case Message.DCM:
         case Message.PARENT_NOTIFICATION:

           if (type == Message.DCM)
           {
             gui.sendText("    This is a DCM  Message");

             try
             {
         //    DCM  dcm = new DCM(packet); // use this if also using appendDCM()-crcn
//              bytes = Array.getSubArray(packet, 1, packet.length); //bug fix -crcn
//              bytes = Array.getSubArray(packet, 0, packet.length); //bug fix -crcn

//              DCM dcm = new DCM(bytes); //bug fix -crcn

//              gui.sendText("DCM message ia created.");
//              gui.sendText(dcm.toString());

//              MessageEvent me = new MessageEvent( //-comment out -crcp
//                                    this.toString(),
//                                    this,
//                                    ControlExecutive.SAAM_CONTROL_PORT,
//                                    dcm);

               System.out.println("%%%%%%%%%%%% instantiate a DCM message event");

               MessageEvent me = new MessageEvent(
                                    this.toString(),
                                    this,
                                    ControlExecutive.SAAM_CONTROL_PORT,

                                    new Message(packet)); // convey generic msg

              //send this MessageEvent on the Control port.
              try
              {
               gui.sendText("    Forwarding on channel " +
                                   ControlExecutive.SAAM_CONTROL_PORT);
               controlExec.talk(me);
//               gui.sendText("DCM is sent to ControlExecutive.");
              }
              catch (ChannelException tde)
              {
               gui.sendText(tde.toString());
              }
             }
             catch (Exception e)
             {
              gui.sendText(e.toString());
             }//try-catch
           }//DCM

           else if (type == Message.PARENT_NOTIFICATION)
           {
             gui.sendText("    This is a ParentNotification Message");
```

```
            //Assume this class is of type Message.
            try
            {
             ParentNotification pn = new ParentNotification(packet); //-crcn
//             bytes = Array.getSubArray(packet, 1, packet.length); //bug fix -crcn
//             ParentNotification pn = new ParentNotification(bytes); //bug fix -crcn

             MessageEvent me = new MessageEvent(
                                this.toString(),
                                this,
                                ControlExecutive.SAAM_CONTROL_PORT,
                                pn);

             //send this MessageEvent on the Control port.
             try
             {
              gui.sendText("    Forwarding on channel " +
                                ControlExecutive.SAAM_CONTROL_PORT);
              controlExec.talk(me);
             }
             catch (ChannelException tde)
             {
              gui.sendText(tde.toString());
             }
            }
            catch (Exception e)
            {
             gui.sendText(e.toString());
            }//try-catch
           }
           break;


         case Message.UCM:
           gui.sendText("    This is a UCM  Message");
           int flowIdOfServer = PrimitiveConversions.getInt(Array.getSubArray(packet,
                                            index, index + 4)) - 1;
           gui.sendText("    Flow id of the sever is " + flowIdOfServer);

           if (!controlExec.isServer())
           {
            //I need UCM and the LSAs together
            //need to pass the value of numberOfInputMsgs to UCM handler
            gui.sendText("    This is a router so forwarding the packet to ACE");

            packet = Array.concat(numberOfInputMsgs, packet);

            i = numberOfInputMsgs;  //cause for loop to break? [Xie]

            //create a ProtocolStackEvent and send it to ACE
            int channelToAce = ProtocolStackEvent.FROM_PACKETFACTORY_TO_ACE;
            ProtocolStackEvent stackEvent = new ProtocolStackEvent(
               this.toString(),
               this,
               channelToAce,
               packet);

            try
            {
             gui.sendText("    Forwarding on channel " + channelToAce);
             controlExec.talk(stackEvent);
            }
            catch (ChannelException tde)
            {
             gui.sendText(tde.toString());
            }
           }
           else
           {
            //this is the server that UCM is destined
```

```
    int numberOfRoutersInUCM = PrimitiveConversions.getInt(Array.getSubArray(
      packet, index + 20, index + 24));
    gui.sendText("    Number of reachable routers is " + numberOfRoutersInUCM);

    int UCMLength = 4 + 16 + 4 + 4 +
                (numberOfRoutersInUCM * IPv6Address.length);
    bytes = Array.getSubArray(packet, index, index + UCMLength);
    bytes = Array.concat(type, bytes);
    index += UCMLength;
    try
    {
      UCM ucm  =  new UCM(bytes);
      gui.sendText("\n    " + ucm.toString());

      MessageEvent me = new MessageEvent(
          this.toString(),
          this,
          ControlExecutive.SAAM_CONTROL_PORT,
          ucm);

      gui.sendText("    Forwarding packet on channel " +
                                ControlExecutive.SAAM_CONTROL_PORT);
      controlExec.talk(me);
    }
    catch (Exception ex)
    {
      gui.sendText(ex.toString());
    }
  }//end else
  break;

//Huseyin UYSAL
case Message.LSA:
  //I got a link state advertisement so I need to process accordingly
  //processLSAMessage();
  gui.sendText("    This is a Link State Advertisement");
  //Assume this class is of type Message. -crcp

  short messageLength = PrimitiveConversions.getShort(Array.getSubArray(packet, index, index + 2));


  gui.sendText("%%%%%%%Instance LSA msg event: messageLen = " + messageLength +
    " index= " + index);

  MessageEvent me = new MessageEvent( //-crcp do this like DCM
                this.toString(),
                this,
                ControlExecutive.SAAM_CONTROL_PORT,
                new Message(Array.getSubArray      // box as generic message
                    (packet, index - 1, index - 1 + messageLength))
                );


  index += messageLength - 1;   //-crcp  move it here


  try
  {
    gui.sendText("    Forwarding on channel " +
                        ControlExecutive.SAAM_CONTROL_PORT);
    controlExec.talk(me);
    gui.sendText(" Succeeded  talking OK");//-crcp  Throwing NPE!!!!!
  }
  catch (Exception e)
  {
    gui.sendText(e.toString());
  }

  break;
```

```
// this case statement is added by altinkaya in March 2000
// as a requirement for the server probing process
case Message.PREVIOUS_NODE_ACT:
  gui.sendText("    This is a Previous Node Activation Message");
  PreviousNodeAct pna = new PreviousNodeAct();
  byte typeOfProbing;
  int probeID;
  IPv6Header v6Header = null;
  short payloadLength;

  typeOfProbing = packet[index++];
  probeID = PrimitiveConversions.getInt(Array.getSubArray(packet, index,
                                                        index = index + 4));

  try
  {
   v6Header = new IPv6Header(Array.getSubArray(packet, index,
                                          index = index + 40));
  }
  catch (UnknownHostException uhe)
  {
    gui.sendText("An exception occurred while trying to read IPv6Header ");
  }//end try-catch

  payloadLength = PrimitiveConversions.getShort(Array.getSubArray(packet, index,
                                                        index = index + 2));
  pna.setPacketFormat(typeOfProbing, probeID, v6Header, payloadLength);

  gui.sendText("    Type of probing " + pna.getTypeOfProbing(typeOfProbing));
  gui.sendText("    Probing identification " + probeID );

  MessageEvent mep = new MessageEvent(
                      this.toString(),
                      this,
                      ControlExecutive.SAAM_CONTROL_PORT,
                      pna);     //me->mep -crc
  try
  {
   gui.sendText("    Forwarding on channel " +
                              ControlExecutive.SAAM_CONTROL_PORT);
   controlExec.talk(mep);    // me->mep -crc
  }
  catch (Exception e)
  {
   gui.sendText(e.toString());
  }
       break;

     // this case statement is added by altinkaya in March 2000
// as a requirement for the server probing process
case Message.NEXT_NODE_ACT:
  gui.sendText("    This is a Next Node Activation Message");
  NextNodeAct nna = new NextNodeAct();
  byte typeOfProbingNNA;
  int probeIDNNA;
  IPv6Header v6HeaderNNA = null;
  short measurementIntervalNNA;
  byte nicID;

  typeOfProbingNNA = packet[index++];
  probeIDNNA = PrimitiveConversions.getInt(Array.getSubArray(packet, index,
                                                       index = index + 4));
  try
  {
    v6HeaderNNA = new IPv6Header(Array.getSubArray(packet, index,
                                           index = index + 40 ));
  }
  catch (UnknownHostException uhe)
  {
```

```
  gui.sendText("An exception occurred while trying to read IPv6Header ");
 }//end try-catch
 measurementIntervalNNA = PrimitiveConversions.getShort(
                Array.getSubArray(packet, index, index = index + 2 ));
 nicID = packet[index++];

 nna.setPacketFormat(typeOfProbingNNA, probeIDNNA, v6HeaderNNA,
                  measurementIntervalNNA, nicID);

 gui.sendText("    Type of probing " + nna.getTypeOfProbing(typeOfProbingNNA));
 gui.sendText("    Probing identification " + probeIDNNA);

 MessageEvent meNNA = new MessageEvent(
                        this.toString(),
                        this,
                        ControlExecutive.SAAM_CONTROL_PORT,
                        nna);
 try
 {
  gui.sendText("    Forwarding on channel " +
                         ControlExecutive.SAAM_CONTROL_PORT);
  controlExec.talk(meNNA);
 }
 catch (Exception e)
 {
  gui.sendText(e.toString());
 }//end try-catch
     break;


// this case statement is added by altinkaya in March 2000
// as a requirement for the server probing process
case Message.PROBE_RESULT:
 gui.sendText("    This is a Probe Result Message ");
 ProbeResult pr = new ProbeResult();
 int probeIDPR;
 byte numberResultsPR;
 byte typeofResultPR;
 short measurementResultPR;

 probeIDPR = PrimitiveConversions.getInt(Array.getSubArray(packet, index,
                                        index = index + 4));
 numberResultsPR = packet[index++];
 typeofResultPR = packet[index++];
 measurementResultPR = PrimitiveConversions.getShort(
                       Array.getSubArray(packet, index, index = index + 2 ));
 pr.setPacketFormat(probeIDPR, numberResultsPR, typeofResultPR,
                 measurementResultPR);
 gui.sendText("Information about the Probe Result " + pr.toString());
 MessageEvent mePR = new MessageEvent(
                       this.toString(),
                       this,
                       ControlExecutive.SAAM_CONTROL_PORT,
                       pr);
 try
 {
  gui.sendText("    Forwarding on channel " +
                       ControlExecutive.SAAM_CONTROL_PORT);
  controlExec.talk(mePR);
 }
 catch (Exception e)
 {
  gui.sendText(e.toString());
 }//end try-catch
break;


// this case statement is added by FATIH in DEC 2000
case Message.TRAFFIC_GENERATORS:
```

```
String param = "";
// this array keeps the IPv6 addresses of the nodes
String [] agentInfoArray = new String [15];
int agentInfoCounter = 0;

gui.sendText("    This is a traffic generator agent ");
//retrieve the number of bytes the class name occupies
byte nameLengthThis = packet[index++];

//extract the name of the class file as a byte array
byte [] elementNameArrayThis = Array.getSubArray(packet, index,
                                            index + nameLengthThis);
index += nameLengthThis;

//convert the name back into a String
elementName = new String(elementNameArrayThis);
gui.sendText("    Name: " + elementName);

//retrieve the length of the Object
length = PrimitiveConversions.getShort(Array.getSubArray(packet, index,
                                                      index + 2));
index += 2;
gui.sendText("    Length:       " + length);
bytes = Array.getSubArray(packet, index, index + length);
index += length;

//Assume this class is of type ResidentAgent
try
{
  //Attempt to define the class using the current
  //class loader.
  loader.defClass(elementName, bytes);
}
catch (LinkageError le)
{
  //If the loader already has a definition for the class
  //a LinkageError will be thrown.  If this happens, we
  //need to instantiate a new class loader and use it to
  //define the class.  A nice little trick we learned from
  //page 55 of Jason Hunter's "Java Servlet Programming" book.

  gui.sendText(le.toString());
  gui.sendText("Class was previously loaded...");
  gui.sendText("Replacing old ClassLoader...");
  Loader newLoader = new Loader();
  newLoader.defClass(elementName, bytes);
}
try
{
  //message is of type Class.
  message = Class.forName(elementName, true, loader);
}
catch (ClassNotFoundException cnfe)
{
  gui.sendText(cnfe.toString());
}
gui.sendText(message.toString());

gui.sendText(" Creating Resident Agent");
ResidentAgentEvent rae = new ResidentAgentEvent(
      this.toString(),
      this,
      ControlExecutive.SAAM_CONTROL_PORT,
      message);
while (index < packet.length)
{
  //gui.sendText(" index = " + index);
  //gui.sendText(" packet length =" + packet.length);
  byte paramLength = packet[index++];
  byte [] paramArray = Array.getSubArray(packet, index,index + paramLength);
```

```
          gui.sendText(" paramLength : " + paramLength);

          index += paramLength;

          //convert the type back into a String *************
          param = new String(paramArray);
          gui.sendText(" param: " + param);
          agentInfoArray[agentInfoCounter] = param;
          agentInfoCounter++;
        }

        /*
        for (int it=0; it < 15 ; it++)
        {
          gui.sendText(agentInfoArray[it] + "/");
        }*/

        // set agent info **************************************
        controlExec.setAgentInfo(agentInfoArray);

        // sending agent to control executive
        try
        {
          gui.sendText(" Forwarding on channel " + ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(rae);
        }
        catch (ChannelException tde)
        {
          gui.sendText(tde.toString());
        }
        break;

       default:
       gui.sendText("!!! Packet type unrecognized: " + type);
      //packet type is unrecognized.  Here we could
      //extract a channel_ID that could be embedded
      //in the packet, and then send the unrecognized
      //element on that channel.
     }//end switch
    }//for
  }//processPacket()
```

8. saam\control\ServerInformation.java

```
//23Oct2001[Margulis] - added firstCycle variable and associated methods to help track information
//        for server's first DCM cycle

                        //[sam]
                          private boolean firstCycle = true;


                        //[sam]
                         /**Added by Margulis
                          * Retrieves variable specifying the firstCycle of the server
                          * @return boolean false = not first cycle; true = this is the first cycle
                          */
                          public synchronized boolean isFirstCycle( )
                          {
                            return this.firstCycle;
                          }//end isFirstCycle
```

```
//[sam]
 /**Added by Margulis
  * Sets variable specifying the firstCycle of the server
  * @return void.
  */
  public synchronized void setFirstCycle(boolean b)
  {
    this.firstCycle = b;
  } //end setFirstCycle
```

9. saam\message\Message.java

```
//14May2001[Margulis] - Added MAGMA_TOKEN static variable

                          //Margulis [sam]
                          public static final byte MAGMA_TOKEN            = 32;
```

10. saam\net\IPv6Address.java

```
//11Apr2001 [Margulis] - Added readObject method
//21Mar2001 [Margulis, Stavritis, Wilkins] - Added "implements Serializeable" to this Class
//                    - Added writeObject method

//[sam]
import java.io.*;

//[sam] Modified class to implement Serializable
public class IPv6Address implements Serializable
{

      //[sam]
  /**Added by Margulis
   * writeObject places the variables in this data structure into a
   * serialized format so that the variables can be sent over the
   * network.  This method is mandated by the Serializable interface.
   * @param out ObjectOutputStream
   */
  private void writeObject(ObjectOutputStream out) throws IOException
  {

     ObjectOutputStream os = new ObjectOutputStream (out);

     os.writeObject(this.address);
     os.flush();

   } // End of writeObject


  //[sam]
  /**Added by Margulis
   * readObject places the variables in the serialized format into a
   * data structure so that the variables can be replaced in the
   * state table.  This method is mandated by the Serializable interface.
   * @param out ObjectInputStream
   */
  private void readObject(ObjectInputStream in) throws IOException
  {
    ObjectInputStream ois = new ObjectInputStream (in);

    try
    {
      this.address = (byte[]) ois.readObject();
    }
    catch (ClassNotFoundException cnfe)
    {
      System.out.println("*********** IPv6Address: Method: readObject: ClassNotFoundException thrown
*******");
    }
    catch (IOException ioe)
    {
      System.out.println("*********** IPv6Address: Method: readObject: IOException thrown *******");
    }

  } //end readObject()
```

11. saam\router\ServerTable.java

```
//23Oct2001[Margulis]-Provide mechanism to determine if a host is already in the list of servers


 //[sam]
/**Added by Margulis
 * Method to check whether server with specified server's router id is in table.
 * @param addr router id of server
 * @return boolean value.
 */
 public boolean hasServerByServerId(IPv6Address addr)
 {
   Enumeration e = serverEntries.elements();
   while (e.hasMoreElements())
   {
     ServerTableEntry look =  (ServerTableEntry) e.nextElement();
     IPv6Address routerID = look.getServerRouterID();
     if (routerID.equals(addr))
     {
      //considered equal when server addresses are equal;
      return true;
     }//end if
   }//end while

   return false;
 }//end hasServerByServerId

 //[sam]
 /**Add by Margulis
  *  Added to return the most current serverPathID
  */
 public int getServerRootPathID(){

   int serverRPID = 1;
   Enumeration e = serverEntries.elements();
   while (e.hasMoreElements())
   {
     ServerTableEntry entry = (ServerTableEntry) e.nextElement();
     System.out.println("************** RootPathID = "+entry.getRootPathId());

     if (entry.getRootPathId() > serverRPID)
     {
       serverRPID = entry.getRootPathId();
     }//end if
   }//end of while

   return serverRPID;

 } // end getServerPathID
```

12. saam\server\BasePIB.java

```
//12Apr2001[Margulis] - Had to serialize many of the methods so that one can send the data structures
//                      over the network.  Also added several new methods.

                    //[sam]
                    /**Modified by Margulis to implement Serializable
                     * PathInformationBase develops a data structure to store to current set of
                     * paths between all routes participating in the Server and Agent based Active
                     * network Management (SAAM) autonomous system.
                     *
                     * Initial developed by Dao Chang Kuo, and John Gibson with the assistance of
                     * Charlie Grassi and Kostas Sambanis for CS4552, Spring 2000.
                     */
                    public class BasePIB extends PathInformationBase implements Serializable
                    {

                     //[sam]
                     /**
                      * The GUI display window for MAGMA and MAGMA Controller.
                      */
                     private MAGMAAdminGui displayMAGMA, magmaController;

                     //[sam]
                     /**Modified by Margulis to implement Serializable
                      * aPIIndex Class. This class makes an object of the index values for a given
                      * element of the array of path id hashtables.  The index order is: Source
                      * Node, Destination Node, and Hop Count  This object cross references a
                      * path to the aPI element which contains it.
                      */
                     private class aPIIndex implements Serializable
                     {

                      //[sam]
                      /**Added by Margulis
                       * writeObject places the variables in this data structure into a
                       * serialized format so that the variables can be sent over the
                       * network.  This method is mandated by the Serializable interface.
                       * @param out ObjectOutputStream
                       */
                      private void writeObject(ObjectOutputStream out) throws IOException
                      {
                       ObjectOutputStream os = new ObjectOutputStream (out);

                       os.writeObject(this.iSource);
                       os.writeObject(this.iDestination);
                       os.writeInt(this.iHopCount);
                       os.flush();

                      } //end writeObject()

                     //[sam]
                     /**Added by Margulis
                      * readObject places the variables in the serialized format into a
                      * data structure so that the variables can be replaced in the
                      * state table.  This method is mandated by the Serializable interface.
                      * @param out ObjectInputStream
                      */
                     private void readObject(ObjectInputStream in) throws IOException
                     {
                      ObjectInputStream instrm = new ObjectInputStream (in);

                      try
                      {
                       this.iSource = (Integer) instrm.readObject();
                       this.iDestination = (Integer) instrm.readObject();
                       this.iHopCount = (int) instrm.readInt();
                      } //end try
```

```
  catch (ClassNotFoundException cnfe)
  {
   System.out.println("aPIIndex class: Method readObject:ClassNotFoundException thrown");
  } //end catch cnfe
  catch (IOException ioe)
  {
   System.out.println("aPIIndex class: Method readObject:IOException thrown");
  } //end catch ioe
} //end readObject()

} // End of aPIIndex class

//[sam]
/**Modified by Margulis to implement Serializable
 *  The path class creates objects which represent the key aspects of a path.
 *  - PathID, an class-wrapped representation of a primitive-type integer (the
 *    class wrapper is necessary for storing the pathID in a hashtable object
 *  - aPIIndex object to cross reference the path to the array of all path IDs
 *  - vNodeSequence which contains all the nodes a path traverses, listed
 *    beginning with the destination and ending with the source.
 *  - vInterfaceSequence object containing the sequence. in reverse order of
 *    all interfaces a path traverses.
 */
private class Path implements Serializable
{

//[sam]
/**Added by Margulis
 * writeObject places the variables in this data structure into a
 * serialized format so that the variables can be sent over the
 * network.  This method is mandated by the Serializable interface.
 * @param out ObjectOutputStream
 */
 private void writeObject(ObjectOutputStream out) throws IOException
 {
   ObjectOutputStream os = new ObjectOutputStream (out);

   os.writeObject  (this.iPathID);
   os.writeBoolean (this.bCreated);
   os.writeObject  (this.objaPIIndex);
   os.writeObject  (this.vNodeSequence);
   os.writeObject  (this.vInterfaceSequence);
   os.writeObject  (this.objPathQoS);
   os.writeInt     (this.iNewFlowID);
   os.writeObject  (this.ahtFlows);
   os.flush();
 } //end writeObject()

//[sam]
/**Added by Margulis
 * readObject places the variables in the serialized format into a
 * data structure so that the variables can be replaced in the
 * state table.  This method is mandated by the Serializable interface.
 * @param out ObjectInputStream
 */
private void readObject(ObjectInputStream in) throws IOException
{
  ObjectInputStream instrm = new ObjectInputStream (in);

  try
  {
   this.iPathID            = (Integer)     instrm.readObject();
   this.bCreated           = (boolean)     instrm.readBoolean();
   this.objaPIIndex        = (aPIIndex)    instrm.readObject();
   this.vNodeSequence      = (Vector)      instrm.readObject();
   this.vInterfaceSequence = (Vector)      instrm.readObject();
   this.objPathQoS         = (PathQoS[])   instrm.readObject();
   this.iNewFlowID         = (int)         instrm.readInt();
   this.ahtFlows           = (Hashtable[]) instrm.readObject();
  } //end try
 catch (ClassNotFoundException cnfe)
 {
  System.out.println("Path class: Method readObject:ClassNotFoundException thrown");
 } //end catch cnfe
 catch (IOException ioe)
 {
  System.out.println("Path class: Method readObject:IOException thrown");
 } //end catch ioe

}//end readObject()

 //[sam]
 /**Added by Margulis
  * Gets the next available flow ID of the path.
  * @return int the next available flow ID.
  */
 public int getFlowID()
 {
  return iNewFlowID;
 }//end getFlowID()

} // End of Path class

//[sam]
/**Modified by Margulis to implement Serializable
 * The PathQoS class represents the key parameters values for a path's
 * Quality of Service for a given service level.
 */
private class PathQoS implements Serializable
{

//[sam]
/**Added by Margulis
 * writeObject places the variables in this data structure into a
 * serialized format so that the variables can be sent over the
 * network.  This method is mandated by the Serializable interface.
 * @param out ObjectOutputStream
 */
 private void writeObject(ObjectOutputStream out) throws IOException
 {
   ObjectOutputStream os = new ObjectOutputStream (out);

   os.writeInt   (this.pathAvBW);
   os.writeShort (this.pathDelay);
   os.writeShort (this.pathLossRate);

   os.flush();

 } //end writeObject()

//[sam]
/**Added by Margulis
 * readObject places the variables in the serialized format into a
 * data structure so that the variables can be replaced in the
 * state table.  This method is mandated by the Serializable interface.
 * @param out ObjectInputStream
 */
private void readObject(ObjectInputStream in) throws IOException
{
 ObjectInputStream instrm = new ObjectInputStream (in);

 try{
    this.pathAvBW = (int) instrm.readInt();
    this.pathDelay = (short) instrm.readShort();
    this.pathLossRate = (short) instrm.readShort();

 }//end try
 catch (IOException ioe){
   System.out.println("PathQos class: Method readObject:IOException thrown");
```

```
    }//end catch

   } //end readObject()

  } // End PathQoS class

//[sam]
/**Modified by Margulis to implement Serializable
  * FlowQoS Class defines objects that represent the key QoS characteristics of
  * a flow request.
  */
  private class FlowQoS implements Serializable
  {

  //[sam]
  /**Added by Margulis
   * writeObject places the variables in this data structure into a
   * serialized format so that the variables can be sent over the
   * network.  This method is mandated by the Serializable interface.
   * @param out ObjectOutputStream
   */
   private void writeObject(ObjectOutputStream out) throws IOException
   {
     ObjectOutputStream os = new ObjectOutputStream (out);

     os.writeShort(this.requestedDelay);
     os.writeShort(this.requestedLossRate);
     os.writeInt  (this.requestedBandwidth);
     os.writeLong (this.timeStamp);
     os.flush();

    } //end writeObject()

  //[sam]
  /**Added by Margulis
   * readObject places the variables in the serialized format into a
   * data structure so that the variables can be replaced in the
   * state table.  This method is mandated by the Serializable interface.
   * @param out ObjectInputStream
   */
  private void readObject(ObjectInputStream in) throws IOException
  {
   ObjectInputStream instrm = new ObjectInputStream (in);

   try
   {
     this.requestedDelay     = (short) instrm.readShort();
     this.requestedLossRate  = (short) instrm.readShort();
     this.requestedBandwidth = (int)   instrm.readInt();
     this.timeStamp          = (long)  instrm.readLong();
   }//end try
   catch (IOException ioe){
     System.out.println("FlowQoS class: Method readObject:IOException thrown");
   }//end catch
  } //end readObject()


  } // End FlowQoS class

//[sam]
 /**Modified by Margulis to implement Serializable
   * The ObsQoS class defines an object responsible for storing observed QoS
   * parameters associated with a given pair of interfaceservice - service
   * level.
   */
  private class ObsQoS implements Serializable
  {
```

```
//[sam]
/**Added by Margulis
 * writeObject places the variables in this data structure into a
 * serialized format so that the variables can be sent over the
 * network.  This method is mandated by the Serializable interface.
 * @param out ObjectOutputStream
 */
 private void writeObject(ObjectOutputStream out) throws IOException
 {
   ObjectOutputStream os = new ObjectOutputStream (out);

   os.writeShort(this.iUtilization);
   os.writeShort(this.iDelay);
   os.writeShort(this.iLossRate);
   os.flush();

  } //end writeObject()


//[sam]
/**Added by Margulis
 * readObject places the variables in the serialized format into a
 * data structure so that the variables can be replaced in the
 * state table.  This method is mandated by the Serializable interface.
 * @param out ObjectInputStream
 */
private void readObject(ObjectInputStream in) throws IOException
{
  ObjectInputStream instrm = new ObjectInputStream (in);

  try
  {
    this.iUtilization = (short) instrm.readShort();
    this.iDelay       = (short) instrm.readShort();
    this.iLossRate    = (short) instrm.readShort();
  }//end try
  catch (IOException ioe)
  {
    System.out.println("ObsQoS class: Method readObject:IOException thrown");
  }//end catch

 } //end readObject()

} // End of ObsQoS class

//[sam]
/**Modified by Margulis to implement Serializable
 * The InterfaceInfo class defines the objects that contain all key
 * information associated with a single SAAM interface.
 */
private class InterfaceInfo implements Serializable
{

//[sam]
/**Added by Margulis
 * writeObject places the variables in this data structure into a
 * serialized format so that the variables can be sent over the
 * network.  This method is mandated by the Serializable interface.
 * @param out ObjectOutputStream
 */
 private void writeObject(ObjectOutputStream out) throws IOException
 {
   ObjectOutputStream os = new ObjectOutputStream (out);

   os.writeObject (this.iNodeID);
   os.writeInt    (this.iTotalBandwidth);
   os.writeObject (this.iServiceLevelAvailableBandwidth);
   os.writeByte   (this.bSubnetMask);
   os.writeObject (this.htPathIDs);
   os.writeObject (this.aobjObsQoS);
```

```
    os.flush();

   } //end writeObject()

  //[sam]
  /**Added by Margulis
   * readObject places the variables in the serialized format into a
   * data structure so that the variables can be replaced in the
   * state table.  This method is mandated by the Serializable interface.
   * @param out ObjectInputStream
   */
  private void readObject(ObjectInputStream in) throws IOException
  {
   ObjectInputStream instrm = new ObjectInputStream (in);

   try{
     this.iNodeID = (Integer) instrm.readObject();
     this.iTotalBandwidth = (int) instrm.readInt();
     this.iServiceLevelAvailableBandwidth = (int[]) instrm.readObject();
     this.bSubnetMask = (byte) instrm.readByte();
     this.htPathIDs = (Hashtable) instrm.readObject();
     this.aobjObsQoS = (ObsQoS[]) instrm.readObject();
   }//end try
   catch (ClassNotFoundException cnfe)
   {
     System.out.println("InterfaceInformationObject class: Method readObject:ClassNotFoundException
thrown");
   }//end catch
   catch (IOException ioe)
   {
     System.out.println("InterfaceInformationObject class: Method readObject:IOException thrown");
   }//end catch

  } //end readObject()

  } // End of InterfaceInfo class

 //[sam]
 /**Modified by Margulis
  * Constructs the BasePIB object with a given reference to a SAAM server.
  * @param theServer the Server.
  */
  public BasePIB (Server theServer)
  {
   // Create Gui for PIB display during generation.
   gui = new SAAMRouterGui("PathInformationBase");

   //[sam]
   // Adding server handle to Magma controller
   displayMAGMA    = new MAGMAAdminGui("DigestPIB", theServer);
   magmaController = new MAGMAAdminGui ("Admin Control", theServer);

   theServer.getControlExec().addComponentGui(gui); // -crcy

   //[sam]
   theServer.getControlExec().addMagmaGui(displayMAGMA);
   theServer.getControlExec().addMagmaGui(magmaController);

   myServer = theServer;

   vIFacesTraversed = new Vector();

   df = (DecimalFormat) NumberFormat.getCurrencyInstance();
   df.setMaximumFractionDigits(2);
   df.applyPattern("#0.00%");

   // Instantiate the array of hashtables to hold path IDs as paths are created
   aPI = new Hashtable [MAX_NODE_NUMBER] [MAX_NODE_NUMBER] [MAX_HOP_COUNT];
   for (int i = 0; i < MAX_NODE_NUMBER; i++)
   {
      for (int j = 0; j < MAX_NODE_NUMBER; j++)
      {
        for (int k = 0; k < MAX_HOP_COUNT; k++)
        {
          // Instantiate hashtables for each element of the Path Info array
          aPI[i][j][k] = new Hashtable();
        }
      }
    } // End of array creation

    // Instantiate each PIB member hashtable
    htRouterIDtoNodeID = new Hashtable();
    htNodeIDtoRouterID = new Hashtable();
    htRouterInterfaceMap = new Hashtable();
    htInterfaces = new Hashtable();
    htPaths = new Hashtable();

    // used for Differentiated Service
    //[GX]: carried over from Henry's code?
    // only six user-ids are registered; read from a file?
    htUserSLSs = new Hashtable();

    // Put six entries to the table
    htUserSLSs.put(new Integer(1), new SLS(SLS.SILVER_CLASS));
    htUserSLSs.put(new Integer(2), new SLS(SLS.BRONZE_CLASS));
    htUserSLSs.put(new Integer(3), new SLS(SLS.GOLD_CLASS));
    htUserSLSs.put(new Integer(4), new SLS(SLS.SILVER_CLASS));
    htUserSLSs.put(new Integer(5), new SLS(SLS.BRONZE_CLASS));
    htUserSLSs.put(new Integer(6), new SLS(SLS.GOLD_CLASS));

    // Instantiate the routing algorithm object.
    routingAlgorithm = new RoutingAlgorithm();

    //Display pib generic information
    gui.sendText(this.toString());

    //Set default state for inter-service borrowing
    setInterserviceBorrowing(INTERSERVICE_BORROWING_DEFAULT);
    setBorrowingThreshold(DEFAULT_BORROWING_THRESHOLD);

  } // End BasePIB constructor()

/**Added by Margulis [sam]
 * Function updateDigestPIB:
 *  Updates the information in the Digest PIB. This information includes a digestAPI
 *  which contains the PathID of the widest path from a source router to a destina-
 *  tion router (widest path is defined as the largest available bandwidth for the
 *  path requested), the htInterface Hashtable with contains detailed information
 *  about each node in the SAAM Region, and DigestHtPaths Hashtable which contains
 *  the PathIDs and Paths from only those source and destation routers which begin
 *  and end with an edge router.
 *
 * @param:  DigestPIB digestPIB
 * @return: DigestPIB digestPIB
 */
  public synchronized DigestPIB updateDigestPIB()

    {
      Integer[][]  digestAPI;  // A two-dimensional array of integers containing Path ID for
                               // the maximum bandwith avaiable between a source and destination router
                               // pair.  The array is set up as follows:
                               // digestAPI[sourceNode][destinationNode].

      DigestPIB digestPIB = new DigestPIB();

      digestAPI = new Integer[MAX_NODE_NUMBER][MAX_NODE_NUMBER];

    for (int i = 0; i < aPI.length; i++)     // From source router (i)
      {
        for (int j = 0; j < aPI[i].length; j++)     // To destination router (j)
        {


          if(i != j){
            digestAPI[i][j] = getWidestPath(i,j,aPI);  // local method to find the path from the
                                                       // source (i) to the destination (j) which
                                                       // contains the largest available bandwidth.
                                                       // The method returns the PathID as an Integer
          } // End if
          else {
            digestAPI[i][j] = new Integer(0);
          } // End else

          // object to be stored in digestAPI Two-D array.
        } // end of j loop

      } // end of i loop

      digestPIB.setDigestAPI(digestAPI);

      digestPIB.setDigestHtInterfaces(htInterfaces);
      digestPIB.setDigestHtPaths(getReducedHtPaths(htPaths, digestAPI));



      //displayDigestPIB(digestPIB);

      return digestPIB;

    }//end updateDigestPIB


 /**Added by Margulis [sam]
  * Function getWidestPath:
  *  This function finds the path from the source (i) to the destination (j) which
  *  contains the largest available bandwidth. The method returns the PathID as an
  *  Integer object to be stored in digestAPI Two-Dimensional array.

  *
  * @param:  int source
  * @param:  int destination
  * @param:  Hashtable aPI[][][]
  * @return: Integer bestPath
  */
 public Integer getWidestPath(int source, int dest, Hashtable aPI[][][])
 {
      Hashtable table = new Hashtable();   // Stores the Hashtable for a
                                           // given source/dest/hop count from
                                           // the aPI Hashtable

      Path currentPath = null;             // Stores a Path Object from the htPaths
                                           // Hashtable.

      Integer bestPath = new Integer(-1);  // Keeps track of the PathID of the path
                                           // with the largest available bandwidth

      int maxBandwidthAvailable = 0;       // A temporary storage bin for the largest
                                           // bandwidth.  This number is compared with
                                           // each path's bandwidth to determine which
                                           // path has the largest available bandwidth
        for (int k = 1; k < aPI[source][dest].length; k++)  // search each path for all hop counts to
                                                   // determine which one has the biggest pipe,
                                                   // regardless of hop count
          {

            table = aPI[source][dest][k];          // create a copy of the aPI hashtable
            Enumeration enum = table.elements();   // enumerate the hashtable aPI copy

             // For each of the PathIDs, get the reference path to determine the max bandwidth available
            while (enum.hasMoreElements())  // search through each path
            {
             Integer currentPathID = (Integer) enum.nextElement();

             currentPath = (Path) htPaths.get(currentPathID);  // get the Path oject for the currentPathID
             if (currentPath != null){
                 if (currentPath.objPathQoS[INT_SERV].getAvailableBandwidth( ) > maxBandwidthAvailable)
                 {
                   maxBandwidthAvailable = currentPath.objPathQoS[INT_SERV].getAvailableBandwidth( );
                   bestPath = currentPathID;}
             }
             else {
             }// End else
            } // end while loop


          }// end of k loop
          return bestPath;   // looking for maximum bandwidth available
                            // once maximum bandwidth available is found,
                            // store only that PathID.

  }  //end getWidestPath()

  /**Added by Margulis [sam]
  * Function getReducedHtPaths:
  *  This function finds paths from the source (i) to the destination (j) and
  *  determines if they begin and end at an edge router.  If they don't begin
  *  and end at an edge router, the path is removed from the hashtable. The
  *  function returns a scaled down version of the BasePIB's htPaths Hashtable.
  *
  * @param:  Hashtable htPaths
  * @return: Hashtable reducedHtPaths
  */
 public Hashtable getReducedHtPaths( Hashtable htPaths, Integer[][] digestAPI)
 {
  Hashtable table = new Hashtable();     // creates hashtable to store
                                        // a copy of the htPaths hashtable

  Enumeration enum = htPaths.keys();     // enumerates the hashtable to observe its contents




  // For each of the PathIDs, get the iNewFlowID number to determine if the path
  // starts and ends at an edge router.
  while (enum.hasMoreElements())  // search through each path
     {
       Integer currentPathID = (Integer) enum.nextElement();


       Path currentPath = (Path) htPaths.get(currentPathID);  // gets the Path object for a
                                                             // given PathID number



       if (currentPath.getFlowID() != 0)                     // looks at the iNewFlowID number
           {
            table.put(currentPathID, currentPath);
           } // end if
       else if (currentPath != null && inWidestPath(digestAPI, currentPathID)){
            table.put(currentPathID, currentPath);
       } // end else if

                                                             // and ends at an edge router
     } // end while loop


   return table;   // removes those paths that do not begin or end with an*/
                   // edge router
  }//end getReducedHtPath()

 /**Added by Margulis [sam]
  * Function inWidestPath:
  *  This function determines whether or not a path found in htPath from Base PIB
  *  is part of digestAPI.  If currentPathID is found in digestAPI, then this
  *  function returns true. If not, it returns false.
  *
  * @param:  Integer[][]  digestAPI
  * @param:  Integer      currentPathID
  * @return: boolean true/false
  */
public boolean inWidestPath( Integer[][] tempDigestAPI, Integer currentPathID)
{
  int i=0;

  while (i < MAX_NODE_NUMBER)  // search through each path
  {
    int j=0;
    while (j < MAX_NODE_NUMBER)  // search through each path
    {
      if (tempDigestAPI[i][j].intValue() != -1 && tempDigestAPI[i][j].intValue() == currentPathID.intValue())
      {
          return true;
      } // end if
      j++;
    }  // end While for j
    i++;
  }// end While for i

  return false;

 }//end inWidestPath()

 /**Added by Margulis [sam]
  * Function getDisplayMagma:
  *  Returns the displayMagma Gui.
  *
  * @param: none
  * @return: MAGMAAdminGui displayMagma
  */
  public MAGMAAdminGui getDisplayMagma()
  {
    return displayMAGMA;
  }//end getDisplayMagma()



 /**Added by Margulis [sam]
  * Function displayDigestPIB:
  *  Display the content of the current PIB.
  *
  * @param: DigestPIB digestPIB
  * @return: void
  */
  public void displayDigestPIB(DigestPIB digestPIB)
  {

    String sDigestPIBstring = "";   // sdigestPIBstring built to allow display to either DOS or GUI window

    if (false) // Set to true to display path information array
    {
     sDigestPIBstring = "The DigestPIB contains following paths:\n";

      for (int i = 0; i < MAX_NODE_NUMBER; i++)
      {
        for (int j = 0; j < MAX_NODE_NUMBER; j++)
        {
           // Skip the case of the same source and destination
           if (i == j)
          {
           continue;
          } //end if (i==j)
          if (!(digestPIB.digestAPI[i][j].intValue() == -1))
          {

             Integer currentPathID = digestPIB.digestAPI[i][j];
             Path currentPath = (Path) digestPIB.digestHtPaths.get(currentPathID);

             sDigestPIBstring += "From Source: " + i + " to destination: " +
                                  j + " ==> Path: " + digestPIB.digestAPI[i][j].intValue() + "\n";
             // Extract that path's information
             PathQoS  x[] = currentPath.getPathQoSArray();

             PathQoS test = x[INT_SERV];
             sDigestPIBstring += "Available Bandwidth: " + test.getAvailableBandwidth() + "\n";
             sDigestPIBstring += "Delay: " + test.getPacketDelay() + "\n";
             sDigestPIBstring += "LossRate: " + test.getPacketLossRate() + "\n\n";
             // End of QoS display




             // Set to true to display the sequence of the nodes the
             // path traverses
             if (true)
             {
              sDigestPIBstring += currentPath.toString();

             }// End if for Node Sequence display


              // Set to true to display the Interface Sequence traversed by
              // a path:
             if (false)
             {
              sDigestPIBstring += currentPath.toString();

             }// End Interface Sequence Display block


          }// End if statement for processing non-empty pathID hashtables

        } // End destination based loop

      } // End source based loop

    }// End if for path information array display


    if (true) // Set to true to display all current paths in the PIB:
    {
     sDigestPIBstring += toStringPaths(1);
    }

    // Set to true to display all current interfaces in PIB and their
    // QoS settings:
    if (true)
    {
     sDigestPIBstring += toStringInterfaces();
    }

    // Dump string to display
     displayMAGMA.sendText("*** DIGEST PIB ***\n");
     displayMAGMA.sendText(sDigestPIBstring);

     displayDigestHtPaths(digestPIB.digestHtPaths);

   } //end displayDigestPIB()
```

```
/**Added by Margulis [sam]
  * Function: displayDigestHtPaths
  *
  * @param: Hashtable digestHtPaths
  * @return: void
  */
  public void displayDigestHtPaths(Hashtable digestHtPaths)
  {
    String digestPIBshow = "Total number of paths in the PIB is: " + digestHtPaths.size();

    if (digestHtPaths.size() > 0) {
      Enumeration enum = digestHtPaths.elements();
      digestPIBshow = digestPIBshow.concat("\n\tPath-ID\t\tNode-sequence\n"); // align entries with tabs
      while (enum.hasMoreElements())
      {
        Path path = (Path) enum.nextElement();
        digestPIBshow = digestPIBshow.concat("\n\t" + path.getPathID().intValue() + "\t\t");


        Enumeration enumNodeSeq = path.getNodeSequence().elements();
        while (enumNodeSeq.hasMoreElements())
        {
          Integer nodeID = (Integer) enumNodeSeq.nextElement();
          digestPIBshow = digestPIBshow.concat(nodeID.toString());

          if (enumNodeSeq.hasMoreElements())      // Append an arrow if this is not
          {                                        // the last node in the sequence
            digestPIBshow = digestPIBshow.concat(" <- ");
          }
          else
          {
            digestPIBshow = digestPIBshow.concat("\n");
            break;
          }
        } // End loop to display node sequence

        PathQoS y[] = path.getPathQoSArray();
        for (int i = 0 ; i < y.length ; i++)
          {
            digestPIBshow = digestPIBshow.concat("QoS parameters for Path Service Level " + i + " is: \n");
            digestPIBshow = digestPIBshow.concat("Available Bandwidth: "  + y[i].getAvailableBandwidth() +
"\n");
            digestPIBshow = digestPIBshow.concat("Delay: "      + y[i].getPacketDelay() + "\n");
            digestPIBshow = digestPIBshow.concat("Loss Rate: "  + y[i].getPacketLossRate() + "\n\n");
          }


      } // End loop to step through paths
    } // End if

      displayMAGMA.sendText(digestPIBshow);

  }//end displayDigestHtPaths()

  //[sam]
  /**Added by Margulis
   * writeObject places the variables in this data structure into a
   * serialized format so that the variables can be sent over the
   * network.  This method is mandated by the Serializable interface.
   * @param out ObjectOutputStream
   */
   private void writeObject(ObjectOutputStream out) throws IOException
   {
      ObjectOutputStream os = new ObjectOutputStream (out);

      os.writeObject(this.aPI);
      os.writeObject(this.htRouterIDtoNodeID);
      os.writeObject(this.htNodeIDtoRouterID);
      os.writeObject(this.htRouterInterfaceMap);
      os.writeObject(this.htInterfaces);
    os.writeObject(this.htPaths);
    os.writeObject(this.htUserSLSs);
    os.flush();

  } //end writeObject()

  //[sam]
  /**Added by Margulis
   * readObject places the variables in the serialized format into a
   * data structure so that the variables can be replaced in the
   * state table.  This method is mandated by the Serializable interface.
   * @param out ObjectInputStream
   */
  private void readObject(ObjectInputStream in) throws IOException
  {
   ObjectInputStream instrm = new ObjectInputStream (in);

   try
   {
    this.aPI = (Hashtable[][][]) instrm.readObject();
    this.htRouterIDtoNodeID = (Hashtable) instrm.readObject();
    this.htNodeIDtoRouterID =  (Hashtable) instrm.readObject();
    this.htRouterInterfaceMap = (Hashtable) instrm.readObject();
    this.htInterfaces = (Hashtable) instrm.readObject();
    this.htPaths = (Hashtable) instrm.readObject();
    this.htUserSLSs = (Hashtable) instrm.readObject();
   }
   catch (ClassNotFoundException cnfe)
   {
    System.out.println("InterfaceInformationObject class: Method readObject:ClassNotFoundException
thrown");
   }
   catch (IOException ioe)
   {
    System.out.println("InterfaceInformationObject class: Method readObject:IOException thrown");
   }

  } //end readObject()

} // End of PathInformationBase class
```

13. saam\server\Magma.java

```
//24May2001[Margulis] - Created to provide for Liquid Software (MAGMA) functionality
package saam.server;

import saam.control.*;
import saam.agent.*;

import saam.server.*;
import saam.event.*;
import saam.message.*;
import saam.util.*;
import saam.util.BannerFrame;
import saam.net.IPv6Address;


import java.lang.Byte;
import java.util.*;
import java.util.zip.*;
import java.net.*;
import java.io.*;

public class Magma
{
  private BannerFrame bf = new BannerFrame("");
  private SAAMRouterGui gui;
  private ControlExecutive controlExec;
  private boolean initHeartBeatRan = false;  //part of 5-09-00 bug fix -crc
```

```
 private Server myServer;
 private FIFOQueue inputQueue = new FIFOQueue(1000);

 private Object theLock = new Object();
 private boolean activeServer;
 private boolean magmaAgentAsleep;
 private Message message;

 private ByteArrayOutputStream baos;

 public Magma (  )
 {
 }//end Magma() Constructor


 public void processMagmaMessage(Server myServer, ControlExecutive controlExec, byte[] magmaPayload )
// awaiting fix for sending kill manner message ~sam-b
 {
  System.out.println("MagmaAgent.processMagmaMessage: THE BEGINNING");

  MAGMATokens magmaToken = new MAGMATokens(magmaPayload);

  System.out.println("Magma.processMagmaMessage: The payload length is = "+magmaPayload.length);


  if (MAGMATokens.CHANGEMAGMA == (magmaToken.getMagmaTokenType()))
  {
   System.out.println("Magma.processMagmaMessage: Change Server ");
   try
   {

     System.out.println("MagmaAgent.processMagmaMessage... before standing up new Server");
     myServer.standUpAutoConfigThread();  //Added to stand up the server object
                                         //This addresses a change in the way that the magma server is to be handled
     System.out.println("MagmaAgent.processMagmaMessage... after standing up new Server");

   }
   catch (Exception e)
   {
    System.out.println(e);
    e.printStackTrace();
   }
  } // end if
  else {
   //This is where you need to insert all the other conditions to handle sending the digestPIB
   //or the other updates to the static variable options
   System.out.println("Magma.processMagmaMessage: SENDDIGESTPIB");
  }// end else

 }//end processMagmaMessage()


 //[sam]
 /**Added by Margulis
  * This is designed to provide the SENDUPDATEDIGESTPIB
  *  with a ByteArrayOutputStream of the DigestPIB.
  *  @param  none
  *  @return ByteArrayOutputStream
  */
 public ByteArrayOutputStream getBaos()
 {
  return this.baos;
 }//end getBaos()



/**
 * The section that follows was removed from Server and added here to add a more
 * fluid agent (Margulis June 2001)
```

```
  */
 //remove filename, fileNo and uncompressedFileSize
public synchronized void squeezeObject(Object o, Deflater df, String fileName)
{

  DeflaterOutputStream dosC;
  ObjectOutputStream oosC;

  //remove this-- no longer in use
   byte buffer[];           // This was added just to prove network send/receive capabilities [Margulis June 2001]
   DatagramSocket rsocket;  // Added to test if object could be sent over a network and received, inflated
                            // and returned to its orginal object form.  added by Margulis in May 2001
   int  arrayLength;



  try
  {
    baos = new ByteArrayOutputStream();
    dosC = new DeflaterOutputStream(baos,df,1024);
    oosC = new ObjectOutputStream(dosC);

    oosC.writeObject(o);
    oosC.flush();
    oosC.close();

    arrayLength = baos.toByteArray().length;
  }//end try
  catch (FileNotFoundException fnfe)
  {
    System.out.println("That file is not found");
  }
  catch (IOException ioe)
  {
    System.out.println ("Bills I/O Problem: "+ioe);
    ioe.printStackTrace();
  }
  catch (StackOverflowError soe)
  {
    System.out.println ("You have just experienced a Stack Overflow Error: " + soe.getMessage());
  }

}//end squeezeObject()



public synchronized Object expandObject(Inflater infl, String fileName, int fileNo, DatagramPacket rpacket )
{

  InflaterInputStream iis;///
  ObjectInputStream ois;//
  Object o = new Object();

    DatagramSocket socket;  // Added to test if object could be sent over a network and received, inflated
                            // and returned to its orginal object form.  added by Margulis in May 2001

      System.out.println("Server.Magma.expandObject()");

  try
  {
    ByteArrayInputStream bais = new ByteArrayInputStream(rpacket.getData());
    iis = new InflaterInputStream(bais ,infl,1024 );
    ois = new ObjectInputStream(iis);
    o = ois.readObject(); //****** removed on May 22 // put this back into service 22 May 2001
    System.out.println("Server.Magma.expandObject: Object Read and Received");

    iis.close();
    ois.close();
  }
  catch (FileNotFoundException fnfe)
 {
  System.out.println("That file is not found");
 }
 catch (ClassNotFoundException cnfe)  // catch reinstated to service on 22May2001 //removed 22 May 2001
 {
   System.out.println("That Class is not found");
 }
  catch (IOException ioe)
 {
  System.out.println ("****** Scott's I/O Problem: ====>"+ioe);
  ioe.printStackTrace();
 }
    catch (StackOverflowError soe)
 {
  System.out.println ("****You have just experienced a Stack Overflow Error***" + soe.getMessage());
 }

 return o;


}//end expandObject()

}//end Magma Class
```

14. saam\server\Server.java

```
//23Oct2001[Margulis] - Several Methods added from March to October for addition of MAGMA
//                  - functionality.  This includes the addition of an innner Class.
//23Mar2001[Margulis,Stavritis, & Wilkins]- UpdateDigestPIB method for creating a Digest PIB

//[sam]
import saam.net.IPv6Address;
import saam.message.MAGMATokens;

//[sam]
import saam.util.MAGMAAdminGui;

//[sam]
import java.util.zip.*;

                    //[sam]
                    /**Modified by Margulis to implement Serializable
                     * The <em>Server</em> is an object within the SAAM architecture that
                     * maintains a picture of the network for use in assigning flows to paths.
                     */
                    public class Server implements  Runnable, Serializable
                    {
                     private BasePIB newPIB;

                     //[sam]
                     private DigestPIB digestPIB;
                     Object compressLock = new Object();
                     FileOutputStream OutputResultsToFile  ;

                     /** Enables the Server to receive and send particular types of messages. */
                     private ControlExecutive controlExec;


                     //[sam]
                     //  Added this Class in response to a change in strategy of instantiating
                     //  a magma agent for Margulis thesis on November 1, 2001
                     private Magma magma = new Magma();


                     //[sam]
                     //These variables are used to provide MAGMA with statistical information
                     //used to determine the amount of overhead created by DigestPIB compression
                     //and for proof of concept that the DigestPIB could be compressed, passed over the
```

```
//the network and expanded into its original form.
int fileBPIBNo = 0;

int fileNo = 0;
int fileBSNo = 0;
int fileBCNo = 0;
public int arrayLength = 3500;  // temp add by Margulis to test UDP transmit of DigestPIB
private DatagramPacket tempPacket;
byte tbuffer[] = new byte[arrayLength];
DatagramPacket testPacket = new DatagramPacket(tbuffer, tbuffer.length);


/**[sam]
 * Constructs a server that will use a specified type of <em>Path Information
 * Base</em>.  The PIB may be in the form of a database structure (which
 * requires an existing ODBC configured local database) or a class
 * object structure. The control executive is the interface to the IPv6
 * protocol stack, in order for messages to flow to and from the network.
 * The final step taken is the deletion of all existing data, which is
 * important only in a database structure since a class object structure is
 * volatile.
 * @param type The type of structure that the PIB is to assume.
 * @param controlExec The control executive that will exchange messages
 * with this server.
 */
public Server(String type, ControlExecutive controlExec)
{
  this.controlExec = controlExec;
  gui = new SAAMRouterGui("Server");
  controlExec.addComponentGui(gui); // -crcy
  newPIB = new BasePIB(this);//-crcy  must now do this after instancing c.e.

  serverRouterID = controlExec.getRouterId();

  //[sam]
  // Added by Bill Wilkins
  try{
  OutputResultsToFile= new FileOutputStream ("OutputResultsToFile");
  }
  catch(FileNotFoundException fnfe){
    System.out.println("No output results Exception");
  }
  //OutputResultsToFile.openToWrite("OutputResultsToFile");

  logfile = new FileIO();
  logfile.openToWrite("server\\log.dat");

}

/**[sam]
 * Receives link state advertisement messages from router and processes the
 * service level pipe status information that they contain. It begins by
 * checking to see if a router with the interface address described by this
 * LSA is known to the PIB. If such a router is known to exist, it then
 * checks to see if the service level pipe described by this LSA is known to
 * the PIB. If the service level pipe is known, then update its status.
 * Otherwise, add the SLP with the specified QoS characteristics. Finally,
 * update the effective QoS for the paths that pass over this service level
 * pipe by calling the determineEffectiveQoSForPaths().
 * @param router A representation of a router as defined by an LSA.
 */

public void processLSA(LinkStateAdvertisement LSA)
{

  //[sam]  All but last line of code in this method was added by Margulis
  //different kinds of compression algorithm's that can be used with Java libraries
  Deflater df0 = new Deflater(Deflater.NO_COMPRESSION);
  Deflater df1 = new Deflater(Deflater.BEST_SPEED);
  Deflater df2 = new Deflater(Deflater.BEST_COMPRESSION);
```

```
    Deflater dfBPIB = new Deflater(Deflater.NO_COMPRESSION);

    double uncompressedFileSizeBASEPIB = 1;

    DigestPIB inflatedDigestPIB = new DigestPIB();
    digestPIB = new DigestPIB();
    gui.sendText("Debug: calling server.processLSA()");

    newPIB.processLSA(LSA);
    System.out.println("&&&LSA Call&&& \n");

    //  This is the beginning of MAGMA coded added [sam]
    digestPIB = newPIB.updateDigestPIB();

    newPIB.getDisplayMagma().sendText("*** Expanded digestPIB ***");
    newPIB.displayDigestPIB(digestPIB);


    magma.squeezeObject(digestPIB, df1, "digestPIB_Best_Speed_LSA");


//  newPIB.processLSA(LSA);  // Xie-nov01
  }

  /**[sam]
   * Receives and processes flow requests from applications. It begins
   * by finding a source and a destination router. These routers may be where
   * the applications are residing themselves, which is our standard situation.
   * The application could, however, reside on some host that is not registered
   * with the PIB as a router. In this case, the appropriate source or
   * destination router would be a router connected to the same link. <p>
   * The PIB is checked to ensure that there is the effective QoS available on
   * some path to satisfy the request. If a satisfactory path is found, a new
   * unique flow id is assigned and this new flow is associated with that path.
   * Each router in the path is retrieved and a new flow routing table entry is
   * sent to each. If no path can provide the requested level of QoS, then the
   * flow is assigned to zero, which will be interpreted by IPv6 as best effort
   * traffic. Finally, a flow response is sent back to the application to
   * inform it of its assigned flow id. If the flow id that is return is zero,
   * it will be the application's responsibility to either lower it QoS request
   * or to send its traffic as best effort.
   * @param flow_request The message requesting the establishment of a flow.
   */
  public void processFlowRequest(FlowRequest flowRequest)
  {


    //[sam]  All but last line of code in this method was added by Margulis
    //different kinds of compression algorithm's that can be used with Java libraries
    Deflater dfBPIB  = new Deflater(Deflater.NO_COMPRESSION);
    Deflater df0 = new Deflater(Deflater.NO_COMPRESSION);
    Deflater df1 = new Deflater(Deflater.BEST_SPEED);
    Deflater df2 = new Deflater(Deflater.BEST_COMPRESSION);

    double uncompressedFileSizeBASEPIB = 1;

    DigestPIB inflatedDigestPIB = new DigestPIB();
    newPIB.processFlowRequest(flowRequest);

    System.out.println("&&&Flow Call&&& \n");

    digestPIB = newPIB.updateDigestPIB(); //updateDigestPIB added by Scott Margulis,
                                          //George Stavritis, and Bill Wilkins in March 2001
    newPIB.getDisplayMagma().sendText("*** Expanded digestPIB ***");
    newPIB.displayDigestPIB(digestPIB);
    System.out.println("Server.processFlowRequest: Before expandObject");
    magma.squeezeObject(digestPIB, df1, "digestPIB_Best_Speed_Flow");


    newPIB.processFlowRequest(flowRequest);
  }//End of processFlowRequest

 /**Added by Margulis [sam]
  * This method was developed to provide a handle for the MAGMAControlGui to send
  * a MAGMATokens message from the exiprocessMagmaControllersting host to another host
  * @param  MagmaControlGui  This is the Object which will contain the functionality
  *                      for the MAGMA server
  * @return void
  */
  public void processMagmaController (MagmaControlGui magmaControlGui)  // complete Send Button action
in MagmaControlGui ~sam-a
  {

    IPv6Address newMagmaIPv6Address = magmaControlGui.getIPv6Address();
    MAGMATokens magmaMessage;
    MAGMAAdminGui displayMAGMA = new MAGMAAdminGui("DigestPIB", this);
    if (magmaControlGui.isCHANGEMAGMAselected())
    {
      System.out.println("Inside processMagma Controller: ChangeMagma to:
"+magmaControlGui.getIPv6Address().toString());
      magmaMessage = new MAGMATokens(newMagmaIPv6Address);

//  We Needto modify this next command to send the magmaMessage

      try
      {
        System.out.println("Inside server.processMagmaController: payloadLength = "+
magmaMessage.getBytes().length);
        System.out.println("Inside server.processMagmaController: 1st byte = "+
PrimitiveConversions.getInt(Array.getSubArray(magmaMessage.getBytes(),0,1)));
        System.out.println("Inside server.processMagmaController: IPv6 = "+
newMagmaIPv6Address.toString());

      controlExec.send(this,
                    magmaMessage,
                    serverRootPathID,//1,
                    PSUEDORANDOMSOURCEPORT,
                    newMagmaIPv6Address,
                    (short) 8000);

      // kill the server banner assumes that the new server and server banner stands up
      killServerBanner();


      }//end try
      catch (Exception e)
      {
        System.out.println("Inside processMagmaController... "+e);
        e.printStackTrace();
      }//end catch

      killAutoConfigThread();

    }//end if
    else{
      System.out.println("Inside processMagma Controller: ChangeMagma is FALSE");
    } // end of if-else "isCHANGEMAGMAselected"


    if (magmaControlGui.isSENDUPDATEDDIGESTPIBselected()){
      System.out.println("Inside processMagma Controller: SENDUPDATEDDIGESTPIB is TRUE");
      magmaMessage = new
MAGMATokens(newMagmaIPv6Address,"SENDUPDATEDDIGESTPIB",magma.getBaos());

//  We Need to modify this next command to send the magmaMessage

      try
      {
        System.out.println("Inside server.processMagmaController: payloadLength = "+
magmaMessage.getBytes().length);
```

```
        System.out.println("Inside server.processMagmaController: 1st byte = "+
PrimitiveConversions.getInt(Array.getSubArray(magmaMessage.getBytes(),0,1)));
        System.out.println("Inside server.processMagmaController: IPv6 = "+
magmaControlGui.getIPv6Address().toString());

        controlExec.send(this,
                         magmaMessage,
                         serverRootPathID,//1,
                         PSUEDORANDOMSOURCEPORT,
                         newMagmaIPv6Address,
                         (short) 8000);
        }//end try
      catch (Exception e)
        {
          System.out.println("Inside processMagmaController.SENDUPDATEDDIGESTPIB... "+e);
          e.printStackTrace();
        } //end catch
    }//end if
    else{
      System.out.println("Inside processMagma Controller: SENDUPDATEDDIGESTPIB is FALSE");
    } // end of if-else "isSENDUPDATEDDIGESTPIBselected"


    if (magmaControlGui.isSENDUPDATEDSTABLEVARIABLESselected()){
      System.out.println("Inside processMagma Controller: SENDUPDATEDSTABLEVARIABLES is TRUE");
    }
    else{
      System.out.println("Inside processMagma Controller: SENDUPDATEDSTABLEVARIABLES is
FALSE");
    } // end of if-else "isSENDUPDATEDSTABLEVARIABLESselected"


    if (magmaControlGui.isSENDUPDATEDRIDTONIDHTselected()){
      System.out.println("Inside processMagma Controller: SENDUPDATEDRIDTONIDHT is TRUE");
    }
    else{
      System.out.println("Inside processMagma Controller: SENDUPDATEDRIDTONIDHT is FALSE");
    } // end of if-else "isSENDUPDATEDRIDTONIDHTselected"


    if (magmaControlGui.isSENDUPDATEDUSERSLSINFOselected()){
      System.out.println("Inside processMagma Controller: SENDUPDATEDUSERSLSINFO is TRUE");
    }
    else{
      System.out.println("Inside processMagma Controller: SENDUPDATEDUSERSLSINFO is FALSE");
    } // end of if-else "isSENDUPDATEDUSERSLSINFOselected"

}//end processMagmaController()

 /**Margulis [sam]
  * This provides the liquid portion of Server to SAAM. Though this method is just a way to
  * forward the information obtained by the Control Executive to the Magma portion of this server,
  * it maintains privacy and visibility of magma's control, only to this server.
  * @param ControlExecutive The control executive of the server host
  * @param byte The payload of the MAGMA message being processed
  */
  public void processMagmaMessage( ControlExecutive controlExec, byte[] magmaPayload )
  {
    magma.processMagmaMessage(this, controlExec, magmaPayload);
  }//end processMagmaMessage()


/**Margulis, created on November 5, 2001[sam]
  * Stands up thread for dcm sending from the server.  This method is called
  * by the magmaMessageProcessor.
  * @return void.
  */
  public void standUpAutoConfigThread()
  {
    if (controlExec.getServerTable().hasServerByServerId(controlExec.getRouterId()))
    {
      autoConfig();
    }
    else
    {
      int tServerType = PRIMARY_SERVER;
      int tSC_MetricType = 0, tSC_CycleTime = 200 , tSC_GlobalWaitTime = 250;

      int tServerFlowID = controlExec.getServerTable().getServerRootPathID()+2;
      System.out.println("tServerFlowID = "+tServerFlowID);

      Configuration config = new Configuration ((byte) tServerType,
                                                 tServerFlowID,
                                                (byte) tSC_MetricType,
                                                 tSC_CycleTime,
                                                 tSC_GlobalWaitTime);


      processConfiguration(config);
    }

  }//standUpAutoConfigThread()

  /**Margulis, created on November 5, 2001 [sam]
  * Takes down thread for sending dcm from the server.  This method is called
  * by the magmaMessageProcessor to remove dcm functionality of server so that
  * this can be moved to another host.
  * @return void.
  */
  public void killAutoConfigThread()
  {
    controlExec.setInitHeartBeat(false);
    controlExec.setPrimaryServerStatus(false);
    controlExec.removeMagmaMenufromMainGui( );
    configThread.stop();

    newPIB.stopPIBTester(); // Xie-nov01

  }//end of autoConfig()

 //[sam]
 /**Modified by Margulis
  * Triggers DCM sending. and provides continuous refreshment of SAAM Region
  * with DCM messages.  Modified to use AutoConfigExec.broadcastDCM() and break
  * dependency on emulationTable. -crcb
  * @return void.
  */
  public void run()
  {
    final int initialWait = 30;

//    final int numOfCycles = 99;//Troy's validation  [TW]
//    final int numOfCycles = 1;//Paulo's validation  [PS]
    final int numOfCycles = 3;//normal default for debug/development
//    final int numOfCycles = 10000;  // large enough to test endurance

    gui.sendText("\n Server  will send first DCM after " + initialWait + " seconds");
    System.out.println("\n Server  will send first DCM after " + initialWait + " seconds");

    try
    {
      configThread.sleep(1000 * initialWait);
    }
    catch (InterruptedException ie)
    {
    }

    //[sam]
    displayServerBanner(controlExec.getPlayerId());

    for (int cycle = 0; cycle < numOfCycles; cycle++)
  {
   Vector hostInterfaces = this.controlExec.getInterfaces();
   System.out.println ("Number of interfaces is " + hostInterfaces.size());

   gui.sendText("Broadcasting DCM packets out of (all) interfaces(s).\n" +
      "Number of interfaces is " + hostInterfaces.size()); //Add this info msg -crcb

   IPv6Address nextHopToServer = null; //When servers "originate" their own DCM,
                         // there is no nextHopToServer. -crb
   AutoConfigurationExecutive ace = controlExec.getAutoConfigurationExecutive(); //-crcb

   //[sam]
   // needs to periodically look for routerId because the host router configuration may change
   serverRouterID = controlExec.getRouterId();

   ace.broadcastDCM(serverRootPathID, serverRouterID, nextHopToServer,
      metricType, globalTime, getDCMSeqNum()); incrementDCMSeqNumber(); //-crcb

   try
   {
    Thread.sleep(this.cycleTime); //from demoStation
   }
   catch (InterruptedException ie)
   {
    gui.sendText("thread sleep problem");
   }

/*********** Temporarily Disable for development -crc*************
   if (getServerType() == PRIMARY_SERVER)
   {  // Only if primary server
     gui.sendText("*****DIAG*** CYCLETIME EXPIRED***send probe now");
    // -- Mustafa's probing experiment
     try
    {
      gui.sendText("Server:DCM/UCM cycle done.*********************\n" +
         "    waiting 30Secs to send probe activation messages");
      Thread.sleep(30000);
      gui.sendText("Server:SENDING Probe Activations");
     }
    catch (InterruptedException ie)
    {
      gui.sendText("Thread sleep problem ");
     }
     initiateProbe();
   }//end if primary server
*******************************************************end disable**/
     //[sam] magma shutdown
     if (!controlExec.isServer()) break;
    }//end for (cycle = ...)

  }//end run()

//[sam]
/**
  * Displays the Server Banner frame. If server agent changes hosts,
  * this banner is closed and a new banner frame is opened.
  * @param none  public void displayServerBanner(String address)
  * @return void
  */
  public void displayServerBanner(String address)
  {
    bf.setFrameText(address +" HOSTS THE PRIMARY SERVER");
    bf.setVisible(true);

  }//end displayServerBanner()
```

```
//[sam]
/**
 * Kills the Server Banner frame.
 * @param none
 * @return void
 */
 public void killServerBanner()
 {
    System.out.println("==============> Inside server.killServerBanner: before killing banner
<==================");
    bf.setVisible(false);

 }//end killServerBanner()
```

15. saam\util\MAGMAAdminGui.java

```
//26Apr2001[Margulis] -Created... Refomated SAAMRouterGui class to handle MAGMAAdminGui[sam]

package saam.util;

import saam.server.*;
import saam.server.Server;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;     //for BadLocationException -crc
import javax.swing.border.*;


                //[sam]
                /**
                 * Title:        MAGMA Administration Gui
                 * Description:  This code provide the admin of a control panel for the MAGMA software
                 * @author       LCDR Scott Margulis
                 * @version 1.0
                 */
                /**
                 * A simple Graphical User Interface that consists of a JPanel with
                 * a JTextArea and a JTextField that can be updated with the sendText
                 * and setTextField methods respectively.
                 */
                public class MAGMAAdminGui extends JPanel {

                  public JPanel panel = new JPanel();

                  private boolean show = true;
                  private int rows = 6;
                  private int cols = 25;

                  private JTextArea messages = new JTextArea(rows,cols);
                  private JTextField tf = new JTextField();
                  private String title;
                  private int lineCountChopThreshold = 200; // default trigger limit to reclaim memory
                                                            // tweak for special cases in the constructor

                  //Added server handle to Magma Controller
                  private Server myServer;


                  //Added Server Variable to add server handle to Magma Controller
                  public MAGMAAdminGui(String title, Server server){
                    super(new BorderLayout());
                    this.title = title;

                    // adding server handle to Magma Controller
                    this.myServer = server;
```

```
  if (!title.equals("Admin Control")){
       add(messages, BorderLayout.CENTER);
       add(tf, BorderLayout.SOUTH);
       messages.setBackground(new Color(150, 172, 190));
       tf.setBackground(new Color(150, 150, 255));
       tf.setFont(new Font("helvetica", Font.BOLD, 12));
        setLocation(0, 0);
  } // end if
  else {
    //Added Server Variable to add server handle to Magma Controller
      MagmaControlGui cg = new MagmaControlGui(this.myServer);
      add(cg,BorderLayout.CENTER);
    setLocation(0, 0);
  } // end else

  if(title.equals("DigestPIB"))  //Tailor memory reclaimation
          lineCountChopThreshold = 20000;
}//end MAGMAAdminGui() Constructor


public String getTitle()
{ // -crcy
 return title;
} // -crcy

public void sendText1(String message)
{  //original sendText(), not used-crc
  if (show)
   messages.append(message + "\n");
}

public void setTextField(String message){
  if (show)
   tf.setText(message);
}

public JTextArea getTextArea(){
  return messages;
}

public JTextField getTextField(){
  return tf;
}

public void setTextFieldFontSize(int size)
{
  if (show)
    tf.setFont(new Font("helvetic", Font.BOLD, 12));
}

 /**
  * Adds capability of modifying default MAGMAAdminGui TextArea Fonts such as
  * those Frames employed by ProtocolStack menu items.
  * Method attempts to do this in a Swing event thread-safe manner.
  * It is unsafe to attempt to do this by other means once the Gui has been rendered "visible".
  * @param   fontType   Type examples which JDK guarantees across all platforms to be available:
  * <b>Monospaced</b>, <b>SansSerif</b>, <b>Dialog</b>, and <b>DialogInput</b>.  <p>
  * Try helvetica or others if cross-platform issues are moot.
  * @param   sizeInPoints  Exactly what it says.
  */
public void setTextAreaFont(final String fontType, final int sizeInPoints) { // -Why not? -crc
   if (show){
         Runnable setFont = new Runnable() {
             public void run() {
                 messages.setFont(new Font(fontType, Font.BOLD, sizeInPoints));
             }
         };
       SwingUtilities.invokeLater(setFont); setFont = null;
  }//end if show
}
```

```
  public String toString(){
    return title;
  }

  //*******START TEST CODE for swing thread****** -crc
        //following is test code to do gui updates properly
   /**
    * It is unsafe to modify the JTextArea once it has been rendered "visible" from outside
    * Swing's event thread.  This method supercedes the previous "unsafe" way AND also adds
    * some memory management to truncate MAGMAAdminGui text panels once they exceed a threshold
number
    * of lines.
    *
    * @param   guiStr  Current message to be painted in the JTextArea of this SAAMRoutergui.
    */
  public void sendText(String guiStr) { //If this tests out ok, we'll use it
        final String temp = guiStr +  "\n";
    final int chopThisManyLines = lineCountChopThreshold / 2;
    if (show) {
          Runnable setText = new Runnable() {  //if so, have this wait to execute in
                                                //Swing's event thread
             public void run() {
                 int lineCount = messages.getLineCount(); //Sampled when this actually runs!
                 if (lineCount > lineCountChopThreshold){
                    try{  // chop the String.  Hope that GC can reclaim. At least limit usage.
                     messages.replaceRange("Truncating.\n" , 0,
                          messages.getLineEndOffset(lineCount - chopThisManyLines) );
                    }
                    catch (BadLocationException ble) {
                     ble.printStackTrace();
                     System.out.println("****DIAG*** MAGMAAdminGui: lineCount= " + lineCount);

                    }
                    catch (IllegalArgumentException iae) {
                     iae.printStackTrace();
                     System.out.println("****DIAG*** MAGMAAdminGui: lineCount= " + lineCount);
                    }
                 }
                        messages.append(temp);
             } // end run()
          }; //end Runnable setTxt
      SwingUtilities.invokeLater(setText);
      setText = null;
    }//end if show
  }//end sendText
 //*******END TEST CODE for swing thread******

}//end MAGMAAdminGui Class
```

16. saam\util\MagmaControlGui.java

```
//26Apr2001[Margulis] -Created... Refomated SAAMRouterGui class to handle MAGMAControlGui[sam]

package saam.util;

// adding Server handle to this class
import saam.server.*;
import saam.server.Server;
import saam.net.*;
import saam.net.IPv6Address;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
```

```
//[sam]
/**
 * Title:        MAGMA Controller Gui
 * Description:  This code provide a control panel for the MAGMA software
 * @author       LCDR Scott Margulis
 * @version 1.0
 */
public class MagmaControlGui extends JPanel {

   public String title;

  public JPanel panel = new JPanel();
  public JComboBox nextMAGMAServerCB = new JComboBox(SAAMRegionHosts.getNextMAGMA());
  public JButton changeMagmaButton = new JButton();
  public JCheckBox changeMagma,sendDigest, sendSTABLEVARIABLES, sendHT, sendSLS;
  public JButton sendButton = new JButton();

  private boolean bCHANGEMAGMAselected = false;
  private boolean bSENDUPDATEDDIGESTPIBselected = false;
  private boolean bSENDUPDATEDSTABLEVARIABLESselected = false;
  private boolean bSENDUPDATEDRIDTONIDHTselected = false;
  private boolean bSENDUPDATEDUSERSLSINFOselected = false;
  private boolean bSendButtonSelected = false;

  private Server myServer;

  //Added Server Variable to add server handle to Magma Controller
  public MagmaControlGui(Server server) {

    super(new GridLayout(1,1));

    //Added Server Variable to add server handle to Magma Controller
    this.myServer = server;

    setFont(new Font("Courier", 1, 20));

    //SetPreferences
    TitledBorder tb= new TitledBorder("Enter IP Address of new MAGMA Server");
    tb.setTitleFont(new Font("Courier", 1, 20));

    //TOP
    nextMAGMAServerCB.setPreferredSize(new Dimension(500,65));
    nextMAGMAServerCB.setMinimumSize(new Dimension(500,65));
    nextMAGMAServerCB.setMaximumSize(new Dimension(500,65));
    nextMAGMAServerCB.setBorder(tb);
    nextMAGMAServerCB.setBackground(new java.awt.Color(166,162,187));
    nextMAGMAServerCB.setFont(new Font("Courier", 1, 20));
    nextMAGMAServerCB.setEditable(true);

    changeMagma        = new JCheckBox( "Change MAGMA Servers", false);
    sendDigest         = new JCheckBox( "Send Digest Info to all Router Agents", false);
    sendSTABLEVARIABLES = new JCheckBox( "Send Stable Variables to all Router Agents", false);
     sendHT            = new JCheckBox( "Send Significant Hashtables to all Router Agents", false);
     sendSLS           = new JCheckBox( "Send New SLS Info to all Router Agents", false);

     sendButton.setPreferredSize(new Dimension(120,60));
    sendButton.setMinimumSize(new Dimension(120,60));
    sendButton.setMaximumSize(new Dimension(120,60));
    sendButton.setText("Send");
    sendButton.setFont(new Font("Dialog", 1, 20));

     //Set Background Color and Border
       panel.setBackground(new java.awt.Color(166,162,187));
       panel.setBorder(BorderFactory.createEtchedBorder());
       panel.setLayout(null);

     //Set Position
       Insets insetsT = panel.getInsets();
       nextMAGMAServerCB.setBounds(360 + insetsT.left, 165 + insetsT.top,500,65);
       changeMagma.setBounds(360 + insetsT.left, 240 + insetsT.top,550,50);
```

```
      sendDigest.setBounds(360 + insetsT.left, 295 + insetsT.top,550,50);
      sendSTABLEVARIABLES.setBounds(360 + insetsT.left, 355 + insetsT.top,550,50);
      sendHT.setBounds(360 + insetsT.left, 405 + insetsT.top, 550, 50);
      sendSLS.setBounds(360 + insetsT.left, 460 + insetsT.top, 550, 50);
      sendButton.setBounds(530 + insetsT.left, 525 + insetsT.top, 120, 60);


    //Set CheckBox Color
      changeMagma.setBackground(new java.awt.Color(166,162,187));
      sendDigest.setBackground(new java.awt.Color(166,162,187));
      sendSTABLEVARIABLES.setBackground(new java.awt.Color(166,162,187));
      sendHT.setBackground(new java.awt.Color(166,162,187));
      sendSLS.setBackground(new java.awt.Color(166,162,187));

    //Set Font Size
      changeMagma.setFont(new java.awt.Font("Dialog", 1, 20));
      sendDigest.setFont(new java.awt.Font("Dialog", 1, 20));
      sendSTABLEVARIABLES.setFont(new java.awt.Font("Dialog", 1, 20));
      sendHT.setFont(new java.awt.Font("Dialog", 1, 20));
      sendSLS.setFont(new java.awt.Font("Dialog", 1, 20));


  //ADD
    panel.add(nextMAGMAServerCB);
    panel.add(changeMagma);
    panel.add(sendDigest);
    panel.add(sendSTABLEVARIABLES);
    panel.add(sendHT);
    panel.add(sendSLS);
    panel.add(sendButton);

    add(panel);


   //REGISTER EVENTS
   CheckBoxHandler checkHandler = new CheckBoxHandler();
   changeMagma.addItemListener ( checkHandler );
   sendDigest.addItemListener ( checkHandler );
   sendSTABLEVARIABLES.addItemListener ( checkHandler );
   sendHT.addItemListener ( checkHandler );
   sendSLS.addItemListener ( checkHandler );

   //Add sendButton Listener

   SendButtonHandler sendButtonHandler = new SendButtonHandler();
   sendButton.addActionListener( sendButtonHandler);


     Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation((d.width - getSize().width) / 2, (d.height - getSize().height) / 2);
    setSize(1000,800);

} // end of MagmaControlGui constructor


/**
 *  Added to handle changing of MagmaControl to new Router
 */
public IPv6Address getIPv6Address()
{
  SendButtonHandler sbh = new SendButtonHandler();
  String newMagma = sbh.getNewMagma();
  IPv6Address magmaAddress = new IPv6Address();

  if (newMagma.equals("Broadcast"))
  {
   //******************************************************************************
   //******************************************************************************
    System.out.println("Add a Message Box here to handle Broadcast"+
                            " not allowed to be selected with ChangeMagma!");
    //*****************************************************************************
    //*****************************************************************************
    newMagma = ("99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1");
   }
   try
   {
    magmaAddress = IPv6Address.getByName(newMagma);
   }
   catch (Exception e)
   {
     System.out.println(e);
   }

   return magmaAddress;

 }//end getIPv6Address()

 public JButton getSendButton()
 {
  return sendButton;
 }//end getSendButton()

 // static initializer for setting look & feel
 static {
   try {
     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   }
   catch (Exception e) {}
 }//look and Feel

 public class SendButtonHandler implements ActionListener {
   String newMagma = (String) nextMAGMAServerCB.getSelectedItem();
   public void actionPerformed( ActionEvent ae)
   {

       // added June 23, 2001 to keep track of selected items before executing
       // Send Button.  Should think about including Yes/No button to make sure
       // that Administrator wants to change Magma servers before this actually
       // happens and that the correct Next Magma server is selected.
       if ( bCHANGEMAGMAselected)
       {
        System.out.println("TRUE: Sending New Magma Server");
       }
       else
       {
        System.out.println("FALSE: Not Sending New Magma Server");
       } // end if-else

     if ( bSENDUPDATEDDIGESTPIBselected)
      {
        System.out.println("TRUE: Sending DigestPIB");
       }
     else
      {
       System.out.println("FALSE: Not Sending DigestPIB");
      } // end if-else


      if ( bSENDUPDATEDSTABLEVARIABLESselected)
      {
       System.out.println("TRUE: Sending Stable Variables");
      }
        else
      {
       System.out.println("FALSE: Not Sending Stable Variables");
     } // end if-else


        if ( bSENDUPDATEDRIDTONIDHTselected)
      {
```

```
    System.out.println("TRUE: Sending Stable Variables");
   }
  else
   {
    System.out.println("FALSE: Not Sending Stable Variables");
  } // end if-else

    if ( bSENDUPDATEDUSERSLSINFOselected )
   {
    System.out.println("TRUE: Sending New SLS");
  }
  else
   {
    System.out.println("FALSE: Not Sending New SLS");
  } // End else if

   System.out.println("Changing Magma Servers..."+newMagma);
   bSendButtonSelected = true;

   // complete Send Button action
    processMagmaController();
} // end actionPerformed


/**
 *  Added to handle changing of MagmaControl to new Router
 */
public String getNewMagma()
{
 return newMagma;
} // end getNewMagma


} // End SendButtonHandler

private class CheckBoxHandler implements ItemListener
{
 public void itemStateChanged ( ItemEvent ie )
 {
   if ( ie.getSource() == changeMagma && changeMagma.isSelected())
    {
     System.out.println("TRUE: Sending New Magma Server");
     bCHANGEMAGMAselected = true;
    }
    else if ( ie.getSource() == changeMagma && !changeMagma.isSelected())
    {
     System.out.println("FALSE: Not Sending New Magma Server");
     bCHANGEMAGMAselected = false;
    }
   else if ( ie.getSource() == sendDigest && sendDigest.isSelected())
    {
    System.out.println("TRUE: Sending DigestPIB");
   bSENDUPDATEDDIGESTPIBselected = true;
   }
   else if ( ie.getSource() == sendDigest && !sendDigest.isSelected())
    {
    System.out.println("FALSE: Not Sending DigestPIB");
    bSENDUPDATEDDIGESTPIBselected = false;
    }
   else if ( ie.getSource() == sendSTABLEVARIABLES && sendSTABLEVARIABLES.isSelected())
    {
   System.out.println("TRUE: Sending Stable Variables");
   bSENDUPDATEDSTABLEVARIABLESselected = true;
   }
   else if ( ie.getSource() == sendSTABLEVARIABLES && !sendSTABLEVARIABLES.isSelected())
    {
    System.out.println("FALSE: Not Sending Stable Variables");
    bSENDUPDATEDSTABLEVARIABLESselected = false;
   }
   else if ( ie.getSource() == sendHT && sendHT.isSelected())
```

```
        {
        System.out.println("TRUE: Sending Stable Variables");
        bSENDUPDATEDRIDTONIDHTselected = true;
      }
      else if ( ie.getSource() == sendHT && !sendHT.isSelected())
        {
        System.out.println("FALSE: Not Sending Stable Variables");
        bSENDUPDATEDRIDTONIDHTselected = false;
      }
      else if ( ie.getSource() == sendSLS && sendSLS.isSelected())
        {
        System.out.println("TRUE: Sending New SLS");
        bSENDUPDATEDUSERSLSINFOselected = true;
      }
      else if ( ie.getSource() == sendSLS && !sendSLS.isSelected())
        {
        System.out.println("FALSE: Not Sending New SLS");
        bSENDUPDATEDUSERSLSINFOselected = false;
      } // End else if

    } // End itemStateChanged

  } // End CheckBoxHandler

  public void processMagmaController()
  {
   // complete Send Button action
   myServer.processMagmaController(this);
  }

  public boolean isCHANGEMAGMAselected()
  {
   return bCHANGEMAGMAselected;
  }

  public boolean isSENDUPDATEDDIGESTPIBselected()
  {
   return bSENDUPDATEDDIGESTPIBselected;
  }

  public boolean isSENDUPDATEDSTABLEVARIABLESselected()
  {
   return bSENDUPDATEDSTABLEVARIABLESselected;
  }

  public boolean isSENDUPDATEDRIDTONIDHTselected()
  {
   return bSENDUPDATEDRIDTONIDHTselected;
  }

  public boolean isSENDUPDATEDUSERSLSINFOselected()
  {
   return bSENDUPDATEDUSERSLSINFOselected;
  }


}//End MagmaControlGui Class
```

LIST OF REFERENCES

[BRE99] Brenton,_Chris, Mastering Network Security, Sybex Network Press, 1999.

[CIG01] Cigital Labs, Definitions: Fault Tolerance, [http://www.cigitallabs.com/resources/definitions/fault_tolerance.html], 2001.

[DEC99] Deconinck , Dr. ir. Geert, Fault Tolerant Systems, ESAT / Division ACCA , Katholieke Universiteit Leuven, October 1999.

[FRE00] Freed, Les, "Germ Warfare at Work," PC Magazine, May 9, 2000, pg. 184-197.

[GRA91] Gray, Jim and Siewiorek, Daniel P., "High-Availability Computer Systems," *IEEE/Computer*, v.24, no. 9, p. 39-48, September 1991.

[???99] A Model, Analysis and Protocol Framework for Soft State-based Communication [http://www.cs.utah.edu/~kwright/paper_summs/network_papers/sstp-sigcomm99.html]

[JOH89] Johnson, BW, "Design and Analysis of Fault-Tolerant Digital Systems", Addison-Wesley, 1989.

[KAP99] Kaplan, Namik, "Prototyping of an Active and Lightweight Router," March 1999

[KAT99] Kati, Effraim, Fault-Tolerant Approach for Deploying Sever Agent-based Active network Management (SAAM) server in Windows NT Environment to Provide Uninterrupted Services to Routers In Case of Server Failure(s), March 2000.

[KON00] Kon, Fabio, et. Al., "2K: A Distributed Operating System for Dynamic Heterogeneous Environments," Department of Computer Science, University of Illinois at Urbana-Champaign, 2000.

[LEV95] Leveson, N.G., Safeware. System Safety and Computers, Addison-Wesley, 1995.

[LYU95] Lyu (Ed.), M., Software Fault Tolerance, John Wiley & Sons, 1995.

[MIS99] Mishra, Shivakant; Huang, Yanjun; and Kuntur, Harshavardhan; "DaAgent: A Dependable Mobile Agent System," Department of Computer Science, University of Wyoming, June 1999.

[NPS00] Introduction to Computer Security: Course Notes for CS3600, Naval Postgraduate School Center for Information System Security Studies and Research, Spring 2000.

[PRA96] Pradhan, D.K., Fault-tolerant Computer System Design, Prentice Hall, 1996.

[RAN95] Randell, e.a., "Predictably Dependable Computing Systems", Springer-Verlag, 1995.

[SAX99] Saxena, Nirmal R and McCluskey, Edward J., "Fault-Tolerance with Multithreaded Computing- A New Approach," Center for Reliable Computing, Standford University, 1999.

[SKO99] Skousen, Alan and Miller, Donald, "Using a Single Address Space Operating System for Distributed Computing and High Performance," Computer Science and Engineering Department, Arizona State University, Tempe, Arizona, 1999.

[STO99] Stone, Gary N.; Lundy, Bert; and Xie, Geoffrey; "Network Policy Languages: A survey and a new approach," 1999.

[THO01] Thompson, John M., "Hacking 101", November 2001.

[TIE99] Tiefert, Brian, Modeling Control Channel Dynamics of the SAAM Architecture using the NS Network Simulation Tool, September 1999.

[XIA98] Xipeng Xiao and Lionel M. Ni, "Internet QoS: the Big Picture," University of Michigan, 1998.

[XIE98] Xie, Geoffrey; Hensgen, Debra; Kidd, Taylor; and Yarger, John; "Effective Management of Integrated Services Using A Path Information Base," May 14, 1998.

[UAZ96] Liquid Software: A New Paradigm for Networked Systems, [http://www.cs.arizona.edu/liquid/]

[WIE93] Wiener, L.R., Digital *Woes*: why we should not depend on software, Addison-Wesley, 1993.

[WUD98] Wu, Daniel; Agrawal, Divyakant; and El Abbadi, Amr; "StratOSphere: Mobile Processing of Distributed Objects in Java," Department of Computer Science, University of California, Santa Barbara, 1998.

[WUT99] Wu, Thomas C., An Introduction to Object-Orientated Programming with Java, McGraw-Hill, 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Stephan Lapic
   Information Systems Analysis Branch D822
   Space and Naval Warfare System Center
   San Diego, CA 92152

4. CAPT Vic See
   SPAWAR Space Field Activity
   14675 Lee Road
   Chantilly, VA 20151-1715

5. LtCol Mike Folsom
   SPAWAR Space Field Activity
   14675 Lee Road
   Chantilly, VA 20151-1715

6. Mr. Burt Odner
   SPAWAR Space Field Activity
   14675 Lee Road
   Chantilly, VA 20151-1715

7. LCDR Kelvin Upson
   SPAWAR Space Field Activity
   14675 Lee Road
   Chantilly, VA 20151-1715

8. Chairman, Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5100

9. Professor Geoffry Xie, code CS/XG
   Naval Postgraduate School
   Monterey, CA 93943-5100

10. Mr. Daniel Warren, code CS/WD
Naval Postgraduate School
Monterey, CA 93943-5100

11. Mr. Cary Colwell, code CS
Naval Postgraduate School
Monterey, CA 93940-5100

12. Dr. Bernard J. Ulozas
Naval Air Warfare Center
Training Systems Division
12350 Research Parkway
Orlando, FL 32826

13. LT Frank Margulis
1011 NW 106th Avenue
Plantation, FL 33322

14. LCDR Scott Margulis
13054 Harvest Place
Clifton, VA 20124