NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ADVANCED QUALITY OF SERVICE MANAGEMENT FOR NEXT GENERATION INTERNET

by

Paulo R. Silva

September 2001

Thesis Advisor: Geoffrey Xie
Second Reader: Bert Lundy

Report Documentation Page

Report Date 30 Sep 2001	**Report Type** N/A	**Dates Covered (from... to)** -

Title and Subtitle Advanced Quality of Service Management for Next Generation Internet	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Paulo R. Silva	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Research Office Naval Postgraduate School Monterey, Ca 93943-5138	**Performing Organization Report Number**
Sponsoring/Monitoring Agency Name(s) and Address(es)	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract	
Subject Terms	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 200	

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Advanced Quality of Service Management for Next Generation Internet			**5. FUNDING NUMBERS**
6. AUTHOR(S): Paulo R. Silva			
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): DARPA and NASA			**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 us unlimited.			**12b. DISTRIBUTION CODE** Statement A

13. ABSTRACT *(maximum 200 words)*

Future computer networks, including the Next Generation Internet (NGI), will have to support applications with a wide range of service requirements, such as real-time communication services. These applications are particularly demanding since they require performance guarantees expressed in terms of delay, delay jitter, throughput and loss rate bounds. In order to provide such quality-of-service (QoS) guarantees, the network must implement a Resource Reservation mechanism for reserving resources such as bandwidth for individual connections. Additionally, the network must have an Admission Control mechanism, for selectively rejecting some QoS-sensitive flow requests based on resource availability or administrative policies.

The Server and Agent based Active network Management (SAAM) is a network management system designed to meet the requirements of NGI. In SAAM, emerging services models like Integrated Services (IntServ) and Differentiated Services (DiffServ), and the classical Best Effort service are concurrently sharing network resources. This thesis develops and demonstrates in SAAM a novel resource management concept that addresses the difficulties posed by QoS networks. With the new resource reservation and admission control approaches, the sharing mechanism is dynamic and adapts to network load. It ensures high resource utilization while meeting QoS requirements of network users.

14. SUBJECT TERMS: Next Generation Internet, Resource Management, Resource Allocation, Admission Control, Network Utilization, Quality of Service, Guaranteed Service, Integrated Service, Differentiated Service, Best Effort Service, Flows, Path Information Base.			**15. NUMBER OF PAGES:** 200
			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

ADVANCED QUALITY OF SERVICE MANAGEMENT FOR NEXT GENERATION INTERNET

Paulo R. Silva
Lieutenant Commander, Portuguese Navy
B.S., Portuguese Naval Academy, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: [illegible]
Paulo R. Silva

Approved by: [illegible]
Geoffrey Xie, Thesis Advisor

[illegible]
Bert Lundy, Second Reader

[illegible]
Chris Eagle, Chairman
Department or Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Future computer networks, including the Next Generation Internet (NGI), will have to support applications with a wide range of service requirements, such as real-time communication services. These applications are particularly demanding since they require performance guarantees expressed in terms of delay, delay jitter, throughput and loss rate bounds. In order to provide such quality-of-service (QoS) guarantees, the network must implement a Resource Reservation mechanism for reserving resources such as bandwidth for individual connections. Additionally, the network must have an Admission Control mechanism, for selectively rejecting some QoS-sensitive flow requests based on resource availability or administrative policies.

The Server and Agent based Active network Management (SAAM) is a network management system designed to meet the requirements of NGI. In SAAM, emerging services models like Integrated Services (IntServ) and Differentiated Services (DiffServ), and the classical Best Effort service are concurrently sharing network resources. This thesis develops and demonstrates in SAAM a novel resource management concept that addresses the difficulties posed by QoS networks. With the new resource reservation and admission control approaches, the sharing mechanism is dynamic and adapts to network load. It ensures high resource utilization while meeting QoS requirements of network users.

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

Many people deserve a word of thanks for making this research possible. First, I appreciate the confidence Dr. Geoffrey Xie placed in me during the past year. His invaluable aid and mentoring fostered motivation and enthusiasm, greatly contributed to the successful completion of this research. It has been an absolute privilege to work with him. I also thank Mr. Cary Colwell for his availability and patient assistance rendered during this study. Thanks, also, to the entire SAAM team. The sense of camaraderie and teamwork gave purpose to this research. I am also very grateful to my colleagues at NPS, together with their families, helping to balance academic and personal needs. Most of all, I thank my wonderful wife Paula, and children Inês and Pedro, for they unconditional support, patience and love, especially during the last two years. They are my inspiration and the true focus of my work.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION

Quality-of-Service (QoS) is definitely one of the most popular and challenging research topics in Internet computer networking nowadays. Today's Internet is an extremely versatile communication service for a wide range of applications. However, it is only aimed at providing best-effort (BE) service, where traffic is processed as quickly as possible, with no guarantees as to timeliness or actual delivery. Although this model perfectly fits many applications like e-mail or regular web browsing, it is commonly perceived that the best effort service cannot adequately support delay-sensitive and/or loss-sensitive applications, such as Internet telephony, multimedia conferencing, telemedicine and many others. These applications have in common the requirement for certain level of network QoS guarantees, measured by throughput, network delay and data loss rate.

In order to meet the QoS requirements of all potential traffic over the Internet, different approaches have been proposed. The Differentiated Services (DiffServ) approach is intended to provide a discrete number of service levels or classes, thus making it a scalable solution. Since DiffServ only prioritizes traffic among a limited number of classes, it does not truly provide for full QoS guarantees on a per user session basis. A different approach, the Integrated Service (IntServ) model is aimed to provide per-flow QoS guarantees, where a flow may represent the traffic generated by individual applications.

QoS guarantees can only be met if network resources such as link capacity and buffer space are previously allocated to requesting applications. Such a mechanism is called Resource Reservation. Networks must also have a way of selectively rejecting new flow requests based on resource availability or administrative policies. This mechanism is called Admission Control. Additionally, with the requirement to support multiple classes of service over the same infrastructure, networks have to provide a model for Link Sharing[1]. Collectively, resource reservation, admission control, and link sharing address

[1] Also termed Network Provisioning

the problem of Resource Management. Since applications with vastly different QoS demands will need to use the same infrastructure, resource management solutions must be flexible, adapting to different traffic mixes and load fluctuations. Moreover, a growing number of applications not only demand QoS guarantees but also generate an increasingly large volume of data, putting a huge load on current networks. Thus, another important objective of resource management is efficient use of resources.

Next Generation Internet (NGI) is one of several initiatives of the networking community to develop networks capable of both guaranteed and best effort services. The Server and Agent based Active network Management or SAAM, is an ongoing research project under the NGI initiative and it is sponsored by the Defense Advanced Research Projects Agency (DARPA) and National Aeronautics and Space Administration (NASA). SAAM implements and proves the feasibility of a server-based network management concept that addresses the resource management problem.

B. MILITARY RELEVANCE

The vision for future joint war fighting of US military is described in Joint Vision 2020 (JV2020) [1]. The concept of network-centric warfare (NCW), first conveyed in the JV2010 and carried forward in JV2020, represents a fundamental shift from the previous platform-centric warfare. Interoperability with external agencies and among forces of the allied nations is a growing necessity as recently proved with the combined NATO operations in the Balkans. Military operations in the current information age are organized around the NCW concept, through which information superiority translates into increased combat power. NCW is enabled by effectively networking sensors, decision makers and shooters to achieve shared awareness, increased speed of command and high levels of self-synchronization.

The NCW environment creates a wide range of network service requirements, only possible to meet through active and adaptive networks. SAAM is one of such networks being prototyped at the Naval Postgraduate School. Appendix A identifies some key enabler technologies of SAAM, which illustrate the importance of SAAM in the context of the NCW environment. The work of this thesis greatly contributes for the improvement of the SAAM concept and further extends the potential of SAAM to become a solution to the technical problems posed by NCW.

C. THESIS OBJECTIVES

The main objective of this research is to develop, implement and test a mechanism for managing the resources of quality of service capable networks, like SAAM. Special consideration is made with regard to multiple classes of QoS services sharing the same network infrastructure and the resource allocation mechanism for efficient utilization of network resources.

D. RESEARCH QUESTIONS

This thesis study provides answers for the following questions: (1) how can SAAM efficiently manage network resources while providing support for different classes of QoS traffic? (2) What is the impact of inter-service borrowing on the overall performance of QoS capable networks like SAAM?

E. SCOPE OF THE STUDY

Previous research efforts have already showed to be feasible the concept of a central network management authority for providing QoS guarantees to network users, as current SAAM prototype demonstrates. Cheng and Gibson [2], and Queck [3] created the foundations for a QoS management model. This thesis draws from their work, but is mainly focused on extending the potential of SAAM servers to efficiently and dynamically manage network resources while supporting different classes of guaranteed services. Specifically, the core data structure residing in the heart of SAAM, the Path Information Base (PIB), has been redesigned to provide support for a novel and more efficient resource management mechanism.

F. THESIS ORGANIZATION

The remainder of the thesis is organized into the following chapters:

- Chapter II – Background. This chapter introduces some of the underlying concepts related with quality-of-service and resource management in next generation networks. Additionally, a brief description of the SAAM concept is presented.

- Chapter III – Efficient Resource Management. This chapter details the development of the resource management mechanism required to manage resources of QoS capable networks like SAAM. In specific, a novel inter-service borrowing mechanism is presented.
- Chapter IV – Design. Details the design of the network management mechanism as applicable to SAAM.
- Chapter V – Implementation. Describes the implementation and integration details of the resource management mechanism developed in this thesis.
- Chapter VI – PIB Test and Performance Analysis. Describes the test and evaluation of PIB.
- Chapter VII – Conclusions and Recommendations. Concludes the thesis study with conclusions and the identification of areas of further study.
- Finally, several appendixes are included to support the thesis study.

II. BACKGROUND

Current trends in the development of real-time network applications indicate that the future Internet architecture will need to support a diversity of applications with different QoS requirements. Ongoing research on QoS has proven that enabling end-to-end QoS over the Internet introduces complexity in its overall functionality. Moreover, it affects areas like network management, business patterns of networking companies, and it also changes the way customer perceives the services offered by the network. Finding an efficient solution for end-to-end QoS over the Internet is not only one the most popular but also a very challenging research topic in computer networking today.

The current Internet architecture provides only simple services like IP addressing, routing, fragmentation and reassembling of IP datagrams. It relies on higher-level transport protocols for sequential and assured data delivery, and provides no guarantee as to timeliness and throughput of traffic. These services are widely known as best-effort services. Although these services are sufficient for traditional Internet applications like e-mail, web browsing or file-transfer, the same is not true for the emerging wave of applications like IP telephony, multimedia conferencing, or audio and video streaming. Consequently, the need to provide the current Internet with the mechanisms required to support QoS on the Internet is natural.

The SAAM project currently under development at the Naval Postgraduate School defines and implements a model of network management that provides a solution for the so-called Next Generation Internet (NGI). This chapter gives an overview of current Internet architectures related with providing QoS over the Internet, and how they fit the QoS requirement of next generation internet. Additionally, the SAAM model is briefly described.

A. QUALITY OF SERVICE ARCHITECTURES

The efforts to enable end-to-end QoS over the existing IP infrastructure, have led to the development of two different architectures: the Integrated Services (IntServ) architecture and the Differentiated Services (DiffServ) architecture. Although fundamentally different, both of these architectures are designed to support QoS over the Internet.

1. Integrated Service

The aim of the Integrated Service architecture is to provide customer on-demand QoS guarantees, e.g. bandwidth, delay and loss rate, and is ideally suited for real-time applications. The architectural design of IntServ is based on the notion that in order to fulfill the QoS requirement of the customers, network resources should be managed and controlled [4], thus implying that the admission control and resource reservation are the building blocks of this architecture. Event though IntServ provides the means for end-to-end QoS, it is still not widely implemented. Due to the maintenance of per-flow information, classification, reserving and management resources per-flow introduces complex scalability problems, especially at the core of high-speed networks where the number of flows to process is in the thousands to million ranges. Currently, IntServ has proven to be easily deployed only in small networks, where the number of IntServ flow is moderate and manageable.

2. Differentiated Services

The Differentiated Services architecture or DiffServ was developed to avoid the scalability problem and the complexity of IntServ. However, DiffServ only provides quality differentiation on traffic aggregates without strict guarantees on individual flows. This quality differentiation only offers to network users the guarantee that some traffic receives better service than others. The few predetermined levels of QoS are usually called traffic classes. As an example of a DiffServ model, classes may referred to as Gold, Silver and Bronze. Access control to service is regulated by pre-established service level agreements (SLA) between network access providers (ISPs) and their clients, which specify the service level agreed upon and the fees of the service.

B. RESOURCE MANAGEMENT

There are a number of emerging requirements for resource management in the Internet; these requirements include both link-sharing services and services for supporting QoS traffic. Link-sharing is required whenever network resources are to be shared among different agencies or traffic classes. Resource reservation and admission control are the base for providing QoS guarantees. All these three mechanisms play a vital role for the efficient management of resources in modern QoS networks.

1. Resource Reservation

As stated in [4], there is an inescapable requirement for networks to be able reserve resources, in order to provide QoS guarantees for specific user flows. The allocation of resources is typically done on a flow-by-flow basis as each new flow requests admission to the network. This in turn, requires flow-specific state information to be kept by routers along the path followed by the flow. In current Internet, based on the best-effort model, state information is only maintained by end applications. Such a stateless network is simple, robust and scales very well. A soft state approach for a QoS network would be desirable. Currently, the Resource Reservation Protocol (RSVP) is being used to support reservation of resources over an IP based network. It does so by simply providing a set of rules for the network and requesting applications to exchange information regarding the QoS requirement and admission and then to setup the required network resources.

2. Admission Control

Admission control is required to implement the decision logic that determines whether a new flow can be granted the requested QoS, without affecting guarantees already granted to previously admitted flows. In addition to ensuring that QoS guarantees are met, admission control may be used to enforce network administrative policies on resource reservations. Finally, admission control may also be an important tool on accounting and network administrative reporting. The simplest model of admission control would be the case in which the user asks for a specific QoS and the network either accepts or declines the request depending upon available resources. Since many applications can still be able to get acceptable service for different levels of QoS, the negotiation may often be more complex until a final flow spec is agreed upon.

As previously stated, the network is required to keep state information, i.e., remember the QoS parameters of past requests. One approach to admit a new flow would be to compute the worse case QoS bounds for each service based on such state information. A different approach, which is likely to provide better resource utilization, would rely on routers monitoring actual link usage by existing flows, and use this measured data as the basis for admission control. Although this approach as a higher risk of overload, it may yield more efficient link utilization. Furthermore, such a soft state approach may scale better for large-scale deployment.

3. Link Sharing

Link-sharing is a resource management mechanism created to manage network resources, such as bandwidth. Link-sharing relies on the basic assumption that bandwidth is a network resource that will always be limited. Some argue that when technologies like optic fiber be fully matured and widely adopted, bandwidth will no longer be of concern. However, the most commonly accepted counter-argument says that new applications will then exist to consume existing bandwidth. Link-sharing services are required whenever a network link is to be shared between agencies, protocol families, or service classes [5]. Multiple agencies may share the bandwidth of a link, where each one pays a fixed share of the costs, expecting to receive a guaranteed share of the link bandwidth. A second requirement is for link sharing of bandwidth on a link between protocol families. Controlled link sharing is desirable because protocols families have different responses to congestion. Another example for link-sharing is to share the bandwidth on a link between different classes of services, such as IntServ, DiffServ and Best Effort classes. All of the above three examples of link sharing explicitly deal with the aggregation of traffic on a link. While there are a number of different motivations for link-sharing in the network, the requirements for link-sharing are essentially the same, whether the link sharing is between organizations, service classes or families of protocols.

C. OVERVIEW OF SAAM

SAAM is an intelligent active network management system, developed to meet the requirements of next generation Internet. SAAM offers a solution to the problem of providing QoS guarantees while maintaining the simplicity, robustness and scalability of its underlying TCP/IP architecture. The SAAM model establishes a hierarchy of servers and routers that are grouped into hierarchically structured SAAM regions, as depicted in figure 2.1. SAAM servers assume the responsibility of all network management decisions within their own region, thus allowing for SAAM lightweight routers to focus only on traffic forwarding. This novel approach is the basis for implementing a complex service model that integrates multiple classes of QoS services. The service model for SAAM provides support for IntServ and DiffServ and with the current best-effort services.

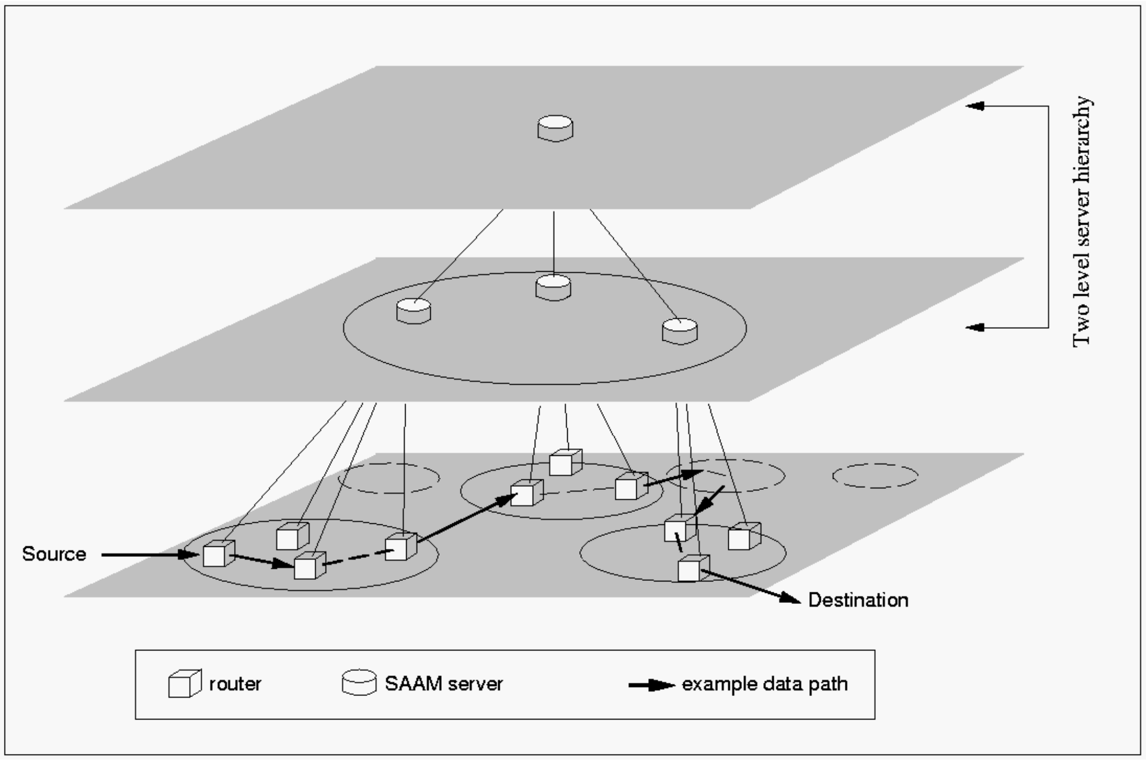


Figure 2.1 Hierarchical organization of SAAM.

1. SAAM Server

The SAAM server is a vital player within the SAAM architecture. The server dynamically builds and updates a knowledge base called Path Information Base (PIB) about its region upon receiving status information from its dependent routers. As the central repository of information for the SAAM region, the server is in a privileged position to make the best decisions in terms of granting network resources, optimizing network utilization, selection of the best routes, perform load balancing and any other network operational and administrative tasks, without having the burden of traffic processing only associated with the routers.

2. SAAM Router

Unlike standalone routers in existing IP networks, the SAAM router is conceptually simple and robust. Since all complex QoS routing decisions are taken by the server, the main task of the router is to process and forward traffic. Additionally, they monitor traffic and periodically report to the Server, the status of their links. The status information includes current link utilization, packet delay and data loss rate, on a per-service-class basis. Routing tables are updated by the server and contain information about the flow label and outbound interface to next router. SAAM routers are therefore required to maintain minimal state information, which provides for scalability.

3. SAAM Operation

The SAAM server initially discovers its region by flooding down the network with a special control message. Routers will in turn be aware of their server and report back their existence, which include all links between them and physical bandwidth available at their interfaces. The server uses this information to identify all paths within its region and to initially allocate bandwidth among all services classes supported, before starting accepting user traffic. Users request service at edge routers, which in turn will forward the request to the server. The admission control mechanism is then responsible for either accepting or rejecting the request based upon resource availability. Flow acceptance results in resources being allocated at each router along the selected path prior to send the new traffic. Edge routers maintain enough per-flow information, required for the purpose of traffic control and policing. At ingress or edge routers, packets are labeled to allow routers to forward their traffic. Additionally, IntServ packets will be inserted flow state information, which is then used and updated by the state-less high-speed core-routers when processing and forwarding traffic.

III. EFFICIENT RESOURCE MANAGEMENT

Chapter II introduced the concept of resource management and explained its key role for future networks. Admission control, resource allocation and link-sharing were identified as the building blocks for the activity of managing network resources in forthcoming QoS capable networks. This chapter builds on the previous chapter to describe an efficient and dynamic resource management model, which makes uses of those mechanisms to meet the requirements of the SAAM network. In providing support for multiple levels of QoS, e.g., those defined by the IntServ and DiffServ, a novel concept of inter-service borrowing of bandwidth will be presented as a solution to provide for a better resource management in such QoS networks.

A. RESOURCE MANAGEMENT APPROACHES

The current implementation of SAAM provides for multiple services classes, each offering one or more levels of services. These service classes, Integrated Service, Differentiated Service and Best Effort Service, will concurrently share network resources, e.g. link bandwidth. If the goal of efficient utilization of resources is to be met, then SAAM resource management must ensure that each service level takes an optimum share of network resources at any time, in order to maximize resource utilization. The sharing mechanism needs to be adaptive so that it adjusts to dynamic loads in each service class. It must also ensure that a minimum capacity exists at all times to fulfill the majority of future flow demands from higher priority service levels like IntServ while ensuring that lower service levels will not starve.

The complete link-share model for SAAM is depicted in Figure 3.1. In addition to the already mentioned IntServ, DiffServ and Best Effort service levels, the model also ensures that some bandwidth is reserved for signaling traffic, i.e. SAAM control traffic and finally, some other special traffic, named out-of profile (OP). OP traffic results from misbehaving user applications that generate traffic in excess of their previously negotiated QoS guarantees. In such cases, offending packets are marked as out-of-profile and pushed to the OP service level. Typically, OP packets receive the lowest priority. They are forwarded after all other traffic, i.e., when the link would be otherwise idle.

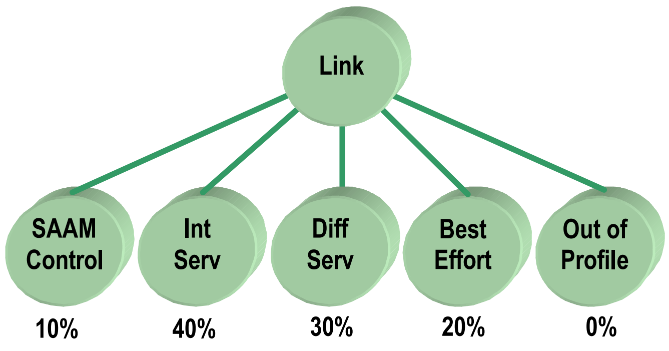


Figure 3.1 Link-share model in SAAM

1. Initial Resource Allocation

In the SAAM network, the SAAM server is responsible for the resource allocation. During initial startup and as part of the configuration cycle, participant routers advertise themselves and their interfaces to the SAAM server, which in turn develops the Path Information Base (PIB). The PIB is a data structure that will be used by the Server to determine all possible paths among all network nodes and to maintain link status information, as reported by routers. From the total bandwidth of each interface as reported by the hosting router, the Server then allocates a predetermined amount to each of the five service levels. This initial bandwidth is called base allocation per service level (BA) and is represented as a percentage of the total interface bandwidth. A typical link share per service level as previously used by SAAM is shown in Figure 3.2.

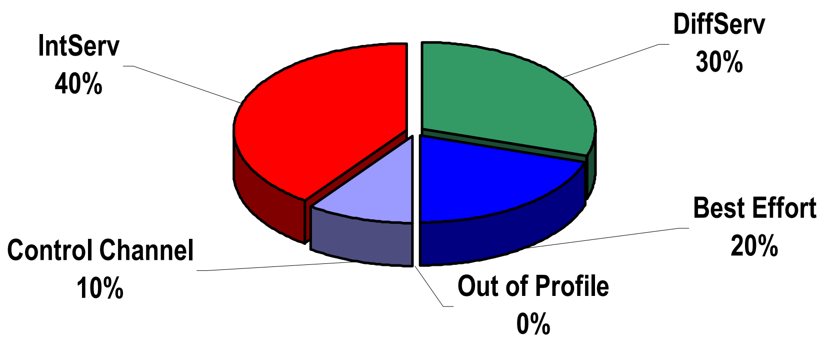


Figure 3.2 Typical link share percentages in SAAM

The share assigned for the SAAM control channels, is vital for the functioning of the entire SAAM region. As a goal in SAAM, the amount of control traffic should never be higher than the assigned 10% of the total bandwidth in each link. The non-conforming traffic of OP service pipe is typically given no bandwidth share. Consequently, OP packets are only served at each router when no other traffic is waiting for service. OP is the only service that may suffer from starvation when the network load is high. Finally, the share among the remaining three service levels shall then be assigned to fulfill the business and administrative goals of the networking service being provided, in an optimum and efficient way.

2. Developing an Optimum Share Model

Lets consider a single link with a maximum bandwidth capacity C_{max}, and with five service levels as mentioned before. Considering the individual capacities C_{CC}, C_I, C_D, C_B, C_{OP}, assigned respectively to SAAM control channels, IntServ, DiffServ, BE and OP service levels, the following expression holds:

$$C_{CC} + C_I + C_D + C_{BF} + C_{OP} \leq C_{\max} \tag{3.1}$$

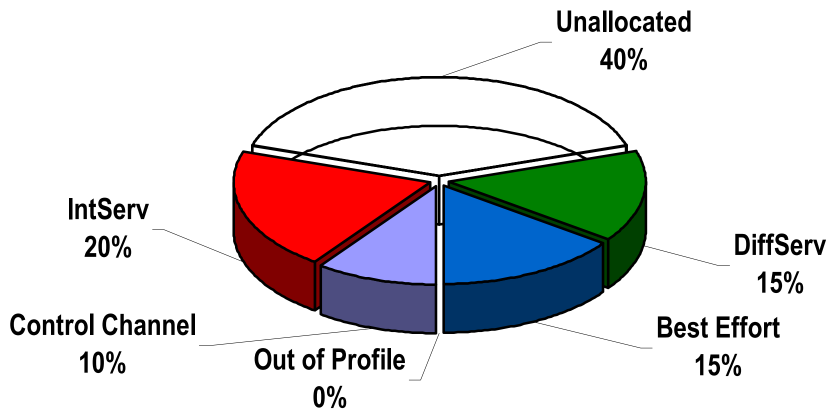


Figure 3.3 Link share model for SAAM with conservative base allocation

There are different alternatives for the distribution of the link share among the different classes of service. One of such alternatives could be performing static allocation, based on historical data and statistical models to forecast future load distribution for all service classes. While this solution requires the least effort after initial setup, it offers no flexibility and it is likely to result in poor network utilization. At some point, one service level may be using its entire allocated share and rejecting or buffering traffic while other services may be using a small fraction of their share, thus making an inefficient use of resources.

A different approach would be to make a conservative start by allocating little or minimum capacity for each of the service classes, which results in some of the link bandwidth not being assigned to any service – unallocated bandwidth (C_u). As new network users are admitted to the different classes of service, the utilization of each service level will then increase and eventually, reach the maximum initial capacity available for the class. From then on, the resource management mechanism ensures that new resource demands can be granted by claiming the initial unallocated capacity as long as there is still some available. The same way classes of service are allowed to claim unallocated bandwidth as needed, they should also release that bandwidth as soon as it is no longer required, thus making it available to other service classes. By dynamically resizing the capacity allocated for a service class, the network automatically adjusts to the profile of the traffic load, which increases the overall network utilization.

The previous approach may be further extended to achieve event better network utilization. In some cases, a heavily loaded service level (say IntServ) may be using all of its base allocation and the entire link unallocated capacity while another service class (say DiffServ) is using only a small fraction of its base allocation. Intuitively, if the possibility for a dramatic increase of DiffServ traffic in the near future is remote, then some fraction of DiffServ's unused base allocation bandwidth could be made available for borrowing by IntServ. This solution would yield even better link bandwidth utilization and decrease the total number of rejections of user flow requests, especially under high network loads. This novel approach is named Inter-service Borrowing and will be explained in detail in the following sections.

B. CONSERVATIVE BASE ALLOCATION

The previous section introduced the concept of a conservative start for allocating bandwidth, which in turn leads to some fraction of the link capacity not being allocated.

The initial inequality as expressed in equation 3.1 should be modified to include the unallocated capacity, thus turning the initial equation into

$$C_I + C_D + C_{CC} + C_{BF} + C_{OP} + C_U = C_{\max},$$

where C_u refers to the unallocated capacity.

At this point, it is of interest to differentiate the concept of initial allocation (base allocation) and current allocation. Base allocation will be denoted by C_{Xo} (for service class *X)* and is a constant value that is assigned on network startup as opposed to C_X which is the current allocated value for service *X*. The current allocation value varies over time, depending on the load of the respective service, and is initially set equal to the base allocation C_{Xo}, i.e., at the startup the following two equalities exist:

$C_{I_o} = C_I$ for IntServ

$C_{D_o} = C_D$ for DiffServ

According to the SAAM model, some of the services will have a constant share of the link bandwidth. We will therefore assume that SAAM control channel and Best Effort services will have a fixed share allocation over time[2]. Additionally, Out of Profile traffic will have no capacity specifically allocated and therefore $C_{OP} = 0$.

Based on the above assumptions, the initial setup of the network is therefore performed in accordance with the following equation:

$$C_{Io} + C_{Do} + C_{CC} + C_{BF} + C_U = C_{\max}. \tag{3.2}$$

With the assumption that C_{CC} and C_{BF} remain constant over time, let us isolate three dynamic terms of interest of this discussion. It will be,

$$\begin{aligned} C_{Io} + C_{Do} + C_U &= C_{\max} - (C_{CC} + C_{BF}) \\ C_{Io} + C_{Do} + C_U &= kC_{\max} \\ C_{Io} + C_{Do} + C_U &= C'_{\max} \end{aligned}$$

IntServ and DiffServ classes of service will use network resources on demand. The initial capacity allocated for these two classes (C_{Io} and C_{Do}) may be used either

[2] An alternative for BE in SAAM, may be the utilization of unused capacity allocated for Control Channels, IntServ and DiffServ. This could be achieved with a work-conserving packet schedule algorithm.

partially or totally. Whenever any of these classes use their base allocation share, they become eligible for taking some of the unallocated capacity as required. Eventually, no more unallocated space is available, which is then the case to try inter-service borrowing, between these two classes. The next section will address this sequence in detail.

C. ADMISSION CONTROL SEQUENCE

As suggested in the previous section, the process of dynamic resizing the bandwidth assigned to a service level goes through a sequence of different phases. This readjusting process is triggered by the arrival of a new IntServ or DiffServ flow request and is part of the admission control mechanism. The admission control procedure can be divided in three distinct steps, each corresponding to a distinct admission phase, as illustrated in figure 3.4, and they are:

- Step 1 - Direct admission
- Step 2 – Dynamic growing
- Step 3 – Inter-service borrowing

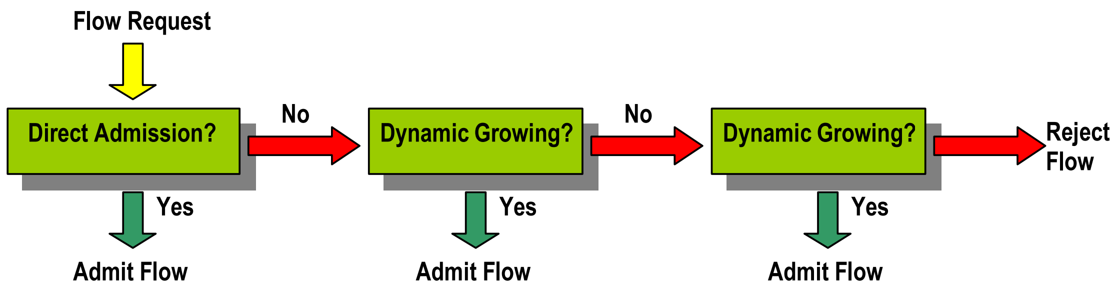


Figure 3.4 Admission control sequence.

1. Direct Admission – Admission Step 1

Direct admission is the first phase of the admission. It is the most basic admission control step, because a single service class is considered. A new flow is admitted at this point if there is enough available bandwidth from the initial base allocation for that service. All new flow requests go through this phase. If admission fails at this stage, it means the service load has increased over the initial base allocation for the service and the flow request must therefore proceed to phase 2. Recall that variables C_I and C_D represent the current allocated capacity for IntServ and DiffServ, respectively. As already stated, initially we have:

$$C_I = C_{Io} \text{ and } C_D = C_{Do}$$

Let's now consider the sequence of steps required to admit a new flow request. For simplicity and without loss of generality, it is assumed that new flow request belong to IntServ. Only two classes of services will be considered: IntServ and DiffServ. However, the following deduction could easily be extended to *n* different classes of services. Considering that a new flow request *f** arrives and has a bandwidth requirement of R_{f*}, the admission criteria for this flow is based on the equation

$$\sum_{f \in B_I} R_f + R_{f*} \le \alpha_I C_I \tag{3.3}$$

where B_I is the set of currently active flows of service I (IntServ).

Equation 3.3 introduces also the factor α_I, which is referred to as the load factor for IntServ. Typical network load management uses this load factor to prevent maximum load to approach the full physical link capacity. Although the actual value for this factor is not relevant for the present discussion, it is included in the expressions herein presented for completeness. Since $0 < \alpha_I \le 1$, we can at most use $\alpha_I C_I$ capacity which is less than or equal to C_I. Whenever inequality 3.3 holds then the new flow *f** may be admitted and the admission procedure does not need to go into next step.

2. Dynamic Growing – Admission Step 2

Dynamic growing refers to the ability for the link share mechanism to dynamically adapt to changing service level load. By starting with a conservative small base allocation of bandwidth for each of the service levels, this step ensures that each service is allowed to received more resources from the unallocated capacity, only when they need them. Inversely, when those resources are no longer required for that service, they are released, thus made available for other services. Suppose inequality 3.3 is not satisfied during the previous step. It means that there is not enough capacity C_I to satisfy the new request. This next step now will attempt to use some of the possible unallocated capacity. The admission condition then becomes

$$\sum_{f \in B_I} R_f + R_{f*} \le \alpha_I \left(C_I + \beta C_U \right) \tag{3.4}$$

The difference from equation 3.3 is that we now try to increase C_I by a small amount, enough to accept the new flow request. Intuitively, we need both

$C_U > 0$ and $\beta > 0$

The amount of unallocated bandwidth claimed to satisfy the new request should be minimum. Therefore, from inequality 3.4 we can derive the optimal value for β as follows:

$$\sum_{f \in B_I} R_f + R_{f*} = \alpha_I C_I + \alpha_I \beta C_U \tag{3.5}$$

$$\beta = \frac{\sum_{f \in B_I} R_f + R_{f*} - \alpha_I C_I}{\alpha_1 C_U}$$

The new flow may then admitted if:

$C_U > 0$

and

$0 < \beta \le 1$

In the case of $\beta = 1$, the new flow is admitted taking all of the unallocated capacity. After admission, the bandwidth allocation for IntServ is increased by a given amount and the unallocated capacity is decreased by the same amount. Respective variables must be updated in the following order:

$C_I = C_I + \beta C_U$

$C_U = C_U - \beta C_U$

By doing this, we are enabling dynamic growing of the total link share for a given class of service, which may happen as long as there is enough unallocated space.

Whenever it is no longer possible to dynamically grow this way, the admission procedure proceeds to step 3.

3. Inter-service Borrowing – Admission Step 3

Inter-service borrowing is to be considered only when the load of one service level peaks and other services are not using their entire base allocated bandwidth. When the load of a given service level is very high, it is possible that the sum of the base allocation with the unallocated space is not enough to fulfill the resource demand for that service. In such case, the admission process fails both phase 1 and phase 2 steps. However, under some circumstances it might be possible to borrow some capacity from other services. This inter-service borrowing mechanism is based on a statistic model for the network traffic and will be explained in the next section. For now, it matters only to focus on the new admission condition applicable to this last case.

The admission condition for phase 2 - equation 3.4 - now no longer holds. Providing that some capacity can be borrowed from some other service, that borrowed bandwidth should be added to the right side of equation 3.4, which then becomes:

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + \beta C_U + \rho_D C_D \right) \tag{3.6}$$

In equation 3.6, C_I refers to the current total allocation for the service I (IntServ), βC_U is the available unallocated capacity, and finally, $\rho_D C_D$ represents the maximum bandwidth that service D (DiffServ in this case) makes available for inter-service borrowing. It is important to note that at this stage in the admission process, the unallocated capacity C_U is either zero or very small, and is insufficient to meet the new flow requirement. That remaining capacity should therefore be completely used before going into inter-service borrowing, which explains the βC_U in equation 3.6. For this reason, $\beta=1$ and the admission condition can me modified to become:

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + C_U + \rho_D C_{D_o} \right) \tag{3.7}$$

This new equation differs from equation 3.5 by the new term $\rho_D C_{Do}$, where C_{Do} is the base allocation for service D (DiffServ) and ρ_D the fraction of that capacity that might be borrowed. Since the base allocation for DiffServ is constant, the solution for the inequality depends upon the value of ρ_D. As already stated, next section will detail a way of calculating ρ_D. For now, assume that we have obtained a value for ρ_D. With this value, either the equation does not hold and the new flow request is rejected, or instead, the new flow can be admitted.

If the new flow is accepted, inter-class borrowing takes place. For instance, using the previous example, it would mean that DiffServ has its available bandwidth partially reduced until resources are released. In order to keep track of this capacity transfer, variables C_I and C_D must be updated to reflect the new allocation transfer. The bandwidth transfer from DiffServ to IntServ in this example should be no more than the necessary to admit the new flow. This means that we are interested in the minimum value for ρ_D that satisfies the inequality 3.7, i.e.

$$\rho_D' = \min(\rho_D)$$

such that,

$$\sum_{f \in B_I} R_f + R_{f*} = \alpha_I (C_I + C_U + \rho'_D C_D) \tag{3.8}$$

Expressing ρ' as the dependent variable, equation 3.8 becomes:

$$\rho_D' = \frac{\sum_{f \in B_I} R_f + R_{f*} - \alpha_I C_I}{\alpha_1 C_D} - \frac{C_I + C_U}{C_D} \tag{3.9}$$

After the admission of the new flow, the allocation variables should be updated as follows:

$$C_I = C_I + C_U + \rho_D' C_D$$

$$C_U = 0$$

$$C_D = C_D - \rho_D' \mathrm{C_D}$$

D. INTER-SERVICE BORROWING EXPLAINED

If the network load were easily predictable and constant, it would be possible to adjust the resource shares of different service levels, in an optimum way, such that the resource utilization would be maximum and the flow rejection rate minimum. However, in the real world, this is never the case and there is always some uncertainty on traffic prediction. Traffic load fluctuations are more than likely and therefore the resource management algorithm should be dynamic and adaptive, in order to maximize resource utilization and ensure availability, even when network load is unbalanced among service levels. Inter-service borrowing seems a logic step in this regard.

1. Aggregated Flow Distribution

As discussed in the previous section, the three-step admission procedure relies on inter-service borrowing at last, to admit a new flow at extreme load levels. The admission condition for inter-service borrowing (equation 3.7) shows that the admission decision is a function of the coefficient ρ, defined as the required fraction of the base bandwidth allocation of the service level to borrow from. A service should make available some of its base allocation capacity only when there is a minimal probability for that service to use that capacity in the near future. The basic idea is to obtain some quantitative measure of the bandwidth that a service can make available for borrowing, without reducing much of its own ability to accept new flows.

Consider that network clients request flows with different throughput requirements. We may assume that the throughput requirements of individual flows follow some type of unknown distribution with the following characteristics:

μ_O - mean value of an individual flow request

σ_O - standard deviation of the flow request distribution

If we now consider that ($R_1, R_2, R_3, \ldots, R_n$) is a random sample from the above distribution and that they represent at any time, the individual throughputs of n active flows, then:

$\overline{R}$ is the mean value of the sample.

According to the Central Limit Theorem [6], $\overline{R}$ is approximately a normal distribution with:

$E(\overline{R}) = \mu_{\overline{R}} = \mu_o$ (mean)

$$V(\overline{R}) = \sigma_{\overline{R}}^2 = \frac{\sigma_o^2}{n} \qquad \text{(variance)}$$

$$\sigma_{\overline{R}} = \frac{\sigma_o}{\sqrt{n}} \qquad \text{(standard deviation)}$$

The total throughput T of the set of n active flows is the sample total and can easily be obtained by the equation bellow:

$$T = \sum_{i=1}^{n} R_i$$

and

$$E(T) = nE(\overline{R}) = n\mu_o \qquad \text{(expected mean value for total throughput)}$$

$$V(T) = nV(\overline{R}) = n\sigma_o^2 \qquad \text{(expected variance)}$$

$$\sigma_T = \sqrt{n}\sigma_o \qquad \text{(standard deviation)}$$

2. Probabilistic Bandwidth Utilization

Now consider that the number of flows n is constant and that the throughput requests of individual flows follow the same distribution with mean μ_T and standard deviation σ_T. For any given probability p, we are now interested to obtain a value x such that

$$P(T \le x) = p\,.$$

So, it will be

$$P\left(Z \le \frac{x - \mu_T}{\sigma_T}\right) = p$$

$$\Phi\left(\frac{x - \mu_T}{\sigma_T}\right) = p$$

$$\Phi\left(\frac{x - n\mu_o}{\sqrt{n}\sigma_o}\right) = p$$

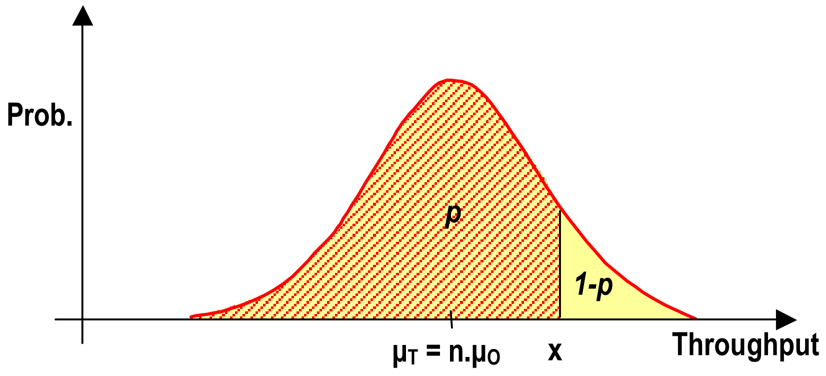


Figure 3.5 Aggregated throughput distribution as a normal distribution. Total throughput T of all n active flows is less than x with a probability p.

As an example, lets consider a probability *p = 0.9495 ≈ 95%.*

We then have

$$\Phi\left(\frac{x - n\mu_o}{\sqrt{n}\sigma_o}\right) = 0.9495$$

$$\frac{x - n\mu_o}{\sqrt{n}\sigma_o} = 1.64$$

$$x = n\mu_o + 1.64\sqrt{n}\sigma_o \qquad (3.10)$$

or

$$x = \mu_T + 1.64\sigma_T \qquad (3.11)$$

Equations 3.10 and 3.11 allow us to obtain a value of *x* based on the probability of 95%. Different values could be selected. Depending on what data is available, either equation might be used. For instance, if we precisely know the mean flow request, the standard deviation of the distribution of those flow requests and the number of active flows, equation 3.10 is to be used. If otherwise we know nothing about the original flow distribution or if the number of flows is not exactly known, we may otherwise have to use equation 3.11.

3. A Scalable Soft-State Solution for SAAM

One of the underlying goals of SAAM is to keep the minimal network status information, i.e. maintain a soft state, thus providing for robustness and scalability, without too much increase of complexity. For instance, once new flows are admitted, no information is kept with regard to the actual number of active flows. In equation 3.10, x is expressed as a function of the number of active flows and therefore this equation cannot be used in SAAM. The only alternative is equation 3.11, which expresses x as a function of the two parameters μ_T and σ_T, that characterize the distribution of the total throughput T.

We can estimate μ_T with

$$\mu_T = \sum_{f \in B} R_f \ ,$$

where B is the set of currently active flows of this service class. Equation 3.11 now becomes

$$x = \sum_{f \in B} R_f + 1.64\sigma_T \tag{3.12}$$

The SAAM server continuously receives link state information from routers. This information contains not only bandwidth utilization per class of service but also other QoS metrics like data delay and data loss rate. This soft state approach decreases the complexity at the core routers because no state information is maintained at flow level. No mechanism to explicitly notify network about termination of a flow exists. In fact, once a new flow is admitted and resources are reserved accordingly, the process of releasing those resources is implicitly part of the periodic link state advertising. The central SAAM server only maintains updated information about the throughput of the set of active flows for each router interface. Consequently, the term $\sum_{f \in B} R_f$ is easily obtained for every class of services. However, in 3.11 we still have the unknown standard deviation parameter of the flow aggregate distribution. With the knowledge of the number of active flows and the characteristics of the individual flow request distribution, it would be straightforward to compute the standard deviation. However, since we assumed to know nothing about the number of flows nor about the original distribution, a different approach must be followed.

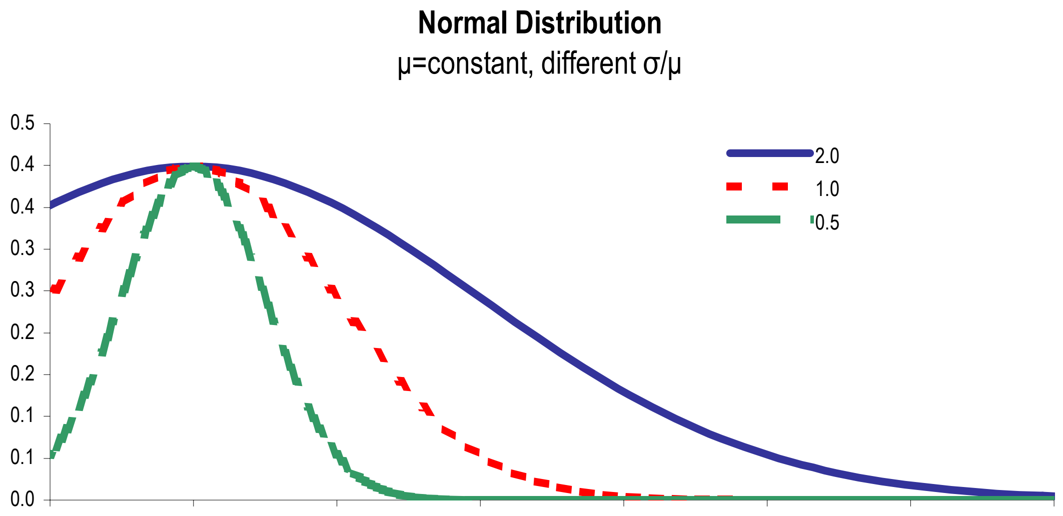


Figure 3.6 Normal distribution for different three ratios σ/μ.

It was already shown that the total throughput distribution and the individual flow request distribution are related with each other and that the following equations apply:

$$\sigma_T = \sqrt{n}\sigma_0 \tag{3.13}$$

and

$$\mu_T = n\mu_0 \tag{3.14}$$

Dividing equation 3.13 by equation 3.14 we obtain

$$\frac{\sigma_T}{\mu_T} = \frac{\sqrt{n}\sigma_0}{n\mu_o}$$

$$\frac{\sigma_T}{\mu_T} = \frac{1}{\sqrt{n}}\frac{\sigma_0}{\mu_o} \tag{3.15}$$

It is reasonable to assume that some information is known about the profile of the individual flow requests. From past data, there should be at least knowledge about the mean value of the bandwidth request per flow. However, we will take a conservative approach by assuming that we only know something about the shape of the flow request distribution, i.e. the ratio σ_o/μ_o.

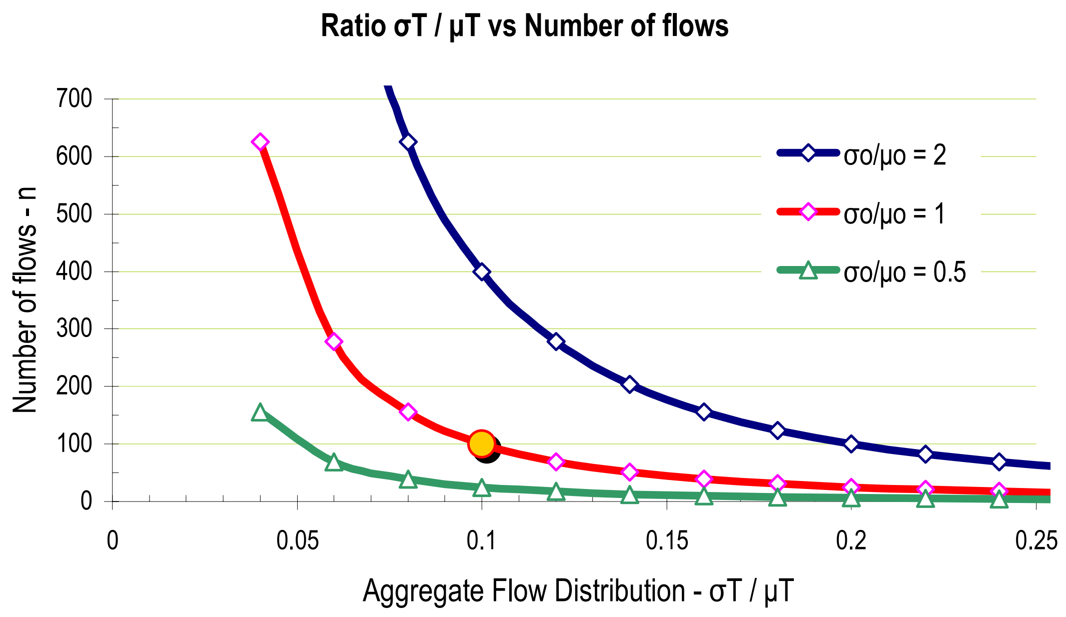


Figure 3.7 Ratio σ_T/μ_T versus the number of participant flows for three different shapes of flow request distributions.

Figure 3.7 shows a graph plotted with equation 3.15. For the aggregated throughput distribution, the number of flows as a function of the ratio σ_T /μ_T, for three different shapes of the original flows request distribution, i.e. for the ratios σ_o/μ_o = 2, 1 and 0.5.

Lets now assume that the standard deviation of the original flow distribution is equal to its mean, i.e. $\sigma_o / \mu_o = 1$. This is considered a conservative approach since the real distribution is likely to be much more concentrated about the mean, i.e. with a ratio $\sigma_o / \mu_o < 1$. Additionally, it is assumed a substantial number of active flows, for example n = 100. Referring to the graph in figure 3.7, these assumptions apply to the (0.1,100) coordinate point, marked with a big circle. By inspecting the graph, it can be observed that as the number of flows increases or the σ_o / μ_o ratio decreases (narrow original flow distributions) the ratio σ_T /μ_T, also decreased. With the given assumption, equation 3.14 becomes

$$\frac{\sigma_T}{\mu_T} = \frac{1}{\sqrt{100}} \times 1 = 0.1$$

$$\sigma_T = 0.1\mu_T$$

We can now replace σ_T in equation 3.12, which becomes

$$x = \sum_{f \in B} R_f + 1.64 \times 0.1\mu_T$$

and finally conclude with:

$$x = \sum_{f \in B} R_f + 0.164 \sum_{f \in B} R_f$$

or

$$x = 1.164 \sum_{f \in B} R_f \qquad (3.16)$$

In summary, when the number of active flows equal or exceeds 100, and the original flow request distribution satisfies $\sigma_o \leq \mu_o$, we can ensure with a certainty of 95% that the throughput requirement of this service class will not exceed the value x as given by equation 3.16. Figure 3.8 is a pictorial representation of this concept applied to the DiffServ class.

As discussed in the previous section, the admission condition for inter-service borrowing was given by

$$\sum_{f \in B_I} R_f + R_{f*} \leq \alpha_I \left(C_I + C_U + \rho_D C_{D_o} \right)$$

The term $\rho_D C_D$ represents the maximum capacity that can be borrowed, can now be expressed as follows:

$$\rho_D C_D = \alpha_D C_D - x$$

$$\rho_D C_D = \alpha_D C_D - \left(\sum_{f \in B_D} R_f + 0.164 \sum_{f \in B_D} R_f \right)$$

$$\rho_D C_D = \alpha_D C_D - 1.164 \sum_{f \in B_D} R_f$$

and the complete admission equation becomes

$$\sum_{f \in B_I} R_f + R_{f*} \le \alpha_I \left(C_I + C_U + \alpha_D C_D - 1.164 \sum_{f \in B_D} R_f \right) \tag{3.17}$$

Equation 3.16 is therefore applicable for a probability of 95% and a number of active flows $n \geq 100$.

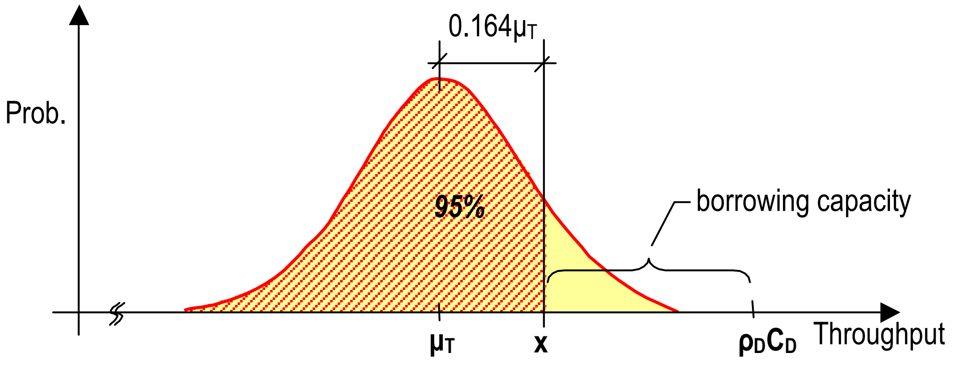


Figure 3.8 Pictorial representation of the borrowing capacity of DiffServ, based on the probability of 95%.

E. CHAPTER SUMMARY

QoS capable networks and the consequent need for resource allocation present new challenges for the efficient use of network resources. The link-sharing mechanism allows multiple service levels to share the same link, however in providing such versatility, dynamic and adaptive share mechanisms must be used in order to preserve high levels of resource utilization. The concept of resource management described in this chapter, allows for multiple services levels to adjust their link share by dynamically resizing their allocated capacity and using the unallocated capacity until they reach an point of equilibrium, which self adjusts to changing network load conditions. Moreover, by means of inter-service capacity borrowing, it is possible to further enhance resource utilization. It should be understood, however, that the complexity associated with this resource management approach only makes sense because it is assumed that network resources are limited and we want to maximize resource utilization levels. Next chapter will detail the implementation of all these concepts in the current SAAM prototype.

IV. SAAM QOS MANAGEMENT DESIGN

In the previous chapter, the theory behind the new advanced resource management concept was explained in detail. As the main component of the SAAM server, the Path Information Base (PIB) defines the server behavior in all QoS management tasks. This chapter will discuss the design specifics of PIB and the required changes to enable inter-service borrowing. The chapter starts by first describing the overall design of a SAAM server. Then the PIB and its main functions are described in detail. Finally, we conclude with the design details of inter-service borrowing and PIB re-design in support of the new QoS management capabilities of SAAM.

A. THE SAAM SERVER

1. Overview

The basic component of the SAAM prototype is the SAAM router. In its simplest form, the SAAM router is a Java based application that uses a layered architecture to emulate the full Internet Protocol stack and all the functionality associated with a SAAM router. Java multithreading is highly used, which allows all different emulated router components, to function in parallel and to establish data and control channels among them. Although the router model is designed to work over current IPv4 networks, the current version of the SAAM prototype supports only IPv6. The emulated physical layer performs the bridge interface between the underlying IPv4 infrastructure and the IPv6 SAAM prototype.

The SAAM server is an extension of a SAAM router. The server application resides in the outer layer of a SAAM router, i.e. the application layer of the protocol stack. That router is then able to perform all the normal functions associated with a router plus all tasks specific to the SAAM server. Figure 4.1 depicts this modularized approach.

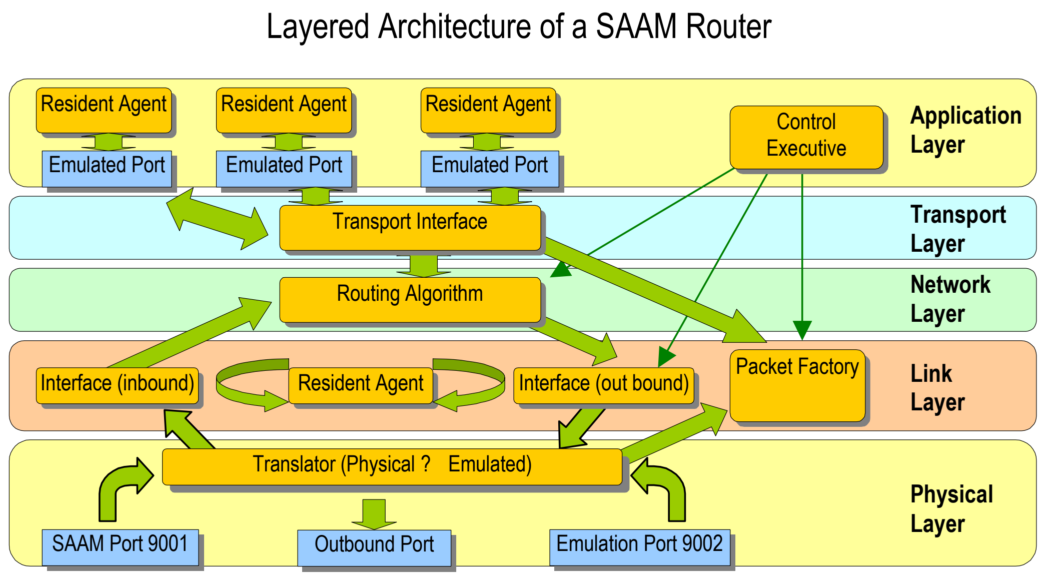


Figure 4.1 Layered architecture of a SAAM router. The SAAM server is a SAAM router with a resident agent application that performs server specific operations.

2. Main Functions of the SAAM Server

The server model was originally developed by NPS graduates Vrable and Yarger [7]. Since then, several enhancements have been made, namely the introduction of the basic QoS management capability by Queck [3], signaling channels by Akkoc [8], backup server functionality by Kati [9], and dynamic PIB generation by Cheng and Gibson [2]. The functions of the current SAAM server prototype may be divided into three types as described in the sections that follow.

a. Active Network Control

When the server first initiates, it tries to establish communication with other SAAM players (SAAM routers and eventual SAAM backup servers). This is done during the initial configuration cycle, by sending special control messages through all server interfaces. Routers in turn will receive this message and continue flooding the rest of the network until edge routers are reached. This initial configuration cycle closes with messages being exchanged among routers and back to the server. When the initial cycle is complete, all participating routers will have registered the presence of a server in the region and updated their routing table to create signaling channels for forwarding control traffic from and to the server [8]. Active network control involves also security related tasks like authentication and integrity protection of control traffic [9].

b. Topology Management

After the initial startup phase and when all signaling channels have been established, participating routers will then take the initiative of reporting their capacities to the server by sending Link State Advertisement (LSA) messages. The first LSA sent by a router typically describes the physical characteristics of each of its interfaces, including the IPv6 address and associated link bandwidth. As the server processes LSA messages from routers, it creates an image of the network by aggregating all pairs of connected interfaces into point-to-point links and discovering all possible path combinations from these links. During normal operation, the topology may dynamically change as interfaces are added or removed through LSA messages.

c. QoS Resource Management

Resource management is the main and most work-intensive function of a SAAM server. It refers to those server actions directly related with maintaining the performance status of all links in its region, the resource reservation and user admission control.

B. THE PATH INFORMATION BASE

As the central repository of information about all paths connecting pairs of routers in the SAAM region, the Path Information Base (PIB) is the core component of the SAAM server and is one of the most important modules of the SAAM architecture. The basic design details of PIB have been documented in [2]. The PIB exchanges information with the network by means of four SAAM specific messages. Two of these messages, the link status advertisement (LSA) and the flow request messages are inbound messages and forwarded by the server agent to the PIB module. In response to each flow request message of a client application, the PIB sends a flow response message to the client and under some conditions, a routing update message to the relevant routers. Both messages are generated within the PIB module and sent to the hosting server, which in turn forwards them to the destination routers. Figure 4.2 shows the basic input/output of PIB as described above.

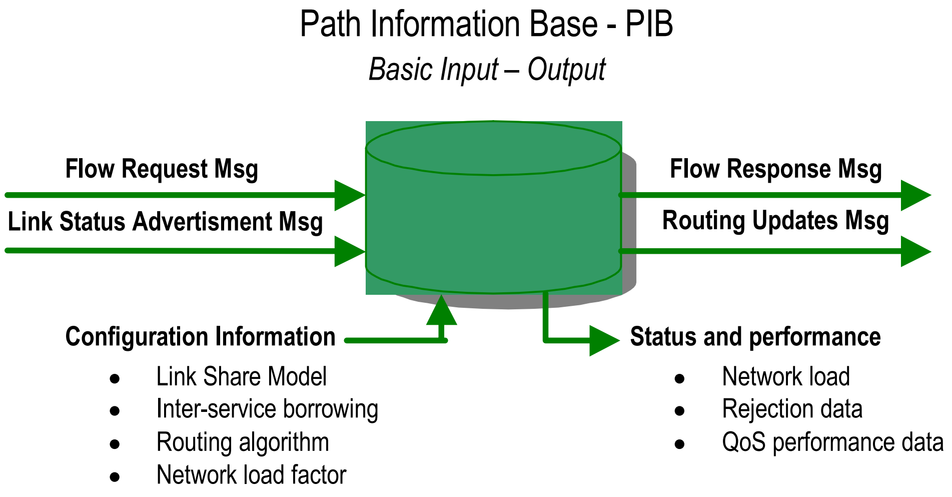


Figure 4.2 The basic input/output channels in the Path Information Base

In addition to the message exchanges during normal network operations, the PIB may accept configuration information during initial startup and, report network status and performance data to an external module. That external module could be part of a network management tool that is capable of sending set or query commands to the PIB, either remotely through especial SAAM messages or locally by conventional console access. An example of such commands would be to turn inter-service borrowing on or off. The development of such functionality is not part of this study and is left for future work.

Figure 4.3 shows the different functional blocks within PIB and the interactions among them. Those functional modules may are organized about two different functional goals: topology management and resource management.

1. Topology Management

Topology management refers to the ability of PIB to discover the topology of the SAAM region and maintain status information about all links. As shown in the top block of figure 4.3, it is achieved by processing LSA messages that the PIB receives from supported routers. Each LSA contains smaller chucks of information called Interface State Advertisement (ISA), each one associated with one of the interfaces of the reporting router. A single ISA may be of typeADD, REMOVE or UPDATE, in order for the PIB to respectively add a new interface, remove an existing interface or update the interface status. The interface status data maintained by PIB are those that are relevant for providing QoS guarantees as defined in SAAM, i.e. the physical link bandwidth, and the link bandwidth utilization, average queuing delay and packet loss rate per service level as observed and advertised by the hosting router.

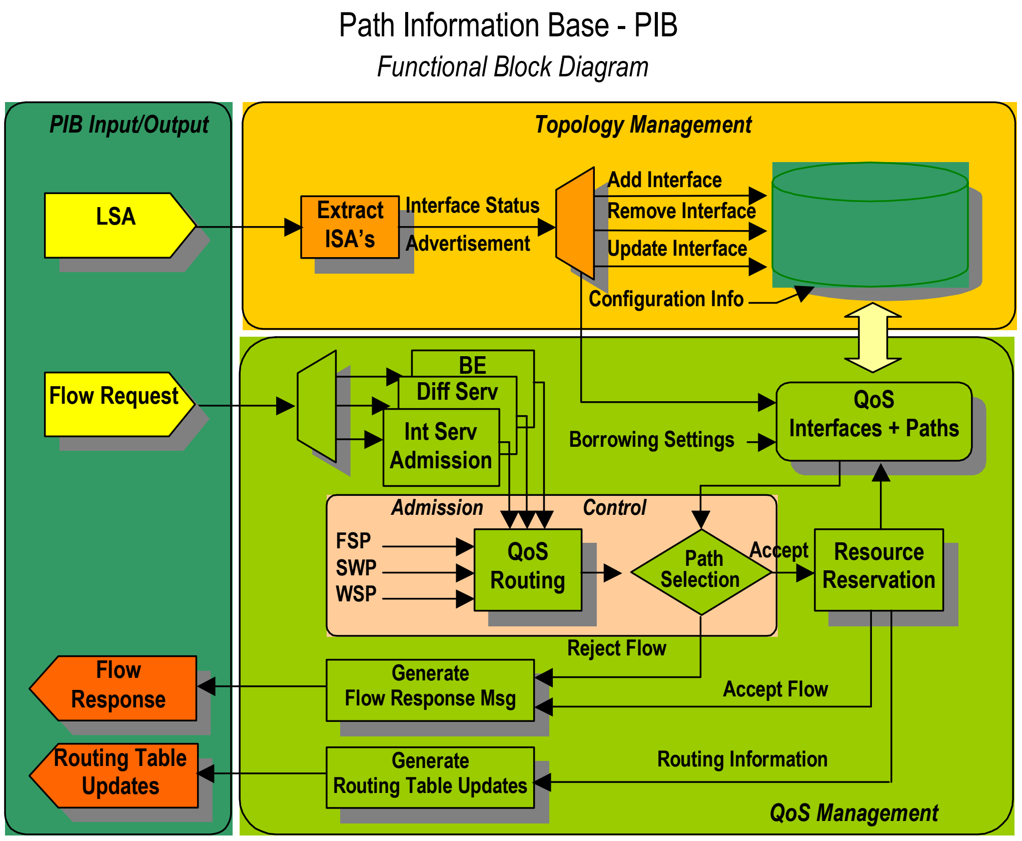


Figure 4.3 The main functional blocks of PIB.

As the PIB adds or removes interfaces, it dynamically regenerates the topology associated with its SAAM region and rearranges the collection of possible paths interconnecting supported routers.

2. QoS Management

When an interface is first advertised, the PIB records its hosting router id, the physical bandwidth, its IPv6 based identification and the information about the neighbor interface, as derived from both the neighbor router id and the number of bits of the subnet mask. The total interface bandwidth is then partitioned into smaller portions and assigned to individual service levels in accordance with the SAAM service model. The current

SAAM prototype supports five different service levels - IntServ, DiffServ, Best Effort, Out of Profile and SAAM control channel.

a. *Interface Information in PIB*

As already stated, each interface in PIB maintains information about observed QoS per service level (e.g. observed utilization, delay and loss rate), which are periodically refreshed by LSAs containing ISAs of type UPDATE. Based upon the QoS data and the allocated bandwidth per service level, the PIB also computes and stores the available bandwidth per service level. Since the available bandwidth depends on the utilization level as reported by routers, it should be recalculated every time the utilization level changes for each of the service level.

b. *Path Information in PIB*

As new interfaces are added to PIB, new paths are also created. Each path element in the PIB contains a unique 16-bit integer based path id, the sequence of outbound interfaces the path traverses, and the path's QoS properties in terms of maximum available bandwidth, the bound on end-to-end packet delay and packet loss rate. Each path QoS property can be expressed as a function of the same QoS property of each interface traversed by the path. While the packet delay bound and the loss rate of the path are the summation of the delay bounds and loss rates of the interfaces respectively, the path available bandwidth is the minimum available bandwidth among all interfaces, i.e. the available bandwidth of the bottleneck interface.

c. *Processing a Flow Request*

When processing a new flow request, the first step, which is common to all requests, is to ensure that source and destination nodes are physically connected, i.e., there is a path in PIB connecting the two nodes. If the destination is unreachable from the source, then the client application will be notified via a flow response message. Otherwise, depending upon the type of request, the admission sequence follows a different procedure as described below.

The simplest admission sequence applies for best effort flow requests. In this case, among all paths available from the source to destination, the admission control module selects one of them in accordance with the predefined path selection scheme. For best effort, no resources need to be reserved, however, if the path has not yet been set up, the resource reservation module ensures the routing tables at each router the path traverses are updated before the affirmative flow response is sent back to the requestor.

The admission sequence for IntServ and DiffServ are similar with the exception that DiffServ requires the additional identification of the network customer running the client application and the confirmation of his Service Level Agreement with the network [3]. The specifics of the agreement and thus the design of this admission step depend upon the service model supported. For both IntServ and DiffServ the next step will then be the selection of a path that meets the QoS requirements of the new flow. At this stage, the admission control unit of PIB simply compares the QoS requirements of this new flow with the QoS properties of all feasible paths to determine the most suitable path. The searching strategy is dictated by the selected routing scheme (i.e., the specific criteria for determining the most suitable path) and by the eventual need to perform inter-service borrowing. Once the path is selected, the resource reservation module of PIB then ensures that the PIB state is updated appropriately to reflect the fact a set of interfaces have just set aside a portion of their resources to support the new flow. This update process directly involves changes to the QoS properties of the path to which the flow is admitted and potentially, of other paths that traverse common outbound interfaces. Furthermore, when a flow is admitted through inter-service borrowing, the update process must be duplicated across the two service levels involved. Finally, a flow response message is sent to the client application. The response is either a flow approval containing the designated flow label that the client application must use to mark its packets, or a flow rejection indicating that no path can meet the QoS requirements of the flow request.

d. Path Selection and QoS Routing

Path selection refers to choosing the best path among all feasible paths that are able to support the QoS parameters as specified in the flow request. There are different approaches to QoS routing [3]: Widest-Shortest Path (WSP), Shortest-Widest Path (SWP) and Shortest-Distance Path (SDP). SWP emphasizes on preserving network resources by selecting a shortest path while WSP provides for load balancing by first selecting widest paths. Due to time constraints, the analysis of the most appropriate schema is left for future research. For the purpose of this thesis, the selected approach is a simplified version of the SWP to what we call First Shortest Path (FSP). The current PIB stores paths with the same source and destination nodes in one array sorted in the increasing order of their hop counts. FSP yields the best possible searching time by retrieving the first path in the array that meets the QoS requirements of the flow request. When inter-service borrowing is enabled and regardless of the route selection schema, inter-service borrowing should only be considered after all paths in the array have been evaluated first without considering inter-service borrowing.

C. INTER-SERVICE BORROWING DESIGN

The underlying theory for inter-service borrowing was already discussed in the previous chapter. Inter-service borrowing only has to do with resource management thus all changes made to the SAAM prototype to enable inter-service borrowing were confined to the PIB module. The following sections describe the design specifics of inter-service borrowing.

1. Inter-service Borrowing at the Interface Level

It was previously stated that PIB maintains two types of QoS data per service level: observed QoS as reported by routers, and available bandwidth as calculated within the PIB. Figure 4.4 below illustrates the typical bandwidth partitioning within one of the service levels participating in inter-service borrowing (DiffServ in this example).

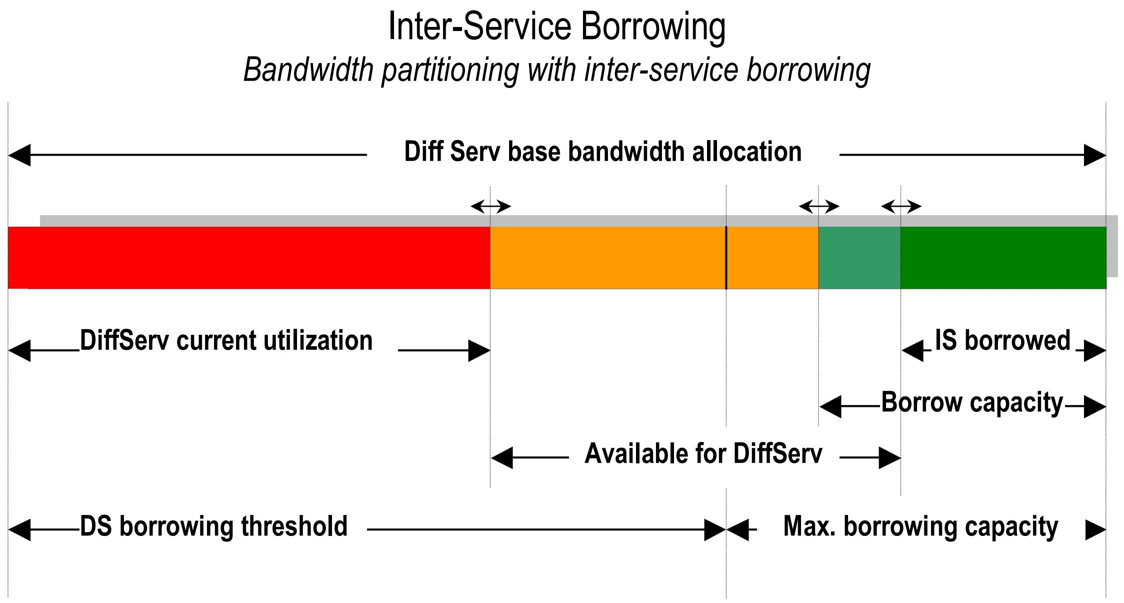


Figure 4.4 Bandwidth partitioning within a single service level that participates in inter-service borrowing (DiffServ in this example).

The whole bar represents the bandwidth base allocation initially assigned to DiffServ. The current DiffServ utilization is shown in red on the left, indicating that the service is under-utilizing its allocated bandwidth, thus suggesting that some capacity could be made available for inter-service borrowing. On the right side of the bar, it is observed that IntServ is already borrowing some capacity from DiffServ (dark green). The capacity made available for IntServ is calculated within PIB as a function of the DiffServ current utilization and the borrowing threshold, i.e., the maximum capacity that might be made available for borrowing. During the admission control sequence, only some of these quantities are of interest. For instance, for admitting a DiffServ flow through this interface, it suffices to know the bandwidth availability of DiffServ. When performing the admission control for IntServ, it only matters to know the borrowing capacity of DiffServ and the amount previously borrowed. With these assumptions and considering the other two QoS metrics of interest (packet delay and loss rate), the following are the minimum set of data associated with a single service level of each interface within PIB:

- Current bandwidth utilization (reported by routers with LSAs)
- Packet queueing delay (reported by routers with LSAs)

- Packet loss rate (reported by routers with LSAs)
- Available bandwidth (calculated in PIB)
- Borrowing capacity (calculated in PIB)

2. Inter-service Borrowing at the Path Level

The path object within PIB contains only those elements of information that are relevant to the admission control function. For the path selection process, it suffices to know the available bandwidth, with and without inter-service borrowing, the packet delay from origin to destination and the packet loss rate for each of the supported service levels. This information is computed from available data stored at each of the interfaces traversed by the path. Figure 4.5 illustrates how the information about three interfaces is used to produce the available bandwidth for a path traversing those interfaces.

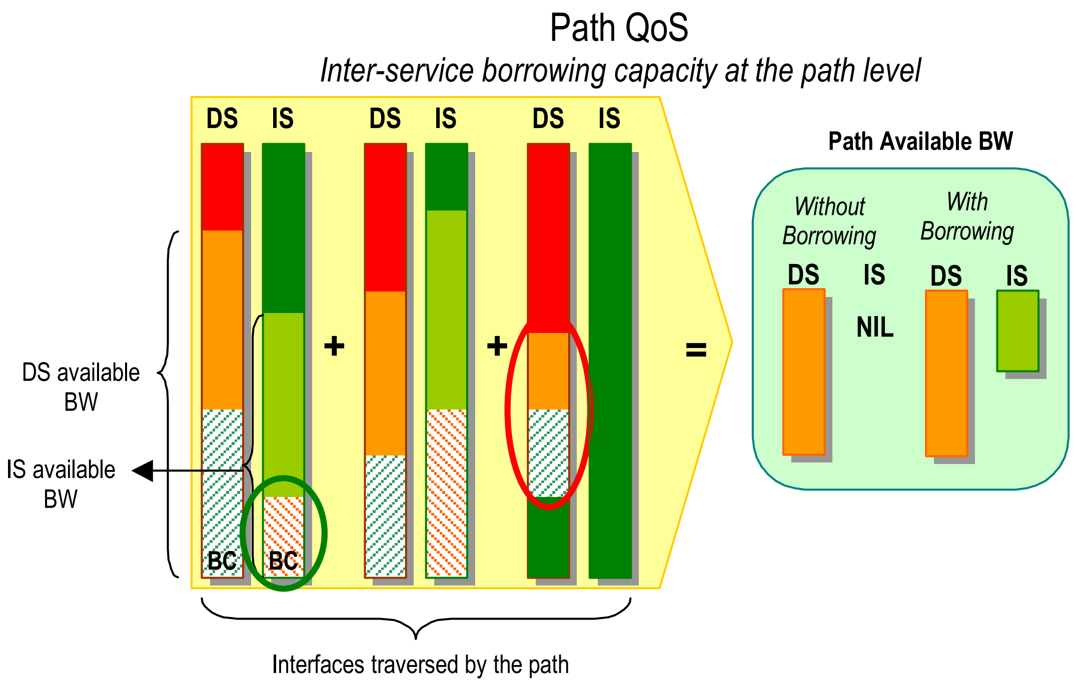


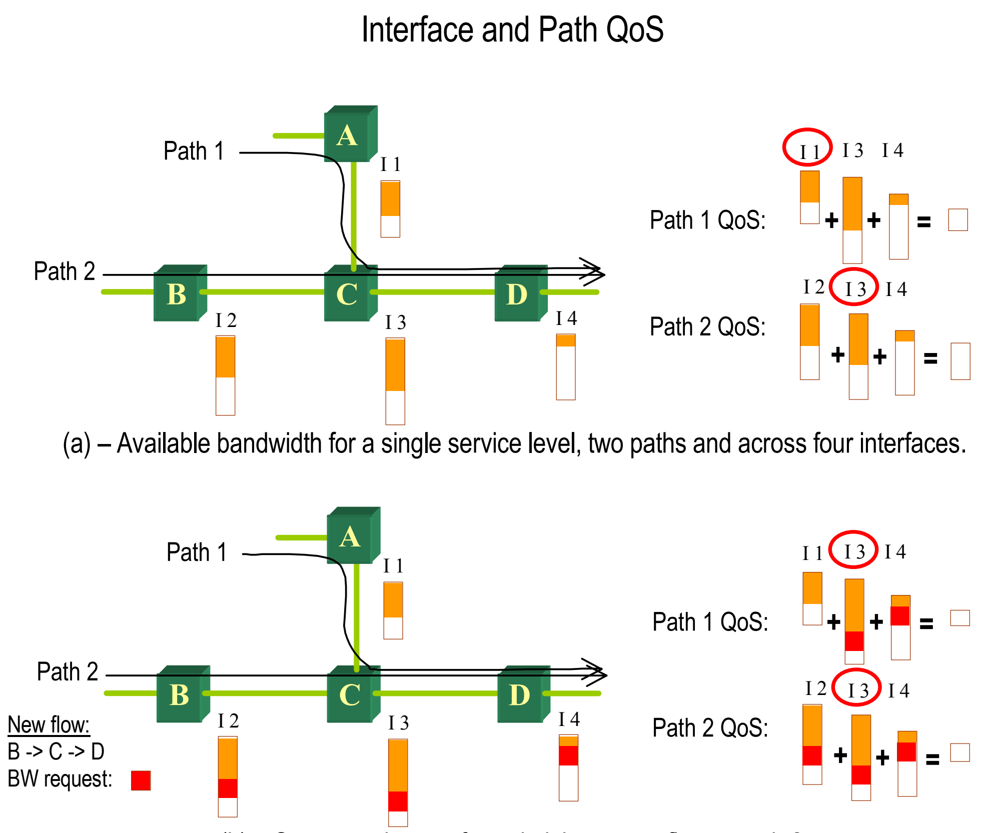
Figure 4.5 Obtaining path QoS information from the interface data

In the example, the third interface shows a high utilization level of IntServ, which is already borrowing some capacity from DiffServ (dark green). Consequently, this path cannot admit any more IntServ flows unless inter-service borrowing is considered. This is shown in the right box, which contains the information associated with the path as calculated from the interface status. Despite not being shown in the figure, in addition to the available bandwidth, the path object contains also the end-to-end packet delay and packet loss rate information.

3. Propagating Interface State Changes

The QoS properties of a path may change due to the admission of a new flow or due to changes of observed performance in one or more of the interfaces it traverses. After processing a LSA of type UPDATE, one or more interfaces in one more service levels may have their observed QoS data modified. In such case, the set of paths that traverse the modified interfaces must also be evaluated and eventually changed to reflect the new state of the interfaces. As already mentioned, the LSA update mechanism allows the PIB to follow a soft state approach in keeping track of admitted flows. Once flows are admitted, there is no need for an explicit flow termination notification and it suffices for PIB to maintain per-interface and per-service level actual aggregated throughput. While changes in service level bandwidth utilization may have no impact in the path information, path delay and loss rate changes will immediately affect the QoS properties of all paths traversing the interface.

A second reason for an interface to change its state is the admission of a new flow. Figure 4.6 shows an example of two paths that share two interfaces in routers C and D. In 4.6(a), it can be observed that interfaces 1 and 3 are the bottleneck for paths 1 and 2 respectively. After admission of a new flow to path 2, all interfaces along the path have their available bandwidth reduced by the amount that is allocated to the new flow. The following step is to propagate changes to those interfaces to all other paths traversing them. In this case, it is observed that the QoS information of path 1 also changes. Moreover, the bottleneck router for path 1 changed from router A to router C (from interface 1 to interface 3).



(a) – Available bandwidth for a single service level, two paths and across four interfaces.

(b) – Same as above, after admitting a new flow to path 2.

Figure 4.6 Propagation of interface state changes after admission of anew flow

The example of figure 4.6 is a rather simple illustration of the propagation of interface state changes. Utilization changes in any of the two services participating in inter-service borrowing are likely to propagate not only across the QoS of paths and interfaces within the service level of the flow request, but also from one service level to the other. For instance, if a new IntServ flow is admitted using capacity borrowed from DiffServ, changes will occur in the available bandwidth of both services. The propagation of such changes is likely to become process-intensive, especially in large SAAM regions and with a high rate of flow request arrivals.

4. Performance Issues

The complexity and the associated performance concern outlined in the previous section, was always present during the redesign of PIB. While the implementation of inter-service borrowing represents an overhead for the PIB computation, some aspects of the previous PIB design have been modified, which are thought to represent an improvement over the overall PIB efficiency. For instance, when processing an ISA, the interface data only changes if the reported data vary above a predefined variation threshold. In any case, eventual changes will only be propagated after all service state advertisement (SSA) information blocks are processed for that interface. Initially, the propagation of changes occurred after processing each of the SSAs, which could happen as many as fifteen times per each ISA, i.e. the number of service levels (five) multiplied by the number metrics (three – utilization, delay and data loss rate). The revised PIB ensures that the propagation of changes happens at most once per ISA processed, which represents a considerable efficiency improvement over the previous version.

During normal network operation, PIB will be processing either link status messages or flow requests. The arrival rate and the size of link status messages are dictated by the predefined LSA cycle duration and the number of router and interfaces supported in the SAAM region. The overhead represented by the arrival of flow requests depends upon the amount of network resources, e.g. link capacity, and other external factors like average flow request rate and flow duration. While maintaining a short LSA cycle is desirable for an increased accuracy of PIB to better respond to new flow demands, it also represents an increase in processing overhead. Such overhead may be excessive in presence of a high rate of flow requests. Long LSA cycles however, shall be avoided especially is the network load changes very rapidly. Because of time limitations, choosing optimum values for those parameters is outside the scope of this study and is left for future evaluation.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SAAM RESOURCE MANAGEMENT IMPLEMENTATION

The SAAM prototype has been developed over the past three years by several students contributing to the SAAM project with their thesis efforts. As part of this thesis study, the new resource management concept as described in the previous chapters was fully implemented in Java and integrated into the Java based SAAM prototype. The implementation and integration details will be described in this chapter. Appendix B contains the SAAM source code that was either created or modified to incorporate the new functionality. With the embodiment of this functionality, the SAAM prototype now has a greatly improved resource management capability.

A. OVERVIEW

As described in chapter IV, the Path Information Based (PIB) is the core element of a SAAM server. The PIB module is self-sufficient, containing both network data and operations that are responsible for the SAAM server behavior. SAAM resource management behavior is therefore fully provided by PIB. Consequently, the PIB was the only module of SAAM that was modified to incorporate the new resource management mechanism detailed in this thesis. The PIB is implemented by the `BasePIB` java class, which in turn contains several inner classes. Table 5.1 contains the complete listing of all those classes with a brief description about the purpose of each one. The detailed description of each of the `BasePIB` data members is fully documented in [2]. Thus, the focus of this chapter is on the parts of PIB that were created or modified as part of this thesis.

The implementation of inter-service borrowing required an extension of the existing data structures that supported SAAM and a redesign of the PIB behavior implementation. Most of the changes directly related with data objects, such as the `InterfaceInfo` and `Path` objects, were accomplished by incorporating additional data members. This was required in order for those objects to keep track of inter-service borrowing capacity, and path available bandwidth. The redesign of the PIB behavior was achieved through a number of changes to existing `BasePIB` methods and additions of several new methods. In addition, the path selection mechanism was tightly integrated into the existing `BasePIB` code. A new class named `RoutingAlgorithm` was created to encapsulate the path selection behavior, making it modular and ready for further development with regard to the route selection process.

Class Name	Description
Main PIB class:	
*saam.server.**BasePIB***	Main class that implements the PIB
PIB inner classes:	
*Saam.server.BasePIB.**InterfaceInfo***	For the interface object.
*saam.server.BasePIB.**ObsQoS***	For the interface observed QoS.
*saam.server.BasePIB.**PathQoS***	For the path QoS information.
*saam.server.BasePIB.**Path***	For the path objects.
*saam.server.BasePIB.**aPIIndex***	The collection of paths in PIB.
*saam.server.BasePIB.**FlowQoS***	For the QoS flow properties.
*saam.server.BasePIB.**RoutingAlgorithm***	For the path selection scheme.

Table 5.1 List of PIB classes

The following sections detail all the changes mentioned above, starting with all the changes made to the inner classes of *BasePIB,* followed by modifications applied to the `BasePIB` methods, and finally, the implementation of the new `RoutingAlgorithm` class.

B. IMPLEMENTATION DETAILS

This section briefly describes `BasePIB` and each of its inner classes. Each of the sub-sections describes a single class, starting with a general description of the class, and followed by a table of the class's data members and its most important methods.

1. BasePIB Support Classes

Several support classes have been implemented as inner classes of the main `BasePIB` class. This approach can be justified because their functions are only relevant for the PIB itself, which is therefore self-contained within the `BasePIB` class. The following sections briefly describe each of those classes.

a. InterfaceInfo Class

The `InterfaceInfo` class defines the object that contains all key information required by PIB to describe a single SAAM router interface. This object contains static and dynamic information about the interface. The static information is related with the network configuration and is loaded upon initial interface advertisement (ISA-ADD). The dynamic information changes very often upon processing LSA or flow request messages. `InterfaceInfo` data members are listed in the table below.

Name	Type	Description
aObjObjectQoS	*ObsQoS[]*	Array of *ObsQoS* objects indexed by the service level. Maintain status information about observed QoS.
bSubnetMask	*byte*	The number of bits of the subnet mask.
htPathIDs	*Hashtable*	Table with all path paths traversing this interface (outbound direction).
iNodeID	*Integer*	The node id of the hosting router.
iServiceLevelAvailableBandwidth	*int[]*	Available bandwidth per service level.
iServiceLevelBorrowingCapacity	*int[]*	Borrowing capacity per service level.
ITotalBandwidth	*int*	Physical interface

Name	Type	Description
		bandwidth.
iUnallocatedBandwdith	*int*	Absolute bandwidth not allocated to any service level.

Table 5.2 Data members of `InterfaceInfo` class

The table below lists some of the methods provided by the `InterfaceInfo` class.

Name	Return	Description
getServiceLevelAvailableBandwidth()	*int[]*	Retrieves the available bandwidth for all of the service levels.
getServiceLevelBorrowingCapacity(byte)	*int*	Retrieves the borrowing capacity for the given service level.
getUnallocatedBandwidth()	*int*	Gets the unallocated bandwidth.
resetQoS()	*void*	Resets the QoS array.
setServiceLevelBorrowingCapacity(int)	*void*	Sets the borrowing capacity for a given service level.
setServiceLevelUnallocatedBandwidth(int)	*void*	Sets the unallocated bandwidth.

Table 5.3 Methods of `InterfaceInfo` class

b. ObsQoS Class

The `ObsQoS` class defines an object that is associated with a single service level at each interface. This object contains observed QoS information for that service level as reported by routers through LSA messages. The table below lists `ObsQoS` data members.

Name	Type	Description
iDelay	*short*	The data delay observed at this interface for a given service level.
iLossRate	*short*	The data loss rate at this interface for a given service level.
iUtilization	*short*	The bandwidth utilization of a given service level, as a scaled percentage of the total interface bandwidth.

Table 5.4 Data members of `ObsQoS` class

c. PathQoS Class

The `PathQoS` class defines an object that is associated with a single path. This object contains relevant end-to-end QoS and resource availability information for a particular service level. The table below lists `PathQoS` data members.

Name	Type	Description
PathAvBW	*int*	The path available bandwidth for a given service level.
pathAvBWwBorrowing	*int*	The path available bandwidth for a single service level, possible incremented by borrowed bandwidth.
PathDelay	*short*	The total path delay for a given service level, i.e. current delay experienced by traffic along the path.
pathLossRate	*short*	The current total data loss rate for a given service level, experienced by traffic along the path.

Table 5.5 Data members of `PathQoS` class

Method Name	Return Type	Description
getAvailableBandwidth()	*int*	Retrieves path available bandwidth for the service level.

Method Name	Return Type	Description
getAvailableBandiwdthIncludingBorrowing()	*int*	Retrieves path available bandwidth for the service level and considering borrowing.
setAvailableBandwidth (int)	*void*	Sets a new value for available bandwidth.
setAvailableBandwidthIncludingBorrowing(int)	*void*	Sets a new value for available bandwidth including borrowing

Table 5.6 Methods of *PathQoS* class

d. Path Class

The *Path* class defines an object that is associated with a single path. The table below lists data members of the *Path* class.

Name	Type	Description
ahtFlows	*Hashtable[]*	An array of look-up tables for flows assigned to a path. Array is index by service level.
bCreated	*boolean*	Signals whether routing tables have been update at the routers traversed by this path.
iNewFlowID	*int*	For the assignment of flow IDs.
iPathID	*Integer*	The ID of this path.
mirrorPath	*Path*	The mirror path, i.e. the path that traversed the same interfaces but in opposite direction.
objaPIIndex	*aPIIndex*	A cross-reference to the array of path objects.
objPathQoS	*PathQoS[]*	The array of *PathQoS* objects that contain the QoS information of this path, per service level.
vInterfaceSequence	*Vector*	The *Vector* object containing the sequence of interfaces traversed by this path (outbound), listed from destination to source node.

Name	Type	Description
`vNodeSequence`	`short`	A `Vector` object containing the sequence of nodes (routers) the path traverses, listed from destination to source.

Table 5.7 Data members of `Path` class

e. aPIIndex Class

This class is a data member of the `Path` class. It is used to create an index object for quick access to the path information array `aPI`. The index fields are source node, destination node and hop count. Given a path object, this object can be used to obtain the `aPI` element (a vector of paths) that contains the path.

Name	Type	Description
`iDestination`	`Integer`	The last node ID of the path.
`iHopCount`	`int`	The hop count of the path.
`iSource`	`Integer`	The first node ID of the path.

Table 5.8 Data members of `aPPIndex` class

f. FlowQoS Class

The `FlowQoS` class defines an object that characterizes the QoS of a single flow request. The table below lists `FlowQoS` data members.

Name	Type	Description
`requestedBandwidth`	`int`	The bandwidth capacity being requested.
`requestedDelay`	`short`	The requested delay bound.
`requestedLossRate`	`short`	The requested loss rate bound.
`timeStamp`	`long`	When the flow request is made.

Table 5.9 Data members of `FlowQoS` class

g. *RoutingAlgorithm Class*

The `RoutingAlgorithm` class implements a stateless object. Routing algorithm objects behave like function objects, providing only the path selection functionality to PIB, based on the selected routing algorithm. The current version of this class only implements the First-Shortest Path (FSP) algorithm but the class can be easily extended to provide different path selection schemes.

Method Name	Return Type	Description
findPath(src, dest, algorithm)	*Path*	Selects a path from PIB, between src and dest, using the selected algorithm.
findPath(src,dest, qos,sl,borrowing, algorithm)	*Path*	Selects a path from PIB, between src and dest, using the selected algorithm and meeting the required QoS demand with borrowing as option.

Table 5.10 Methods of `RoutingAlgorithm` class

2. BasePIB Class

The `BasePIB` class is the main PIB class. Within the SAAM server, the PIB object is an instance of the `BasePIB` class, extending the underlying organization as inherited from the `PathInformationBase` abstract class. The `BasePIB` comprises several data members organized to create a complex data structure, which ultimately represents an image of the SAAM region, containing both static and dynamic information about the supported SAAM network. As stated before the full detail about the PIB implementation can be found in [2]. For the purpose of this thesis, it is only relevant to cover the parts of PIB that have been modified or created for this thesis or are dimmed to be important for describing the implementation of the new resource management and inter-service borrowing mechanisms. The table below lists the most relevant `BasePIB` data members.

Name	Type	Description
afBASE_ALLOCATION	*float[]*	The fraction of the total interface bandwidth, initially

Name	Type	Description
		allocated to each service level.
afLOAD_FACTOR	*float[]*	The maximum load capacity within each of the service levels.
aPI	*Hashtable[][][]*	A tri-dimensional array of hash tables containing all paths between any nodes for a given hop counts.
BORROWING_PROB_OFFSET	*float*	The borrowing limit as an offset factor, depending upon the selected probability (95% equates to 0.164).
BORROWING_THRESHOLD	*float*	The borrowing threshold.
DISPLAY_FULL_DETAIL	*boolean*	Whether the PIB gui will display full detail.
htInterfaces	*Hashtable*	Collection of *InterfaceInfo* objects, keyed by Interface address string.
htPaths	*Hashtable*	Collection of all *Path* objects, keyed by path id integer object.
routingAlgorithm	*RoutingAlgorithm*	The instance of the Routing Algorithm.

Table 5.11 Data members of *BasePIB* class

The table below lists some of the relevant *BasePIB* methods

Method Name	Return Type	Description
admissionControl_BE (FlowRequest)	*int*	Performs the admission sequence for BE.
admissionControl_DS FlowRequest()	*Int*	Performs the admission sequence for DiffServ.
admissionControl_IS	*int*	Performs the admission

Method Name	Return Type	Description
(FlowRequest)		sequence for IntServ.
BorrowingCapacity (bw, utiliz,sl)	*int*	Calculates the borrowing capacity given the service level bandwidth, current utilization and the service level.
FindPathDelayAndLossRate (Path)	*short[][]*	Discovers path delay and loss rate.
isRouteFeasable(src,dest)	*boolean*	Checks if a path exists between *src* and *dest*.
pathBandwidth(Path)	*int[][]*	Discovers path available bandwidth, with and without borrowing, for all service levels.
pathBandwidth(Path, sl)	*int[]*	Discovers path available bandwidth, with and without borrowing, for the given service level.
ProcessFlowRequest (FlowRequest)	*int*	Processes a new flow request.
ProcessLSA (LinkStatusAdvertisement)	*void*	Processes incoming LSA messages.
RefreshInterfaceBW (InterfaceInfo, bandwidth reduction, svc level)	*void*	Refreshes the interface available bandwidth following the admission of a new flow.
RefreshInterfaceQoS (InterfaceInfo)	*void*	Refreshes the interface available bandwidth after processing an LSA that incurred changes.
refreshPathQoS(Path)	*void*	Refreshes path QoS.
RefreshPathQoS (Path, qos variations)	*void*	Refreshes path QoS based on the provided QoS changes.
resetQoS()	*void*	Resets PIB QoS information .

Method Name	Return Type	Description
setInterserviceBorrowing(Boolean)	*void*	Sets the state of inter-service borrowing.
setupPath(Path, int)	*void*	Sets up a path, by sending the routing table update messages to routers.
updateAvailableBandwidth(Path, bandwidth, svc level)	*void*	Updates the available bandwidth of a path after a flow admission.
updateInterface(InterfaceSA)	*void*	Processes ISAs for updating an interface with observed QoS

Table 5.12 Methods of *BasePIB* class

C. MAJOR PIB MODIFICATIONS

The main functions of PIB were outlined in chapter IV. The previous section briefly listed main components of the PIB implementation. In this section, the PIB operation is described in terms of the two main PIB functions, i.e., processing LSA and Flow request messages.

1. LSA Message Processing

Whenever an LSA is to be processed by PIB, the parent object of PIB (Server) invokes *processLSA()*. This method first checks if it the LSA is from a new node. In that case, the new node is created and added to PIB. In either case, the LSA is stripped into one or more ISAs. Finally, depending upon the type of ISA, *updateInterface(), removeInterface()* or *addInterface()* methods is invoked once per ISA.

a. addInterface()

When an interface is first advertised by a router, the respective *InterfaceInfo* object is created and stored in PIB. The *InterfaceInfo* constructor initializes interface data and then calls the *refreshInterfaceQoS()* method.

b. *updateInterface()*

This method is invoked from `processLSA()`, to process a single ISA-UPDATE. An ISA may in turn contain one or more SSAs for that interface. Each metric value of each SSA is extracted and the respective variable in the interface QoS array updated if the changes is above a predefined threshold. The two arrays `deltaLossRate` and `deltaDelay` keep track of all changes introduced while processing all SSAs in this ISA. After all SSAs are processed, then if the observed QoS of this interface effectively has changed, `refreshInterfaceQoS()` is called. Finally, when the QoS properties of the interface has changed, for every path traversing this interface, the `refreshPathQoS()` is called, providing as arguments the `deltaDelay` and `deltaLossRate` arrays.

c. *refreshInterfaceQoS()*

This method may be invoked by the `InterfaceInfo` constructor during initial interface object instantiation or from the `updateInterface()` method, after processing all SSAs of a single ISA. In either case, it first calculates the unallocated bandwidth from the current service level utilizations and total interface bandwidth. For each service level, it calculates the available bandwidth, service level base allocation and borrowing capacity, all as a function of current utilization and base allocation. For this last purpose, the method `borrowingCapacity()` is invoked.

d. *refreshPathQoS()*

The purpose of this method is to refresh the path QoS after changing the QoS properties of one outbound interface along that path. When invoked from the `updateInterface()` method, the provided `deltaDelay` and `deltaLossRate` information allows rapidly setting of the new path delay and loss rate values. However, for determining the path available bandwidth, it is still required to visit all interfaces traversed by the path. This is achieved by invoking the method `pathBandwidth()`. An overloaded version of this method is invoked by any of the four path constructors, for the initialization of the path QoS. In this case, the path delay and data loss rate is obtained by invoking the method `findPathDelayAndLossRate()`.

e. borrowingCapacity()

Given the current utilization, the base allocation and with the predefined offset factor and borrowing threshold, this method calculates the borrowing capacity of a service level.

f. pathBandwidth()

The method traverses all interfaces of a given path to discover the minimum available bandwidth per service level. For each service level, two values are obtained: available bandwidth with and without inter-service borrowing.

g. findPathDelayAndLossRate()

This method traverses all interfaces of a given path. For each of the service levels, it calculates the path delay and data loss rate across the entire path.

2. Flow Request Message Processing

A Flow Request message is passed to PIB by the Server object invoking the method *`processFlowRequest()`*. This is the starting point for the admission control sequence, which follows different paths depending upon the service level of the flow request. The following is the description of all the methods that are directly involved in this admission control sequence.

a. processFlowRequest()

This method simply receives the Flow Request and then, depending upon the service level of the request, invokes the respective admission control method.

b. admissionControl_BE()

The admission sequence for a BE is the simplest. The information about source and destination routers is first extracted from the message. Then the routing algorithm object is called to select the path and finally the flow response is generated with the flow label information and sent back to the source router

c. admissionControl_IS()

The admission sequence for IntServ starts by first extracting the request information including the QoS parameters. The method *`isRouteFeasable()`* is called to verify if a valid path exists, connecting source and destination routers. If that path exists then the routing algorithm object is invoked to select the best path according to the requested QoS parameters and the selected routing algorithm. Then if it succeeds, network resources will be allocated by calling the method `updateAvailableBandwidht()`. Finally, the new flow is added to the selected path, a Flow Response message is generated and sent to the client application and, if necessary, the routing table updates for the path are sent to the routers across the path.

d. admissionControl_DS()

The admission sequence for a DiffServ flow request differs from the IntServ sequence only in the fact that requesting users need to be identified and their QoS parameters are extracted from the pre-established service level agreement with those users.

e. isRouteFeasable()

This method simply searches the array of path information (`aPI`) for ensuring that there exists at least one physical path between source and destination.

f. updateAvailableBandwidth()

This method performs the resource reservation step in the admission sequence. The arguments are the path, the granted bandwidth and the target service level. Adding a flow to a path reduces the available bandwidth of all interfaces traversed by that path. This method ensures that all of those interfaces are visited and for each one, the method `refreshInterfaceBW()` called to perform the respective interface QoS update.

g. refreshInterfaceBW()

This method is invoked once for every interface traversed by a given path, after admission of new DiffServ or IntServ flow. The available bandwidth for the target service level is reduced by the same amount granted to the new flow. However, the refresh algorithm needs to determine in which stage the bandwidth allocation is at, i.e. direct, dynamic or inter-service borrowing. This is relevant to calculate the new unallocated bandwidth and borrowing capacity. Eventual changes in the unallocated bandwidth are also reflected in the available bandwidth for the other service level. For instance, if the admission of a new IntServ flow decreases the unallocated bandwidth, final available bandwidth for both IntServ and DiffServ need to be updated. Finally, changes in the available bandwidth of a single interface propagate to paths traversing that interface. Consequently, all paths traversing affected interfaces need to be evaluated for eventual path QoS changes. For that purpose, the method `pathBandwidth()` is called. For efficiency reasons, this last path update phase is not done immediately after a single interface change is introduced. Instead, as interfaces are updated, the refresh algorithm keeps track of the affected paths. At last, when all affected interfaces have been updated, each of those paths is evaluated. The efficiency gain is more evident when more than one interface is traversed by the same path.

D. CONTROLLING PIB BEHAVIOR

It is desirable to control the behavior of PIB as it interacts with the other network players. Some of that control may be static and is therefore hard-coded and other can be used by other SAAM modules, like a future network management console. The following table summarizes those parameters of PIB that may change its configuration and the way it operates.

Name	Description
`final boolean TESTING_MODE`	Defaults to false. When set to true, a PIB Tester class is instantiated. The PIB tester has its own gui and can be used to inject flow requests and LSA messages in PIB as well as other interactions with PIB. Chapter VI covers this tester in detail.
`final boolean DISPLAY_FULL_DETAIL`	Defaults to false. When set to true, the PIB gui in the SAAM demo station displays more detailed information about PIB operation. This is ideally suited for PIB diagnosis or to monitor the execution flow within PIB.
`final boolean INTERSERVICE_BORROWING_DEFAULT`	The default state for inter-service borrowing. It defaults to true, which means it is enabled

Name	Description
	means it is enabled.
`final float BORROWING_THRESHOLD`	The borrowing threshold is the level above which base allocation bandwidth can be made available for borrowing. The default value is 0.6, i.e. 60% of the base allocation.
`final float BORROWING_PROBABILITY_OFFSET`	This offset factor is derived from the selected probability for inter-service borrowing, as derived from equation 3.11 in chapter III. Default value is 0.164 as given by the probability of 95%.
`final float afBASE_ALLOCATION[]`	This array defines the base bandwidth allocation for all service levels supported. The share of each service level should be so that the summation is less than or equal to the unit value.
`final float afLOAD_FACTOR[]`	This array defines the maximum load to be supported for each of service levels.
`setInterseriveBorrowing()`	Enables to dynamically enable or disable inter-service borrowing.

Table 5.13 PIB configuration elements

VI. PIB TEST AND PERFORMANCE ANALYSIS

The performance evaluation of the proposed resource management technique is a key aspect for this research study. Some of the concepts developed during the previous chapters have a statistical base. A few assumptions require validation. This chapter describes the approach used for testing and evaluating the new PIB, the design and implementation details of the PIB tester module, the specific experiments carried out, and finally, the analysis of the data collected.

A. APPROACH

After completing the implementation, the code was checked for correctness. Every single section was checked for correct logic implementation in accordance with the resource management algorithm. The execution flow and input/output of the different methods was ensured to be as expected. Once the initial check was complete, a functional check of the whole PIB was conducted. The two main functions being checked were the processing of LSA messages and Flow Request messages. LSA generation is not yet completely functional. Current SAAM prototype produces a limited number of LSAs during the initial configuration cycles. Flow generator agents may be used to generate flow request messages. While this approach was acceptable to check the general correctness of the implementation, it was far less than what is required to test the whole resource management algorithm under normal network operation.

Accurate LSA reporting is essential for PIB to maintain coherent status information about the network. However, the link state monitor in the router prototype is not fully implemented, which causes incorrect link status information to arrive to PIB. Another apparent limitation of the current SAAM prototype has to do with the process of generating flow requests and network traffic. While the current system for launching flow generator agents is adequate for a small number of flows, it does not scale well to several hundreds of flows. Without generation of precise LSA messages, it is not possible to test complete PIB operation. Moreover, testing the new resource management capability of PIB requires complete control over a large number of flows.

B. TEST DESIGN

The performance study carried out in this thesis followed a different approach. The SAAM auto-configuration process is run for only one cycle, at the end of which the target PIB is created based on the LSA messages from LSA monitors of all routers. Afterwards, a computer simulation program called PibTester is used to produce all flow request and LSA messages, circumventing the shortcomings of current traffic generator agents and LSA monitors. The theoretical foundations for this evaluation can be found in [11]. The following sub-sections describe the different steps undertaken.

1. System Definition and Testing Goals

As mentioned before, the new resource management scheme was fully implemented within the PIB. The key component of SAAM that is under study is therefore the PIB of a SAAM server. For that purpose, the test is confined to interactions with the *BasePIB* class, including all of its internal support classes. However, the full SAAM prototype should be up and running as normal, reflecting the normal operating environment. The goals of this test will be to (1) evaluate the correctness of PIB resource management operation under normal load conditions; (2) evaluate the impact of inter-service borrowing in terms of efficient management of network resources. In order to attain these goals, the network must work under extreme load conditions for some service, i.e., at the point where inter-service borrowing is required. Additionally, it should suffice to consider the two service levels with dynamic bandwidth allocation, i.e., IntServ and DiffServ.

2. Defining Services

The system service is identified as network resources allocated to network users. In this study, the focus is on the amount of link capacity the network resource management system is able to allocate to users.

3. Selecting Metrics

In the beginning, the test is limited to validating correct operation of PIB. As Flow Request messages arrive, the PIB should be able to complete the admission procedure and allocate resources accordingly. The accurate LSA generation based on traffic of active flows is a major requirement for the testing. After processing these LSA messages, the PIB should accurately refresh its state to reflect new LSA information. The ability to monitor the PIB state in general and the interface and path status information in particular plays a major role in testing the PIB.

In the second phase of testing, we are interested in the number of flows arriving at PIB versus the number of flows that are rejected, in different service levels. This information should be cross-referenced with the number of active flows, from which it is then possible to calculate the expected link load for each of the service levels.

4. Listing Parameters

Several parameters have been identified to affect the resource management performance. These parameters can be divided into system and workload parameters as follows:

a. Systems Parameters

- LSA generation period;
- Borrowing threshold;
- Borrowing probability;
- Service level Base Allocation;

b. Workload Parameters

- Number of active flows;
- Flow duration / requested bandwidth (distribution);
- Inter-arrival time of flow requests;

5. Factors Under Test

The key factor chosen for the purpose of this test is inter-service borrowing. The focus will be on the response of the PIB resource management with inter-service borrowing enabled as opposed to inter-service borrowing disabled. Additionally, different borrowing threshold values will be used to evaluate the impact on flow rejection.

6. Evaluation Technique

Since the PIB is fully prototyped, it is possible to perform a complete functional test of PIB. The measurement technique will be used to collect performance data from PIB. Simulation will be used to generate flow requests and to LSA messages.

7. Workload

The workload for this test consists of a number of flow requests continuously arriving at PIB, following a predefined network load profile. At the same time, LSA generation ensures accurate link state information is reported to PIB. These LSA messages should emulate the tasks of routers advertising link utilization based on network traffic generated by active flows.

8. Designing the Experiments

Two different topologies will be used for testing as described below.

a. Network A

This is the simplest topology and as depicted in figure 6.1, the network will be made of three nodes: one SAAM server and two SAAM routers. The goal is to test PIB resource management across a single link between two routers. For that purpose, the simulation will generate flow requests for traffic going from A to B. The objective is to obtain utilization and flow rejection data about interface 2..1 at router A, as the network load increases.

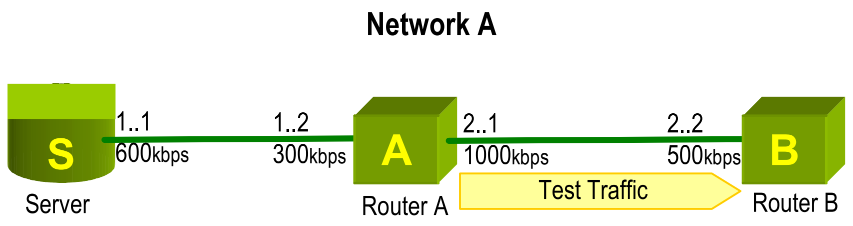


Figure 6.1 Test topology A – 1 server, 2 routers

b. Network B

The focus of network A is to analyze the resource management working at the link level and over a single network link. With network B however, we are interested to observe the behavior of PIB when alternate paths are available and when admission of a single flow indirectly affect other paths. The network is required to be loaded with traffic such that, for any source-destination pair different paths may be selected at different times. Flows are generated from a random source to a random destination among the three routers A, B and C. The selected topology for network B is depicted in figure 6.2.

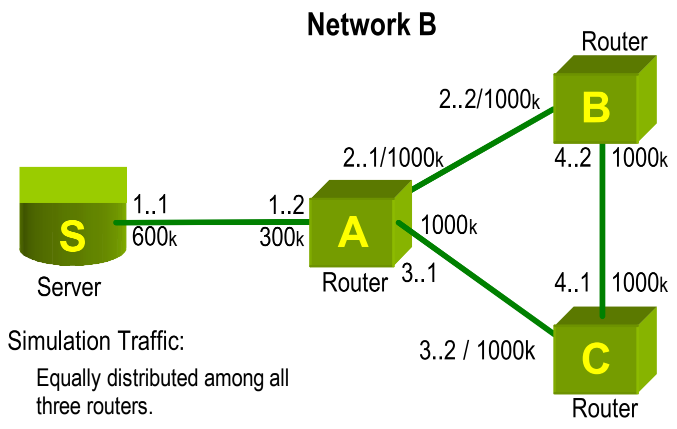


Figure 6.2 Test topology B – 1 server, 3 routers

C. PIB TESTER

1. Tester Requirements

As mentioned in the beginning of this chapter, there were two major factors driving the need for developing a specific PIB tester: accurate LSA generation and effective control over a large number of flow requests. The following basic requirements were identified for that tester:

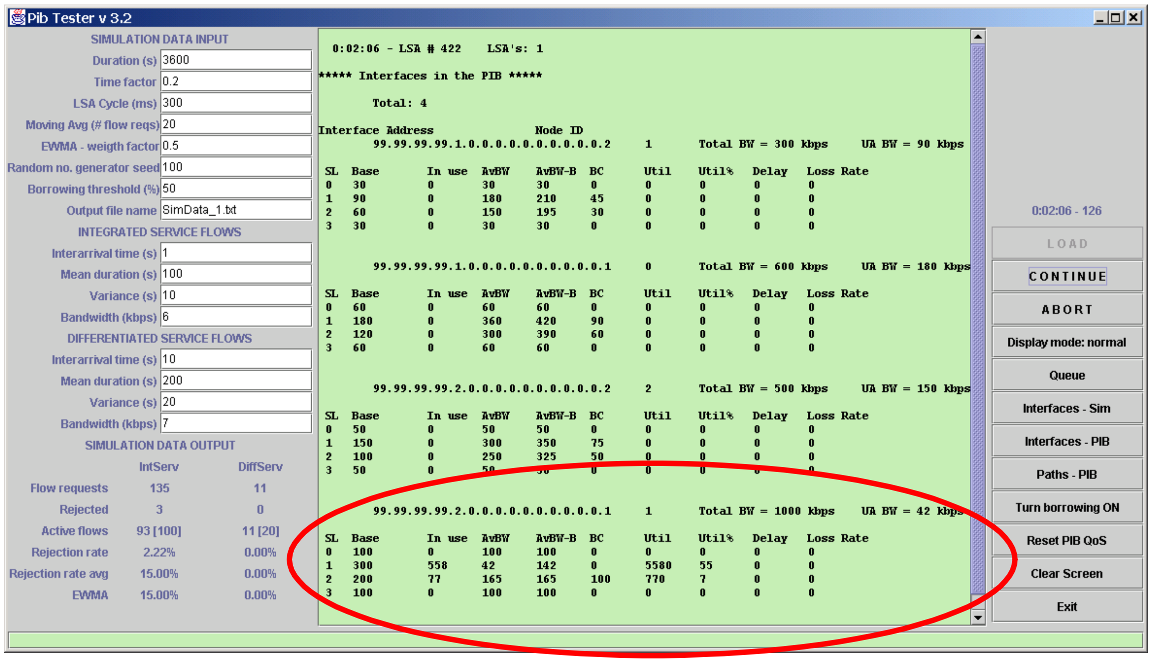
- Generate and send to PIB a stream of flow request messages, in accordance with a predefined link load profile.
- Generate and send periodic LSA messages to PIB. The LSA information should emulate the link status according to the number and characteristics of active flows.
- Store status data every time the system (the PIB and the tester) changes state.
- All testing should be external to PIB.

2. Tester Module

The class `PibTester` was created for the only purpose of testing PIB. `PibTester` is instantiated by `BasePIB` whenever the TESTING_MODE flag in the beginning of `BasePIB` code is set to true. The tester runs in a separated thread and has its own GUI interface. Since the tester is a BasePIB object, it has direct access to BasePIB methods. However, only two methods of BasePIB are used by the tester: `processLSA()` and `processFlowRequests()`. These are exactly the same methods used by the parent class of `BasePIB` (`Server` class) to process incoming LSA and Flow request messages. Synchronization is not required as long as other SAAM players do not send LSA or Flow request messages to the server. The tester automatically waits for initial cycle to complete and no more LSA messages will arrive to PIB from sources other than the tester. Flow generator agents should not be deployed to SAAM routers therefore avoiding flow requests to arrive to PIB. The tester as described above will be the sole player interacting with PIB.

The core element of the `PibTester` is a priority queue, which may contain three types of event objects: flow request, flow termination and LSA. Before a simulation run starts, the priority queue is loaded with flow request and LSA events for the duration of the trial. The number of LSA events is determined by simulation length and the LSA cycle time. The number of flow requests for each of the two services being considered (IntServ and DiffServ) is determined by the respective flow arrival distribution. When a simulation run starts, the simulation time is used to determined when events are dequeued. Every time a flow request event is dequeued, a Flow Request message is sent to PIB. As flows are accepted, the tester updates its own database of interfaces to keep track of active flows and interface utilization. Additionally, a flow termination event is enqueued, with an event time that corresponds to end of the newly admitted flow. When flow termination events are processed, the database of interfaces in the tester is updated to reflect the new network resources becoming available. An LSA event triggers one LSA message for every router represented in the database of interfaces within the tester. The current interface utilization is used to report to PIB the emulated interface status. The LSA will therefore reflect the number of virtual active flows using the network.

During a normal simulation run, the tester GUI displays simulation progress, status information as obtained from the Path Information Base, and limited statistical data. Additionally, at every single simulation event, a line of status data is written to an ASCII file for posterior analysis. Figure 6.3 is a screen shot of the tester interface, during a simulation run of network A. The central area of the display shows individual interface data stored in the Path Information Base. For example, the encircled area contains information about interface 2..1. Appendix 4 contains a detailed description of the tester functionality and appendix 5 contains all of its source code.



SIMULATION DATA INPUT

Duration (s)	3600
Time factor	0.2
LSA Cycle (ms)	300
Moving Avg (# flow reqs)	20
EWMA - weigth factor	0.5
Random no. generator seed	100
Borrowing threshold (%)	50
Output file name	SimData_1.txt

INTEGRATED SERVICE FLOWS

Interarrival time (s)	1
Mean duration (s)	100
Variance (s)	10
Bandwidth (kbps)	6

DIFFERENTIATED SERVICE FLOWS

Interarrival time (s)	10
Mean duration (s)	200
Variance (s)	20
Bandwidth (kbps)	7

SIMULATION DATA OUTPUT

	IntServ	DiffServ
Flow requests	135	11
Rejected	3	0
Active flows	93 [100]	11 [20]
Rejection rate	2.22%	0.00%
Rejection rate avg	15.00%	0.00%
EWMA	15.00%	0.00%

```
 0:02:06 - LSA # 422    LSA's: 1

***** Interfaces in the PIB *****

        Total: 4

Interface Address              Node ID
        99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.2     1       Total BW = 300 kbps     UA BW = 90 kbps

 SL  Base        In use  AvBW     AvBW-B  BC       Util     Util%   Delay   Loss Rate
 0   30          0       30       30      0        0        0       0       0
 1   90          0       180      210     45       0        0       0       0
 2   60          0       150      195     30       0        0       0       0
 3   30          0       30       30      0        0        0       0       0


        99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.0.1     0       Total BW = 600 kbps     UA BW = 180 kbps

 SL  Base        In use  AvBW     AvBW-B  BC       Util     Util%   Delay   Loss Rate
 0   60          0       60       60      0        0        0       0       0
 1   180         0       360      420     90       0        0       0       0
 2   120         0       300      390     60       0        0       0       0
 3   60          0       60       60      0        0        0       0       0


        99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.2     2       Total BW = 500 kbps     UA BW = 150 kbps

 SL  Base        In use  AvBW     AvBW-B  BC       Util     Util%   Delay   Loss Rate
 0   50          0       50       50      0        0        0       0       0
 1   150         0       300      350     75       0        0       0       0
 2   100         0       250      325     50       0        0       0       0
 3   50          0       50       50      0        0        0       0       0


        99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.0.1     1       Total BW = 1000 kbps    UA BW = 42 kbps

 SL  Base        In use  AvBW     AvBW-B  BC       Util     Util%   Delay   Loss Rate
 0   100         0       100      100     0        0        0       0       0
 1   300         558     42       142     0        5580     55      0       0
 2   200         77      165      165     100      770      7       0       0
 3   100         0       100      100     0        0        0       0       0
```

Figure 6.3 A snap shot of the PIB tester, during a simulation run.

D. DATA ANALYSIS AND INTERPRETATION

Several simulation runs were conducted. Each run produced a single file of raw data, which was then imported into an Excel spreadsheet. Excel was used to separate the data from each of the three types of events and generate time graphs showing the progress of the relevant variables. At this stage, the initial and final transient periods were easily identified. Since we are only interested on the steady-state performance, data collected over those two periods were deleted. From remaining data, it was then possible to obtain two types of information: a graphic representation of variables plotted against time and calculated average values during that sample interval. The following sections present the data as described.

1. Borrowing Sensitivity Test With Network A

For the purpose of this test, the interface 2..1 of network A shown in Figure 6.1 was subject to five different load conditions. Table 6.1 below shows the interface state at startup, i.e. with no traffic traversing it.

Service Level	Bandwidth			
	Base Allocation	Available	Available with borrowing	Borrowing Capacity
SAAM Control	100	100	100	0
IntServ	300	600	680	120
DiffServ	200	500	620	80
Best Effort	100	100	100	0

Unallocated	300
Total BW	1000

Table 6.1 Interface configuration at startup.

Table 6.2 shows the flow characterization for load A. Different network loads were achieved by changing the flow duration for IntServ, which implicitly affects the projected number of active IntServ flows. Flow requests for each of the DiffServ and IntServ services were generated using a Poisson distribution, and the flow durations modeled with a normal distribution. Table 6.3 summarizes the projected flow requests for the five different loads, with ensuing traffic going from router A to router E.

Parameter	IntServ	DiffServ
Inter-arrival time (sec)	1	10
Duration – mean (sec)	100	200
Duration – sigma (sec)	10	10
Bandwidth (kbps)	6	7

Table 6.2 Characterization of individual flows

For each of the five loads, four different random generation seeds were used. For each seed, the simulation was run twice, first with inter-service borrowing enabled and then with the borrowing disabled. Therefore, there were a total of 8 simulation runs per load. Appendix D contains summary information for each of the described test conditions.

Load	Projected Data			
	Active Flows		Bandwidth Request (Kbps)	
	IntServ	DiffServ	IntServ	DiffServ
A	100	20	600	140
B	104	20	624	140
C	108	20	648	140
D	112	20	672	140
E	116	20	696	140

Table 6.3 Characterization of individual flows

A single simulation run was done over a period of 1000 seconds, which allowed over a 1,2000 flow requests to be processed for both services. Figure 6.4 shows an example of a time-graph plotted from raw data obtained for load A, without inter-service borrowing and with a seed of 100. When all runs were done, the steady-state interval was considered between 250 and 1000 seconds. This was so since in all test cases after 250 seconds service level loads were observed to have reached a steady state.

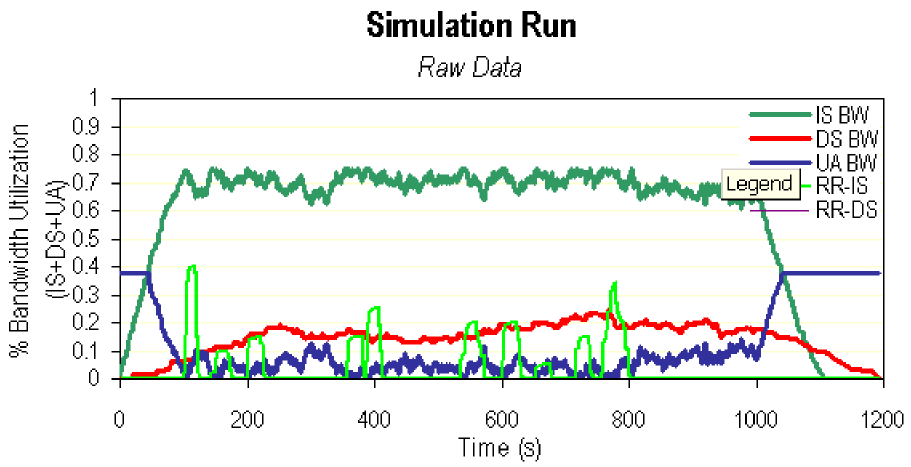
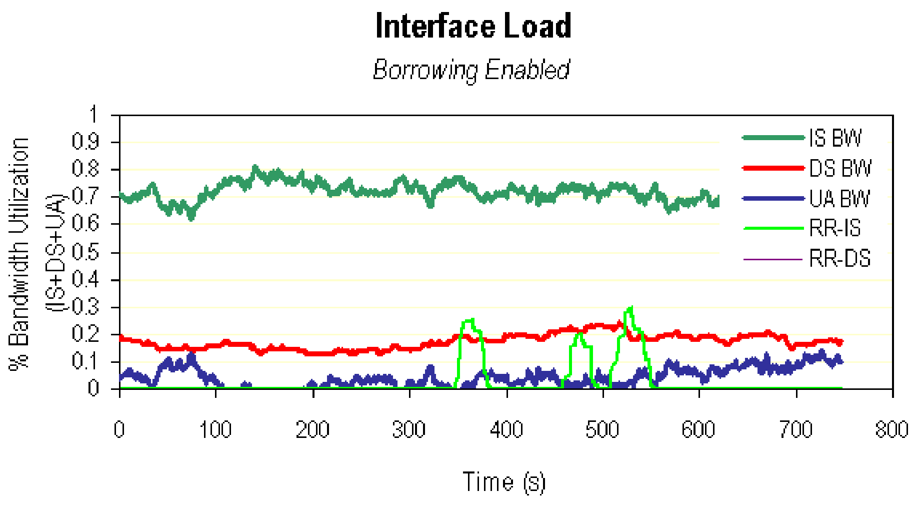


Figure 6.4 Raw data from a single simulation run.

Having defined the steady-state interval for all loads, then the sample respective sample data was extracted from the original data. Figure 6.5 below shows two graphs plotted from data of the same load condition of figure 6.4. The two borrowing states are represented. As can be observed, only the steady-state interval is shown. Each graphs shows the effective interface load per service level (IntServ and DiffServ) plus unallocated bandwidth. Additionally, the rejection rates per service level are also represented.



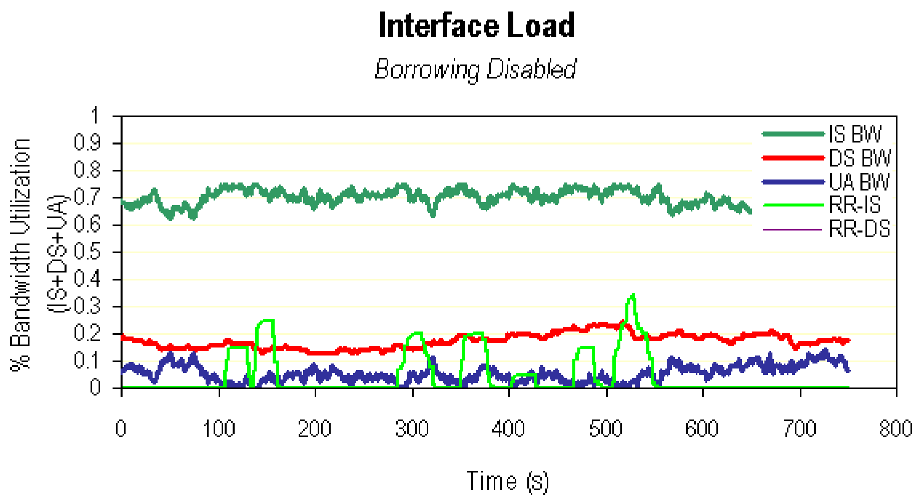


Figure 6.5 Interface 2..1 load during two simulation runs, with and without inter-service borrowing

As previously stated, simulation data details can be found in Appendix D. Graphs in Figures 6.6 and 6.7 summarize the findings. For an increasing load of IntServ and as expected, the flow rejection rate of IntServ also increases. Note that the flow rejection rates were obtained by averaging the results over four runs with different random generator seeds. In all load cases, the impact of inter-service borrowing is extremely significant. It is also noted that the gains of using inter-service borrowing slightly decrease with the increase of the network load which is explained with the saturation of the borrowing capacity made available by the other service.

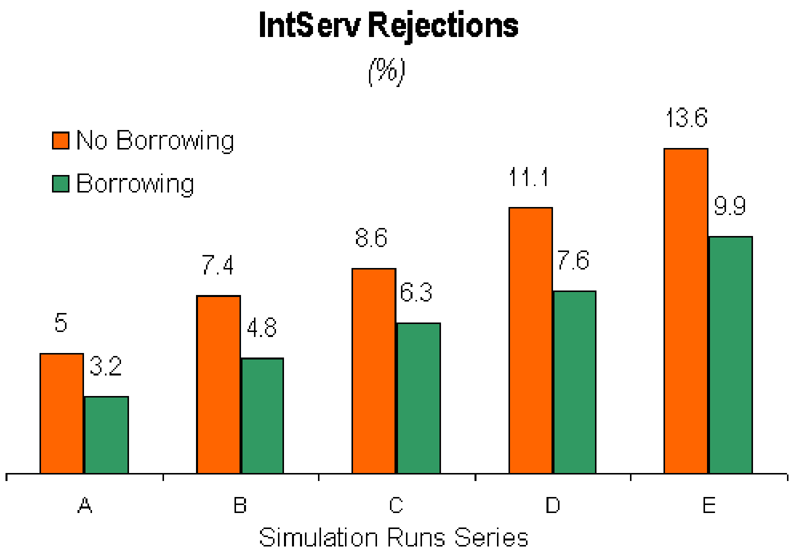


Figure 6.6 Comparative analysis of borrowing versus no borrowing, for increasing network load

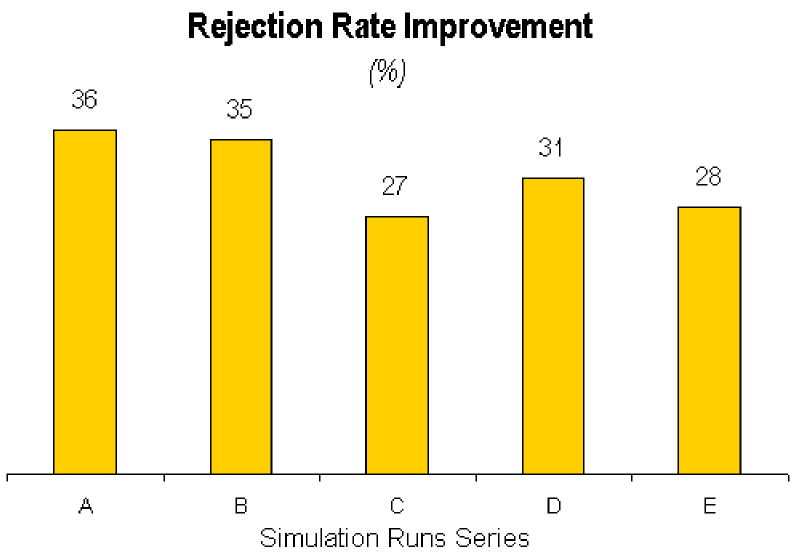


Figure 6.7 Reduction of IntServ flow rejection rate as network load increases

2. Network B Test

For the purpose of this test, the PibTester was adapted to generate flow requests from a random source to a random destination, among all three nodes (A, B and C). Since the test objective was to check the PIB function when more alternate paths are available, there was no need to collect data at the interface level. With the simulation running, the

PibTester GUI, as presented in figure 6.1, offers the option of inspecting the status of individual paths and interfaces at any point during the simulation. This option was used to verify the functionalities of the PIB. The observed results were as expected.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSIONS AND RECOMMENDATIONS

This thesis demonstrated the feasibility of efficient management of network resources while providing support for different classes of QoS traffic. The novel inter-service borrowing mechanism was developed and integrated with SAAM. This new mechanism results in a more dynamic and adaptive link share among supported services. The test and evaluation results show evidence of a significant improvement of the overall network resource utilization. The new resource management concept further strengthens the goal of SAAM to intelligently manage network resources. Some key aspects of this study are covered below.

A. PATH INFORMATION BASE REDESIGN

The implementation of inter-service borrowing in PIB involved changes in every aspect of the PIB internals. Although some new data members were required to be added to existing data structure, some internal algorithms were completely redesigned with efficiency in mind. One of such modifications is related with processing LSA messages and propagating interface updates. Despite the newly added functionality and associated complexity, the revised PIB is considered more robust and more efficient.

B. PIB TEST

The test drive specifically developed for this thesis proved very valuable. The PIB tester was capable of generating not only a continuous stream of flow requests but also the required Link Status Advertisement messages with the adequate periodicity. The friendly and flexible interface of the tester allowed for a large number of testing conditions and network loads to be run against PIB. For the first time, it was possible to stress test all the functional parts of PIB at the same time. Each test run generated large amounts of data. Processing such large quantities of data, validating them and extracting relevant information was at some point overwhelming for the testing platform. However, obtained results were very satisfactory which helps to strengthen the SAAM concept.

C. AREAS FOR FURTHER STUDY

1. Link State Advertisement Cycle

The LSA cycle duration should not be arbitrarily selected. One of the major processing overheads within PIB is the processing of LSAs. If routers advertise the state of their interfaces very frequently and if the SAAM region contains a large number of routers/interfaces, the burden of processing LSAs may impact the performance of PIB. In such cases, flow requests may have to be buffered while waiting for an LSA cycle to complete. However, a short LSA cycle also means that the PIB state is oftener refreshed, thus maintaining a more accurate image of its network. If otherwise the LSA cycle is made too long, PIB state is less accurate which may eventually cause incorrect admission of new flows, leading to buffer overruns at routers. The LSA cycle need not to be constant and could vary depending upon network conditions. Additionally, some kind of prioritization among flow requests and LSA messages arriving to PIB should be implemented. The tuning of the LSA cycle is therefore an important PIB function and should therefore be in the scope of further study of SAAM.

2. Accurate Link State Advertisement

Whatever modifications are made within the Path Information Base, its function is highly dependent on the external two inputs it receives – the Link State Advertisement and the Flow Request messages. Flow requests simply translate the demand of resources from user applications and is already working as expected. However, LSA generation is not yet fully implemented by SAAM routers. The Link Sate Monitor is the module responsible for monitoring interfaces and advertising their state based on observation of real traffic. Current implementation of the Link State Monitor reports zero interface utilization or other inaccurate value regardless of traffic flows. Once these values arrive to PIB in LSA messages, they are subsequently used to update PIB status. Because of the inaccurate state of PIB, the admission control and resource reservation mechanisms of PIB in most cases will perform incorrectly. In order to make the best use of current functionality of PIB it is highly desirable that LSA generation in SAAM be fully implemented.

3. SAAM Network Management

Every component deployed in a network is typically operated either locally through console access or remotely using network management applications. These applications communicate with network components to query their state or to set functional parameters using an application layer protocol such as Simple Network Management Protocol (SNMP). The PIB within a SAAM server has evolved to a stage where some of its behavior can be changed during normal operation. For instance, the inter-service borrowing capacity can be turned on or off during normal operation. An area of potential study would be the design and implementation of an application that was able to generate SNMP traffic destined to specific SAAM modules/agents such as the SAAM server and the PIB, to be able to perform network management functions.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A MILITARY RELEVANCE OF SAAM PROJECT

A. SUMMARY

The vision for future joint war fighting of US military is described in Joint Vision 2020 (JV2020). The concept of network-centric warfare (NCW), first conveyed in the JV2010 and carried forward in JV2020, represents a fundamental shift from the previous platform-centric warfare. Interoperability with external agencies and among forces of the allied nations is a growing necessity as recently proved with the combined NATO operations in the Balkans. Military operations in the current information age are organized around the NCW concept, through which information superiority translates into increased combat power. NCW is enabled by effectively networking sensors, decision makers and shooters to achieve shared awareness, increased speed of command and high levels of self-synchronization.

The NCW environment creates a wide range of network service requirements, only possible to meet through active and adaptive networks. Server and Agent Based Active network Management (SAAM) is one of such networks being prototyped at the Naval Postgraduate School. This document addresses some key enabler technologies of SAAM, which illustrate the importance of SAAM in the context of the NCW environment.

B. DISCUSSION

Joint Vision 2020 builds on the foundation of Joint Vision 2010. Several strategic principles and operational concepts of JV2020 are technically addressed by SAAM.

1. Global Information Grid

JV2020 develops a concept labeled Global Information Grid (GID). The GID requires a network-centric environment that integrates traditional forms of information operations with sophisticated all-source intelligence, surveillance, and reconnaissance in a fully synchronized information operation. SAAM supports dynamic, non-intrusive service deployment with which software agents can be dynamically deployed across the GID and configured to perform various tasks on demand.

The GID will be a globally interconnected, end-to-end set of information capabilities, associated processes, and people to manage and provide information on demand to warfighters, policy makers, and support personnel. The SAAM hierarchical architecture and auto-configuration protocol provide a mechanism for SAAM to scale from single SAAM regions into a global information infrastructure. Additionally, the quality-of-service (QoS) model of SAAM supports guaranteed services, capable of delay guarantees to individual network users, which is an important guarantee for applications that rely on synchronization.

The GID needs to continue functioning when under hostile attacks. The centralized SAAM approach makes SAAM servers a privileged network player possessing the broadest possible view of distributed GID resources. With such visibility over its resources, SAAM servers are able to take a pro-active approach to fault tolerance by creating optimal alternative paths ahead of time. Under malicious aggression, SAAM immediately redirects affected flows to these alternative paths, which happens seamlessly to network users. The survivability of the GID is further enhanced with SAAM's ability of relocating server functionalities to a different physical network node rapidly without significant service degradation. There is no single point of failure.

2. Innovation

JV2020 identifies technical innovation as a vital component of the revolution in future warfare. SAAM is an agent-based network, which means that new functionality can be easily deployed on the fly. New SAAM agents can extend or replace the functions of existing agents. Once the SAAM network infrastructure is deployed, the introduction of network changes, new requirements, or added functionalities can be easily accommodated.

3. Interoperability

Interoperability is the ability to provide services to and from other systems. It is a mandate for the joint force of 2020, especially in terms of communications, and information sharing. Multinational operations like recent NATO operations in the Balkans, once again demonstrated the importance of interoperability. Information systems and equipment that enable a common relevant operational picture must work from shared networks that can be accessed by any authorized participant regardless of the location.

The centralized approach of SAAM to network management provides for a controlled access to network resources. Identification and authentication of network users is supported and it provides the means for implementing secure communication channels. Additionally, SAAM agents can be tailored to provide the bridge between the varying levels of technology of the potential multinational allied nations. Specially configured agents may be deployed to edge nodes that interconnect incompatible system with the shared network. These agents perform all required traffic adaptation, thus supporting interoperability.

4. Communication Command and Control

NCW requires the coexistence of multiple levels of traffic priority for the Communication, Command and Control (C3) channels. The SAAM QoS model enables applications to specify their QoS requirements, including throughput and delay. SAAM resources are allocated hierarchically and the best performing QoS routes can be easily assigned to the highest priority C3 channels, for instance, between front line units and C3 centers.

SAAM implements a better-fit QoS model for battle. With the per-flow management capability, SAAM can establish different priorities for different flows. Different conversations may have different priorities. Battle scenario is constantly changing, and so is supporting network infrastructure and network users. Network resources are limited. Because battle scenarios are frequently in remote and adverse locations, deployment of a network for supporting NCW in those scenarios requires adequate optimization of such limited resources. The intelligent network management approach of SAAM is perfectly suited for those conditions, since allocation of resources is based on both traffic profile and changing network conditions. Whenever QoS performance of some QoS traffic is affected, SAAM network automatically and dynamically adapts to optimize available resources and, equally important, ensures that high priority traffic is favored whenever network conditions degrade.

C. RECOMMENDATION

The work of this thesis greatly contributes for the improvement of the SAAM concept and further extends the potential of SAAM becoming a solution to all major technical problems posed by NCW.

APPENDIX B PIB SOURCE CODE

```
//11Jun2001[PauloSilva] - PIB redesigned to provide for inter-service
                         borrowing.
//                        All code was reviewed and cleaned. Aditionally,
//                        inner class RoutingAlgorithm was added to encapsulate
//                        the routing algorithm behavior.
//28May2001[Xie]        - Rewrite setupPath: vInterfaceSequence only contains
//                        outbound interfaces
//12Mar2001[Colwell]    - Redesign eliminating non-final static data structs
//Dec2000[Xie]          - Reviewed code and cleaned up;
//                        see comments prefixed by [GX]
//Aug2000[Kuo & Gibson] - Created

package saam.server;

import saam.util.*;
import saam.net.*;
import saam.router.Interface;
import saam.message.*;
import saam.server.Server;
import saam.server.diffserv.SLS;

import java.util.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;


/**
 * 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{

  /**
   * Runs PIB under testing mode. A BasePIB tester console (PibTester class)
   * associated with the Path Information Base is started. The tester has its
   * own GUI and allows for direct comunnication with PIB for sending Flow
   * Request and LSA messages or quering PIB state, like interface or path
   * status, toggle inter-service borrowing on/off, etc.
   */
  private final boolean TESTING_MODE = true;

  /**
   * The pibTest is used only in test mode. It will launch a separated gui
   * console.
   */
  private Thread pibTest;

  /**
   * For the level of display detail. If set to true, the pib will display in
   * its gui more detailed information. Should be used PIB diagnosis.
   */
  private final boolean DISPLAY_FULL_DETAIL = false;

  /**
   * The default state of interservice-borrowing capability (true for enable).
   */
```

```
private final boolean INTERSERVICE_BORROWING_DEFAULT = true;

/**
 * The borrowing threshold is a percentage of the a service level base
 * allocation bandwidth, which represents the amount of bandwidth a service
 * will never make available for inter-service borrowing.
 */
protected final float DEFAULT_BORROWING_THRESHOLD = 0.60f;

/**
 * This offset is obtained from the inter-service borrowing algorithm, after
 * applying the quantiles for the normal distribution. This value is used to
 * compute the maximum borrowing capacity of a service level, and is a
 * function of probability of request conflict among service levels.
 * For instance, for the probability of 95%, the offset becomes 0.164.
 */
protected final float BORROWING_PROBABILITY_OFFSET = 0.164f;

/**
 * The routing algorithm module.
 */
private RoutingAlgorithm routingAlgorithm;

/**
 * The current PIB version information.
 */
public final String strPIB_VERSION =
  "----------------------------------------------------------------------\n" +
  " SAAM Network - Path Information Base\n" +
  " Version 2.0b August 2001\n" +
  "----------------------------------------------------------------------\n";

/**
 * The number of different service levels supported
 */
public final int  NUM_OF_SERVICE_LEVELS = 5;

/**
 * SAAM control channel - SAAM service level 0
 */
public static final byte CTRL_CHN    = 0;

/**
 * Integrated Services - SAAM service level 1
 */
public static final byte INT_SERV    = 1;

/**
 * Differentiated Services - SAAM service level 2
 */
public static final byte DIFF_SERV   = 2;

/**
 * Best Effort - SAAM service level 3
 */
public static final byte BEST_EFFORT = 3;

/**
 * Out of Profile -  - SAAM service level 4
 */
public static final byte OUT_PROFILE = 4;

/**
 * Maximum and minimum flow ID numbers (8-bit field)
 */
public final int MAX_FLOW_ID = 255;
public final int MIN_FLOW_ID = 1;
public static int newFlowID = 0;     //[GX]: each path should have own counter

/**
 * This is a magic number used for the Best Effort admission algorithm.
 * For every newly admited, the available bandwdith for Best Services is
 * reduced by this amount. [PS] - the BE admission procedure needs to be
 * revised.
 */
public static int Best_Effort_Allocated_Bandwidth = 50000;

/**
 * The current state of pib interservice borrowing capability.
 */
private boolean isBorrowingEnabled;

/**
 * The current inter-service borrowing threshold.
 */
private double borrowingThreshold;

/**
 * Initial base allocation of interface bandwidth among all service levels.
 * Summation of all shares must be less or equal to 1.0f.
 */
public final float afBASE_ALLOCATION[] = {0.1f, 0.3f, 0.2f, 0.1f, 0.0f};

/**
 * Network load factor for each of the specified service levels. Load factor
 * is used to specify the maximum amount of network resources (bandwidth)
 * that might be in use at any time. The individual service load factor
 * are values between 0 and 1.0.
 */
public final float afLOAD_FACTOR[] = {1.0f, 1.0f, 1.0f, 1.0f, 1.0f};

/**
 * Utilization update threshold. This value multiplied with the current unit
 * of utilization (ServiceSA.UTIL_UNIT) yields the level of utilization
 * variation, above which the utilization report (ISA) will change trigger
 * an interface update.
 */
public final byte thresholdUtilization = 5; //times ServiceSA.UTIL_UNIT (0.01%)

/**
 * Delay update threshold. Delay variation threshold above which, a new delay
 * report (ISA) will actually trigger an interface update.
 */
public final short thresholdDelay      = 5;

/**
 * Loss Rate update threshold. Loss Rate variation threshold above which, a
 * new loss rate report (ISA) will actually trigger an interface update.
 */
public final short thresholdLossRate   = 5;

/**
 * Next path ID. Holds the path ID number to be assigned to application
 * flows. Paths 0 to 64 are reserved for signaling channels.
 */
public  static short  iNextPathID = 65;

/**
 * Next node ID. Holds the value of the next sequential node identification
 * number. Defaults to smallest non-negative integer value upon PIB.
 * initialization
 */
public  static int  iNextNodeID =  0;

/**
 * Maximum number of nodes. Holds the maximum number of nodes allowed
 * within a SAAM region.
 */
public final int MAX_NODE_NUMBER = 30;

/**
 * Holds the maximum number of hops any single path can make.
 */
public final int MAX_HOP_COUNT = 8 + 1;

/**
 * To keep track of the number of LSA messages received
 */
private static int iLsaCounter = 0;

/**
 * The Server instance.
 */
private Server myServer;

/**
 * The GUI display window for PIB.
 */
private SAAMRouterGui display;

/**
 * For number formating purposes used in PIB gui.
 */
private DecimalFormat df;
/**
 * A three-dimensional array of hashtables containing all path IDs between a
 * source and destination router pair for specific hop count.
 */
Hashtable  aPI[][][];

/**
 * A table, keyed by the router's largest IPv6 address, associating the router
 * with a unique node iD for the life of the Path Information Base structure.
 * While the router's ID, as determined by its assigned IPv6 addresses, may
 * change, its assigned Node ID remains constant unless the PIB is reset.
 */
Hashtable   htRouterIDtoNodeID;

/**
 *  A reverse look-up table, keyed by the Integer Node ID, used to identify a
 *  router's IPv6 based name.
 */
Hashtable   htNodeIDtoRouterID;

/**
 * A look-up table, keyed by the router's IPv6 based ID, which contains all
 * the active interfaces for a corresponding router.  The interfaces are
 * themselves contained in 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.
 */
Hashtable   htRouterInterfaceMap;

/**
 * A look-up table, keyed by the interface's IPv6 address byte array, which
 * holds the InterfaceInfo for the corresponding interface.
 */
Hashtable    htInterfaces;

/**
 * A look-up table, keyed by iPathID, which enables rapid access to all paths
 * that have been established. The path objects are store within the table
 * elements.
 */
Hashtable  htPaths;

/**
 *  A look-up table, used for Differentiated Service key is the userID, object
 *  stored in this table is SLS object.
 */
Hashtable  htUserSLSs;

/**
```

```
 * A vector to store the collection of interfaces traversed and affected after
 * a single flow request. Used in conjuntion with the PibTester class for test
 * purposes only (when PIB is test mode).
 */
protected Vector vIFacesTraversed;

/**
 * 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, Destinaytion Node, and Hop Count  This object cross references a
 * path to the aPI element which contains it.
 */
private class aPIIndex
{
  /**
   * The Node ID for the source router
   */
  Integer iSource;

  /**
   * The Node ID of the destination router
   */
  Integer iDestination;

  /**
   * The number of nodes each path takes
   */
  int iHopCount;

  /**
   * Constructs an aPIndex object with the provided parameters.
   * @param iS
   * @param iD
   * @param iHC
   */
  public aPIIndex (Integer iS, Integer iD, int iHC)
  {
    iSource = iS; iDestination = iD; iHopCount = iHC;}

  /**
   * Gets the node ID of the source router
   * @return the node ID of the source router
   */
  public Integer getSource()
  {
    return iSource;
  }

  /**
   * Gets the node ID of the destination router
   * @return the node ID of the destination router
   */
  public Integer getDestination()
  {
    return iDestination;
  }

  /**
   * Gest the hop count
   * @return the hop count
   */
  public int getHopCount()
  {
    return iHopCount;
  }

} // End of aPIIndex class


/**
 * 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
{
  /**
   * The path ID wrapped as an object for use in hashtables.
   */
  Integer  iPathID;

  /**
   * If routing entries have been created in routers to support the path
   */
  boolean  bCreated = false;

  /**
   * Used to flag paths that change as a consequence of a new flow admission,
   * easing the process of QoS refreshing of all interfaces/paths directly
   * and indirectly involved
   */

  /**
   * The mirror path. For quick look-up, it keeps a reference to the mirror,
   * where source and destination are swapped.
   */
  Path mirrorPath;

  /**
   * Cross reference for the path to the array of path information objects
   */
  aPIIndex objaPIIndex;

  /**
   * The node sequence vector. Contains the sequence of nodes the path
   * traverses, from destination to source
   */
  Vector vNodeSequence;

  /**
   * The interface sequence vector. Contains the sequence of outbound
   * interfaces traversed by this path, listed from destination node to
   * source node.
   */
  Vector vInterfaceSequence;

  /**
   * The array of path QoS objects, indexed by the service level index.
   */
  PathQoS objPathQoS[] = new PathQoS[NUM_OF_SERVICE_LEVELS - 1];

  /**
   * For the flow IDs.
   */
  int iNewFlowID = MIN_FLOW_ID - 1;

  /**
   * Look-up table of flows assigned to each service level on a path.  Each
   * hashtable is keyed by the flow ID and pairs the actual flow ID with the
   * key generated by the hash scheme.  The array of hashtables is indexed by
   * the service level.
   */
  Hashtable ahtFlows[] = new Hashtable[NUM_OF_SERVICE_LEVELS - 1];

 /**
  * Constructs a single-hop path between a given source node and a destination
  * node.
  * @param iID the PathID to be assigned to the new path.
  * @param index cross reference to Path Info array.
  * @param baInterfaceID the IPv6 address of the outbound interface.
  * @param baNeighborInterfaceID the IPv6 address of the neighbor interface.
  * @param iBandwidth the interface bandwidth.
  */
  public Path (
    Integer iID,
    aPIIndex index,
    IPv6Address baInterfaceID,
    IPv6Address baNeighborInterfaceID,
    int iBandwidth)
  {
    iPathID = new Integer(iID.intValue());
    objaPIIndex = index;

    vNodeSequence = new Vector();   // For the node sequence
    vNodeSequence.addElement(index.getDestination());
    vNodeSequence.addElement(index.getSource());

    vInterfaceSequence = new Vector (); // For the interface sequence

    //Only outbound interfaces is added to the interface sequence
    vInterfaceSequence.addElement(baInterfaceID);

    for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
    {
      // Initalize QoS for each in-profile service level
      objPathQoS[sl] = new PathQoS();

      // Allocate an empty hashtable for flowIDs
      ahtFlows[sl] = new Hashtable( );
    }

    refreshPathQoS(this);

  } // End Path() constructor for single-hop path

 /**
  * Constructs a new multi-hop path between a given source node and and
  * destination node, from a given existing path and appending from front
  * (source).
  * @param iNewPathID the PathID to be assigned to the new path
  * @param iOldPathID the PathID of the old path having the new source node
  * appended
  * @param index the aPIIndex.
  * @param baInterfaceID the IPv6 address of the outbound interface.
  * @param baNeighborInterfaceID the IPv6 address of the neighbor interface
  * (inbound).
  * @param iBandwidth the interface total bandwidth.
  */
  public Path (
    Integer iNewPathID,
    Integer iOldPathID,
    aPIIndex index,
    IPv6Address baInterfaceID,
    IPv6Address baNeighborInterfaceID,
    int iBandwidth)
  {
    iPathID = new Integer(iNewPathID.intValue());
    objaPIIndex = index;

    // Extract the old path from  iOldPathID
    Path oldPath = (Path) htPaths.get(iOldPathID);

    // Stores the sequence of nodes for this path
    vNodeSequence = new Vector();

    // Copy the old path's nodes
    vNodeSequence = (Vector) oldPath.vNodeSequence.clone();
```

```
    // Simply add the new source
    vNodeSequence.addElement(index.getSource());

    // Stores the sequence of interfaces for this path
    vInterfaceSequence = new Vector ();

    // Copy the old interfaces sequence
    vInterfaceSequence = (Vector) oldPath.vInterfaceSequence.clone();

    // Only the neighbor interface need to be added. Since the node is added
    // at the front, the neighbor interface is an outbound interface to this
    // path. The parameter list of this constructor does not need
    // baInterfaceID
    vInterfaceSequence.addElement(baInterfaceID);

    for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
    {
      // Initalize Qos for each service level
      objPathQoS[sl] = new PathQoS();

      // Allocate an empty hashtable for flowIDs
      ahtFlows[sl] = new Hashtable ( );

    } // End for-loop for initializing Path QoS and flows

    refreshPathQoS(this);

  } // End Path() constructor for appending new node to an existing path

 /**
  * Constructs a new path by means of concatenating two existing paths.
  * @param sourceLeg the source section path. Terminated at an interface on an
  * interfaceces to a subnet common to both paths.
  * @param type destinationLeg the destination path section. Contains all
  * interfaces traversed tp get to the destination node, beginning at the
  * interface on the common sunnet.
  * @param baInterfaceID the IPv6 address of the outbound interface.
  * @param baNeighborInterfaceID the IPv6 address of the neighbor interface
  * (inbound).
  * @param iBandwidth the total bandwidth of the outbound inteface.
  */
  public Path (
    Path SourceLeg,
    Path DestinationLeg,
    IPv6Address newInterfaceID,
    IPv6Address neighborInterfaceID,
    int iBandwidth)
  {
    // Create new path ID
    short newID = iNextPathID++;
    iPathID = new Integer(newID);

    // Extract the source and destination path API indexes to provide the
    // cooresponding indexes to create the new PIBarray index for the new
    // path : aPIIndex objaPIIndex
    aPIIndex SourceIndex =  SourceLeg.getaPIIndex();
    Integer SourceNode = SourceIndex.getSource();

    aPIIndex DestinationIndex =  DestinationLeg.getaPIIndex();
    Integer DestinationNode = DestinationIndex.getDestination();

    // Generate hop count from the sum of the hop counts of the path pair and
    // add 1 to account for the addition of the link joining the two paths
    int HopCount =
      SourceIndex.getHopCount() +  DestinationIndex.getHopCount() + 1;

    // Create the actual aPI index
    objaPIIndex = new aPIIndex( SourceNode, DestinationNode, HopCount);

    // [GX] Can we concatenate two vectors in a more efficient fashion?
    // Create new node sequence vector by concatenating the two node sequences:

    // Stores the sequence of nodes for this path
    vNodeSequence = new Vector ();

    Enumeration vDestinationNodeSequence =
      DestinationLeg.getNodeSequence().elements();
    while (vDestinationNodeSequence.hasMoreElements())
    {
      vNodeSequence.addElement(vDestinationNodeSequence.nextElement());
    }

    Enumeration vSourceNodeSequence = SourceLeg.getNodeSequence().elements();
    while (vSourceNodeSequence.hasMoreElements())
    {
      vNodeSequence.addElement(vSourceNodeSequence.nextElement());
    }

    // Assign to the new interface sequence the interface squence of dest. leg
    vInterfaceSequence = DestinationLeg.getInterfaceSequence();

    vInterfaceSequence.addElement(newInterfaceID);

    Enumeration srcLegInterfaces = SourceLeg.getInterfaceSequence().elements();
    while(srcLegInterfaces.hasMoreElements())
    {
      vInterfaceSequence.add(srcLegInterfaces.nextElement());
    }

    // Creates new QoS array for the path's service levels and create flow
    // hashtables
    for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
    {
      objPathQoS[sl] = new PathQoS(); // Initalize Qos for each service level
      ahtFlows[sl] = new Hashtable ( ); // Hashtable of flowIDs
    }

    refreshPathQoS(this);

  } // End path constructor: concatenate two existing multi-hop paths

 /**
  * Constructs a new path from a given path, traversing the same nodes, but in
  * oposite direction.
  * @param originalPath the path to be mirrored
  */
  public Path (Path originalPath)
  {
    // Create new path ID
    short newID = iNextPathID++;
    iPathID = new Integer(newID);

    // Create new index to PIBarray by reversing source and destinations
    aPIIndex originalPathIndex =  originalPath.getaPIIndex();
    Integer newSource = originalPathIndex.getDestination();
    Integer newDestination = originalPathIndex.getSource();
    int numberHops = originalPathIndex.getHopCount();
    objaPIIndex = new aPIIndex( newSource, newDestination, numberHops );

    // Generate node sequence for mirror path from original path
    Vector vOriginalNodeSeq = (Vector)originalPath.getNodeSequence();
    vNodeSequence = new Vector ();
    for (int index = vOriginalNodeSeq.size() - 1; index >= 0; index--)
    {
      vNodeSequence.add((Integer) vOriginalNodeSeq.elementAt(index));
    } // End reversed sequence of nodes traversed by original path

    // Get the original interface sequence
    vInterfaceSequence = new Vector ();
    Vector vOriginalIFaceSeq =
      (Vector)originalPath.getInterfaceSequence();
```

```
// Traverse the nodes of the new sequence and discover which interfaces
// (outbound) must now be included in the new inversed/paired
// interface sequence. Add them to the new vector
for (int index = 1; index <= vNodeSequence.size() - 1; index++)
{
  // Get the inbound (neighboring) interface from the original sequence
  IPv6Address neighborIFace =
    (IPv6Address) vOriginalIFaceSeq.elementAt(
      vOriginalIFaceSeq.size() - index);

  // Get the subnet mask of the link
  byte subnetMaskLength = ((InterfaceInfo)
    htInterfaces.get(neighborIFace.toString())).getSubnetMask();

  // Get the set of interfaces for determining outbound interface
  // First get current nodeID
  Integer nodeID = (Integer) vNodeSequence.elementAt(index);

  // Need to transform nodeID into IPv6Address based RouterID
  IPv6Address routerID = (IPv6Address) htNodeIDtoRouterID.get(nodeID);

  // Get the vector of all interfaces hosted by this RouterID
  Hashtable iFaces =
    (Hashtable) htRouterInterfaceMap.get(routerID.toString());

  // Now search for the matching outbound interface
  Enumeration e = iFaces.elements();
  IPv6Address outboundIFace = null;

  while (e.hasMoreElements())
  {
    IPv6Address check = (IPv6Address) e.nextElement();
    if (Interface.isOnSameNetwork(check, neighborIFace, subnetMaskLength))
    {
      outboundIFace = check;
      break;
    }
  }

  // Add this outbound interface to the interface sequence of mirrored path
  vInterfaceSequence.add(outboundIFace);

} // End of the loop to invert and mirror the original interface sequence

// Copy the original path's quality of service parameters
for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
{
  objPathQoS[sl] = new PathQoS(); // Initalize QoS for each service level

  ahtFlows[sl] = new Hashtable( );
} // End for-loop for initializing Path QoS and flows

// Update pertinent tables

// Add mirror path to the htPaths hashtable
htPaths.put(iPathID, this);

// Add mirror pathID to aPI entry
aPI [newSource.intValue()]
    [newDestination.intValue()]
    [numberHops].put(iPathID, iPathID);

// Add the pathID to each traversed interface's hashtable of pathIDs
Enumeration enumInterfacesTraversed =
  this.getInterfaceSequence().elements();
while (enumInterfacesTraversed.hasMoreElements())
{
  IPv6Address InterfaceAddress =
    (IPv6Address) enumInterfacesTraversed.nextElement();
    InterfaceInfo InfoObject
        = (InterfaceInfo) htInterfaces.get(InterfaceAddress.toString());

    Hashtable PathTable = (Hashtable) InfoObject.getPathIDs();

    PathTable.put(iPathID, iPathID);

  } // End of loop to update mirror path's interface hashtables

  refreshPathQoS(this);

} // End path constructor for return path for an existing path (mirror image)

/**
 * Gets the path ID
 * @return the path ID integer wrapped as an Integer object
 */
public Integer getPathID()
{
  return iPathID;
}

/**
 * Gets the reference of this path in the array of path information objects.
 * @return the reference to the array of path information objects.
 */
public aPIIndex getaPIIndex()
{
  return objaPIIndex;
}

/**
 * Gets the array of path QoS objects associated with each service level.
 * @return the array of PathQoS objects.
 */
public PathQoS[] getPathQoSArray()
{
  return objPathQoS;
}

/**
 * Gets the QoS object of a given service leve.
 * @param bServiceLevel the given service level.
 * @return the PathQoS object of the specified service level.
 */
public PathQoS getPathServiceLevelQoS(byte bServiceLevel)
{
  return objPathQoS[bServiceLevel];
}

/**
 * Gets the vector with the node sequence the path traverses. Nodes are
 * listed from destination node to source node.
 * @return the Vector with the node sequence.
 */
public Vector getNodeSequence()
{
  return vNodeSequence;
}

/**
 * Gets the vector with the sequence of interfaces the path traverses. The
 * list contain only outbound interfaces, from destination to source.
 * @return a Vector with all the interface sequence
 */
public Vector getInterfaceSequence()
{
  return vInterfaceSequence;
}

/**
```

```
 * Gets the next available flow ID for this path.
 * @return the next flow ID.
 */
public int getNewFlowID()
{
  if (++iNewFlowID > MAX_FLOW_ID)
  {
    iNewFlowID = MIN_FLOW_ID;
  }
  return iNewFlowID;
}

/**
 * Gets an hashtable with all flow ID's currently associated with this path,
 * for the given service level.
 * @param iServiceLevel the given service level.
 * @return an hashtable with all flow ID's of the given service level.
 */
public Hashtable getFlowIDs (int iServiceLevel)
{
  return ahtFlows[iServiceLevel];
}

/**
 * Adds a new flow to the path, in a given service level.
 * @param iServiceLevel the given service level.
 * @param iNewFLowLabel the flow label of this flow
 * @return void
 */
public void AddFlow (byte iServiceLevel, int iNewFlowLabel)
{
  ahtFlows[iServiceLevel].put(
    new Integer(iNewFlowLabel), new Integer(iNewFlowLabel));
}

/**
 * Adds a new flow to the path, for a given service level.
 * @param iServiceLevel the given service level.
 * @param iNewFLowLabel the flow label of this flow
 * @param flowQoS the FlowQoS characteristic of the given flow
 * @return void
 */
public void AddFlow (byte  iServiceLevel, int iNewFlowLabel, FlowQoS flowQoS)
{
  ahtFlows[iServiceLevel].put(new Integer(iNewFlowLabel), flowQoS);
}

/**
 * Removes a given flow from the path.
 * @param iServiceLevel the service level to which the flow belongs
 * @param iFlowLabel the label of the flow to be removed
 * @return void
 */
public void RemoveFlow (byte iServiceLevel, Integer iFlowLabel)
{
  ahtFlows[iServiceLevel].remove(iFlowLabel);
}

/**
 * Deletes all flows associated with a path, in a given service level.
 * @param iServiceLevel the given service level
 * @return void
 */
public void DeleteAllFlows (int iServiceLevel)
{
  ahtFlows[iServiceLevel].clear( );
}

/**
 * Returns a String with the path ID and the sequence of node ID the path
 * traverses
```

```
   */
  public String toString()
  {
    String str;

    // Add path ID
    str = this.getPathID().toString() + "\t";

    // Step through each node the path traverses
    Enumeration eNodeSeq = this.getNodeSequence().elements();

    str += ( (Integer)eNodeSeq.nextElement() ).toString();
    while(eNodeSeq.hasMoreElements())
    {
      // Append arrow separator
      str += " < " +
        ((Integer)eNodeSeq.nextElement()).toString();
    }

  return str;

  } // End of toString()

} // End of Path class


/**
 * The PathQoS class represents the key parameters values for a path's
 * Quality of Service for a given service level.
 */
private class PathQoS
{
  /**
   * Minimum available bandwidth among all interfaces transversed by path
   * (outbound interfaces only). The available bandwidth being considered
   * is the bandwidth not being used by a service level out of the initial
   * base allocation (all service levels) plus the interface total unallocated
   * bandwidth (only for service levels that may grow dynamically: IntServ
   * and DiffServ).
   */
  int pathAvBW;

  /**
   * Minimum available bandwidth among all interfaces transversed by path,
   * inluding inter-service borrowing. The bandwidth being considered is the
   * summation of the pathAvBW plus any bandwidth that may be made available
   * throug inter-service borrowing (only IntServ and DiffServ).
   */
  int pathAvBWwBorrowing;

  /**
   * Sum of the delays of all hops taken by path(2 bytes)
   */
  short  pathDelay;

  /**
   * Sum of the loss rates of all hops taken by path(2 bytes)
   */
  short  pathLossRate;

  /**
   * Constructs the default PathQoS object.
   */
  public PathQoS ( )
  {
    pathAvBW = 0;
    pathDelay = 0;
    pathLossRate = 0;
  }

  /**
   * Updates the path delay given a differential delay value.
   * @param delayDelta the delay variation to be introduced.
   * @return void.
   */
  public void updatePathDelay (short delayDelta)
  {
    pathDelay += delayDelta;
  }

  /**
   * Updates the path delay given a differential loss rate value.
   * @param lossRateDelta the loss rate variation to be introduced
   * @return void
   */
  public void updatePathLossRate (short lossRateDelta)
  {
    pathLossRate += lossRateDelta;
  }

  /**
   * Sets a new value for path available bandwidth.
   * @param iBandwidth the new bandwidth value.
   * @return void
   */
  public void setAvailableBandwidth (int iBandwidth)
  {
    pathAvBW = iBandwidth;
  }

  //[PS] created to provide for inter-service borrowing
  /**
   * Sets a new value for path available bandwidth including borrowing
   * @param int the new value for the available bandwidth
   * @return void
   */
  public void setAvailableBandwidthIncludingBorrowing(int iBandwidth)
  {
    pathAvBWwBorrowing = iBandwidth;

    //Ensures that it is never less than zero
    pathAvBWwBorrowing =
      Math.max(0, pathAvBWwBorrowing);
  }

  /**
   * Gets the available bandwidth.
   * @return the available bandwidth
   */
  public int getAvailableBandwidth()
  {
    return pathAvBW;
  }

  //[PS] created to provide for inter-service borrowing
  /**
   * Gets the path available bandwidth including inter-service borrowing.
   * @return the available bandwidth, including inter-service borrowing.
   */
  public int getAvailableBandwidthIncludingBorrowing()
  {
    return pathAvBWwBorrowing;
  }

  /**
   * Gets path delay.
   * @return the current path delay.
   */
  public short getPacketDelay()
  {
    return pathDelay;
  }
```

```
    /**
     * Gets packet loss rate
     * @return the current value of loss rate
     */
    public short getPacketLossRate()
    {
      return pathLossRate;
    }

    /**
     * Sets a new value for the path delay.
     * @param newDelay the new value for path delay.
     * @return void.
     */
    public void setDelay (short newDelay)
    {
      pathDelay = newDelay;
    }

    /**
     * Sets a new value for the path level loss rate.
     * @param newLossRate the new value for the path loss rate.
     * @return void.
     */
    public void setLossRate (short newLossRate)
    {
      pathLossRate = newLossRate;
    }

  } // End PathQoS class


 /**
  * FlowQoS Class defines objects that represent the key QoS characteristics of
  * a flow request.
  */
  private class FlowQoS
  {
    /**
     * The requested delay bound.
     */
    private short requestedDelay;

    /**
     * The requested loss rate bound.
     */
    private short requestedLossRate;

    /**
     * The requested bandwidth.
     */
    private int requestedBandwidth;

    /**
     * The request time stamp reference.
     */
    private long timeStamp;

    //[GX} other fields such as typeOfApplication may be added

    /**
     * Constructs a default FlowQoS object.
     */
    public FlowQoS()
    {
      timeStamp          = 0;
      requestedBandwidth = 0;
      requestedDelay     = 0;
      requestedLossRate  = 0;
    } // End of FlowQoS default constructor
```

```
/**
 * Constructs a FlowQoS with a given timeStamp. Used by BE flows.
 * @param timeStamp the specified time stamp
 */
public FlowQoS(long timeStamp)
{
  this.timeStamp          = timeStamp;
  this.requestedBandwidth = Best_Effort_Allocated_Bandwidth;
  //[GX] need to allow variable BW
}

/**
 * Constructs a FlowQoS with the given initialization values. Used by
 * IntServ and DiffServ flows.
 * @param timeStamp the given time stamp
 * @param requestedBandwidth the given requested bandwidth
 * @param requestedDelay the given requested delay
 * @param requestedLossRate the given requested loss rate
 */
public FlowQoS(
  long timeStamp,
  int   requestedBandwidth,
  short requestedDelay,
  short requestedLossRate)
{
  this.timeStamp          = timeStamp;
  this.requestedBandwidth = requestedBandwidth;
  this.requestedDelay     = requestedDelay;
  this.requestedLossRate  = requestedLossRate;
}

/**
 * Sets time stamp with a given value.
 * @param time the new time stamp value.
 * @return void.
 */
public void setTimeStamp(long time)
{
  timeStamp = time;
}

/**
 * Gets the time stamp value.
 * @return the time stamp value.
 */
public long getTimeStamp()
{
  return timeStamp;
}

/**
 * Sets requestedBandwidth with a given value.
 * @param bandwidth the given requested bandwidth.
 * @return void.
 */
public void setRequestedBandwidth(int bandwidth)
{
  requestedBandwidth = bandwidth;
}

/**
 * Gets the requested bandwidth value.
 * @return the requested bandwidth.
 */
public int getRequestedBandwidth()
{
  return requestedBandwidth;
}

/**
```

```
   * Sets the requested delay with a given value
   */
  public void setRequestedDelay(short delay)
  {
    requestedDelay = delay;
  }

  /**
   * Gets the requested delay value.
   * @return the requested delay.
   */
  public short getRequestedDelay()
  {
    return requestedDelay;
  }

  /**
   *  Sets requested loss rate with a given loss rate.
   *  @param lossRate the given loss rate.
   *  @return void.
   */
  public void setRequestedLossRate(short lossRate)
  {
    requestedLossRate = lossRate;
  }

  /**
   * Gets requested loss rate value.
   * @return the requested loss rate.
   */
  public short getRequestedLossRate( )
  {
    return requestedLossRate;
  }

} // End FlowQoS class

/**
 * The ObsQoS class defines an object responsible for storing observed QoS
 * parameters associated with a given pair of interfaceservice - service
 * level.
 */
private class ObsQoS
{
  /**
   * The service level utilization of bandwidth, as measured at the rounter
   * interface. Measured in percentage and affected by the current unit of
   * measure (ServiceSA.UTIL_UNIT).
   */
  private short  iUtilization;

  /**
   * The actual packet delay accross a single hop, as measured and reported
   * by a router.
   */
  private short  iDelay;

  /**
   * The actual packet loss rate accross a single hop as measures and reported
   * by a rounter.
   */
  private short  iLossRate;

  /**
   * Constructs an ObsQoS object with default initialization parameters.
   */
  public ObsQoS ( )
  {
    iUtilization = 0;
    iDelay       = 0;
    iLossRate    = 0;
  }

  /**
   * Constructs an ObsQoS objects, initialized with the given parameters.
   * @param utilization the given interface utilization.
   * @param delay the link delay.
   * @param lossRate the given link loss rate
   */
  public ObsQoS (short utilization, short delay, short lossRate)
  {
    iUtilization = utilization;
    iDelay       = delay;
    iLossRate    = lossRate;
  }

  /**
   * Sets utilization with a given utilization.
   * @param obsUtil the given utilization.
   * @return void
   */
  public void setUtilization (short obsUtil)
  {
    iUtilization = obsUtil;
  }

  /**
   * Gets the utilization value.
   * @return the utilization value.
   */
  public short getUtilization()
  {
    return iUtilization;
  }

  /**
   * Sets delay with a given delay value.
   * @param obsDelay the given delay value.
   * @param void.
   */
  public void setDelay (short obsDelay)
  {
    iDelay = obsDelay;
  }

  /**
   * Gets the delay value.
   * @return the delay value.
   */
  public short getDelay()
  {
    return iDelay;
  }

  /**
   * Sets the loss rate with a given loss rate value.
   * @param obsLossRate the given new loss rate value.
   * @return void.
   */
  public void setLossRate (short obsLossRate)
  {
    iLossRate = obsLossRate;
  }

  /**
   * Gets the loss rate value.
   * @return the loss rate value.
   */
  public short getLossRate()
  {
    return iLossRate;
  }
```

```
  /**
   * Resets the QoS array.
   * @return void
   */
  public void resetQoS()
  {
    iDelay = 0;
    iLossRate = 0;
    iUtilization = 0;
  }

} // End of ObsQoS class

/**
 * The InterfaceInfo class defines the objects that contain all key
 * information associated with a single SAAM interface.
 */
private class InterfaceInfo
{
  /**
   * The router ID is an wrapped integer that uniquely identifies the router
   * hosting the interface.
   */
  private Integer iNodeID;

  /**
   * The total bandwidth the interface supports.
   */
  private int iTotalBandwidth;

  /**
   * An array on integers that represent the available bandwidth of each
   * service level, indexed by service level number. The available bandwidth
   * represents the amount of bandwidth out of the initial base allocation
   * that is not utilized by the service. Additionally for those service
   * levels that allow for dynamic growing (IntServ and DiffServ), is also
   * includes interface bandwidth that is currently no allocated to any
   * service (Unallocated bandwidth).
   */
  private int[] iServiceLevelAvailableBandwidth =
    new int[NUM_OF_SERVICE_LEVELS - 1];

  /**
   * An array on integers that represent the bandwidth capacity that the
   * service makes available for inter-service borrowing. The array is indexed
   * by service level numbers.
   */
  private int[] iServiceLevelBorrowingCapacity =
    new int[NUM_OF_SERVICE_LEVELS - 1];

  /**
   * The interface bandwidth capacity that is not allocated/claimed to any of
   * the service levels.
   */
  private int iUnallocatedBandwidth;

  /**
   * The number of bits used to determine the interface's network ID out of
   * the interface IPv6 address.
   */
  private byte bSubnetMask;

  /**
   * The table containing all path IDs that traverse this interface (outbound
   * direction).
   */
  private Hashtable htPathIDs;

  /**
   * The array of ObsQoS objects, indexed by service level. Each ObsQoS objetc
   * describes the observed QoS characteristics of a single service level
   * accross the interface.
   */
  private ObsQoS aobjObsQoS[] = new ObsQoS[NUM_OF_SERVICE_LEVELS - 1];

  /**
   * Constructs an InterfaceInfo object with the given initialization
   * parameters.
   * @iNewID the given interface ID number.
   * @iBandwidth the given interface total bandwidth.
   * @param bSubnetMask the given number of bits for the subnet mask.
   * @param htIDs the hash table of paths that traverse this interface.
   */
  public InterfaceInfo (
    Integer iNewID, int iBandwidth, byte bSubnetMask, Hashtable htIDs)
  {
    float allocationFactor = 0.0f;
    setNodeID (iNewID);
    setTotalBandwidth (iBandwidth);

    for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
    {
      iServiceLevelAvailableBandwidth[sl] =
        (int) (iBandwidth * afBASE_ALLOCATION[sl] * afLOAD_FACTOR[sl]);
      allocationFactor += afBASE_ALLOCATION[sl];
      iServiceLevelBorrowingCapacity[sl] = 0;
      aobjObsQoS[sl] = new ObsQoS((byte) 0, (short) 0, (short) 0);
    }

    setUnallocatedBandwidth(iBandwidth -
      (int)((float)iBandwidth * allocationFactor));
    setSubnetMask (bSubnetMask);
    setPathIDs (htIDs);

    refreshInterfaceQoS(this);

  } // End of constructor()

  /**
   * Sets the node ID with a given value.
   * @param iNewID the given value for node ID.
   * @return void.
   */
  public void setNodeID (Integer iNewID) {
    iNodeID = iNewID;
  }

  /**
   * Gets the node ID.
   * @return the node ID.
   */
  public Integer   getNodeID( )
  {
    return iNodeID;
  }

  /**
   * Sets the interface total bandwidth with a given value.
   * @param iBW the new given bandwidth value.
   * @return void.
   */
  public void   setTotalBandwidth (int iBW)
  {
    iTotalBandwidth = iBW;
  }

  /**
   * Gets the bandwidth of the interface.
   * @return the bandwidth to the interface.
   */
  public int getTotalBandwidth( )
```

```
{
  return iTotalBandwidth;
}

/**
 * Sets the available bandwidth of given service level with a new value.
 * @param iBandwidth the new bandiwdth.
 * @param bSL the service level.
 * @return void.
 */
public void setServiceLevelAvailableBandwidth (int iBandwidth, byte bSL )
{
  iServiceLevelAvailableBandwidth[bSL] = iBandwidth;
}

//[PS]
/**
 * Sets a new value for the unallocated bandwidth.
 * @param iNewBWValue the new unallocated bandwidth value
 * @return void
 */
public void setUnallocatedBandwidth(int iNewBWValue)
{
  iUnallocatedBandwidth = iNewBWValue;
}

//[PS] - created to support inter-service borrowing
/**
 * Sets the borrowing capacity for a given service level.
 * @param borrowingCapacity integer w/ the bandwidth available for borrowing
 * @param serviceLevel the service level the bandwidth refers to
 * @return void
 */
public void setServiceLevelBorrowingCapacity(int iBCapacity, byte bSL )
{
  iServiceLevelBorrowingCapacity[bSL] = iBCapacity;
}

/**
 * Gets the current value of available bandwidth, for the specified
 * service level.
 * @param svcLevel the servicel level.
 * @return an integer with the amount of available bandwidth
 */
public int getServiceLevelAvailableBandwidth(byte bSL)
{
  return iServiceLevelAvailableBandwidth[bSL];
}

//[PS] - required to enable inter-service borrowing
/**
 * Retrives the borrowing capacity for the required service level
 * @param iSL the required service level
 * @return an integer with the required value of borrowing capacity
 */
public int getServiceLevelBorrowingCapacity(byte bSL)
{
  return iServiceLevelBorrowingCapacity[bSL];
}

//[PS] - required to enable inter-service borrowing
/**
 * Retrives the current value of unallocated bandwidth
 * @return an integer with the current value of unallocated bandwidth
 */
public int getUnallocatedBandwidth()
{
  return iUnallocatedBandwidth;
}

/**
```

```
   * Sets the subnet mask number with a given value.
   * @param bSMask the given new value for the subnet mask.
   * @return void.
   */
  public void setSubnetMask (byte bSMask)
  {
    bSubnetMask = bSMask;
  }

  /**
   * Gets the subnet mask value.
   * @return the value of the subnet mask.
   */
  public byte getSubnetMask( )
  {
    return bSubnetMask;
  }

  /**
   * Sets the table of path IDs with a given table.
   * @param htIDs the given table of path IDs.
   * @return void.
   */
  public void setPathIDs (Hashtable htIDs)
  {
    htPathIDs = htIDs;
  }

  /**
   * Gets the table of path IDs.
   * @return the table of path IDs.
   */
  public Hashtable getPathIDs( )
  {
    return htPathIDs;
  }

  /**
   * Sets the obsQoS of a specified service level with a given value.
   * @param obsQoS the new obsQoS value.
   * @param bSL the specified service level.
   * @return void.
   */
  public void setQoS (ObsQoS obsQoS, byte bSL)
  {
    aobjObsQoS[bSL] = obsQoS;
  }

  /**
   * Gets the array of obsQoS objects for all service levels.
   * @return the array of obsQoS objects for all service levels.
   */
  public ObsQoS[] getQoS( )
  {
    return aobjObsQoS;
  }

  /**
   * Resets the interface QoS array.
   * @return void.
   */
  public void resetQoS()
  {
    testMsg("InterfaceInfo.resetQoS()");
    float allocationFactor = 0.0f;

    for (int sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
    {
      iServiceLevelAvailableBandwidth[sl] =
        (int) (iTotalBandwidth * afBASE_ALLOCATION[sl] * afLOAD_FACTOR[sl]);
      allocationFactor += afBASE_ALLOCATION[sl];
      iServiceLevelBorrowingCapacity[sl] = 0;
      aobjObsQoS[sl].resetQoS();
    }

    setUnallocatedBandwidth(iTotalBandwidth -
      (int)((float)iTotalBandwidth * allocationFactor));

    refreshInterfaceQoS(this);

  }//end of resetQoS()

} // End of InterfaceInfo class

/**
 * RoutingAlgorithm class implements a set of routing algorithms required to
 * find paths in the PIB structure, meeting a set of given requirements.
 */
private class RoutingAlgorithm
{
  /**
   * For the selection of the First-Shortest Path algorithm.
   */
  public static final byte FIRST_SHORTEST_PATH = 1;

  /**
   * For the selection of the Shortest-Widest Path algorithm.
   */
  public static final byte WIDEST_SHORTEST_PATH = 2;

  /**
   * For the selection of the Shortest Widest Path algorithm.
   */
  public static final byte SHORTEST_WIDEST_PATH = 3;

  /**
   * Default constructor. The RoutingAlgorithm class is a stateless
   * class, and so there exist only a default empty constructor.
   */
  public RoutingAlgorithm()
  {
    // Empty default constructor
  }

 /**
  * Implementation of the First Shortest Path algorithm for Best Effort.
  * The admission procedure for BE requires revion. Currently, a BE flow is
  * admited if there is more than 50k of path available BW in the BE svc level.
  * @param srcRterID the IPv6 address of the source router.
  * @param destRterID the IPv6 address of the destination router.
  * @return the required path or null if no path was found.
  */
  private Path findPathFSP(IPv6Address srcRterID, IPv6Address destRterID)
  {
    testMsg("findPathFSP() for Best Effort");

    InterfaceInfo interfaceInformation =
        (InterfaceInfo) htInterfaces.get(srcRterID.toString());

    int sourceNodeID = ((InterfaceInfo) htInterfaces.
      get(srcRterID.toString())).getNodeID().intValue();

    int destinationNodeID = ((InterfaceInfo) htInterfaces.
      get(destRterID.toString())).getNodeID().intValue();

    Path bestPath = null;

    Hashtable table = new Hashtable();

    //labeled compound statement
    stop:
    {
```

```
    for (int i = 1; i < MAX_HOP_COUNT; i++ )
    {
      testMsg("Hop count = " + i);
      table = aPI[sourceNodeID][destinationNodeID][i];
      Enumeration enum = table.elements();

      if (enum.hasMoreElements())
      {
        //Cycle through each of the paths of the current hop count, between
        //source and destination nodes
        while (enum.hasMoreElements())
        {
          Integer currentPathID = (Integer) enum.nextElement();

          // Extract current path information
          bestPath = (Path) htPaths.get(currentPathID);

          int availableBandwidth =
            bestPath.getPathServiceLevelQoS(BEST_EFFORT).
            getAvailableBandwidth();

          //
          // If available BW is greater than 50k the flow is admited
          // This is a magic number
          if (availableBandwidth >=  50000)
          {

            display.sendText(
              "\t     The selected path is:\t" + bestPath.toString() + "\n" +
              "\t     Available bandwidth: \t" + availableBandwidth);

            break stop;
          }//End of if structure

        }//End of while-loop

      }//End of if structure

    }//End of for-loop

  }//End of labeled stop structure

  return bestPath;

}//End of findPath() for Best Effort Service

/**
 * Searches the PIB path structure for a path between given source and
 * destination routers.
 * Different routing algorihtms may be used depending on user option.
 * Currently, only First-Widest Path algorithm (FSP) is implemented.
 * @param sourceRouterID the IPv6 address of the source router.
 * @param destinationRouterID the IPv6 address of the destination router.
 * @param rotingAlgorithm the routing algorithm to be used.
 * @return the Path object of a suitable path or null if otherwise.
 */
protected Path findPath(
  IPv6Address sourceRouterID,
  IPv6Address destinationRouterID,
  byte routingAlgorithm)
{
  Path resultPath = null;

  switch(routingAlgorithm)
  {
    case FIRST_SHORTEST_PATH:
      resultPath = findPathFSP(sourceRouterID, destinationRouterID);
      break;

    case SHORTEST_WIDEST_PATH:
      display.sendText(
        "\tShortest-Widest Path routing algorithm not implemented");
      break;

    case WIDEST_SHORTEST_PATH:
      display.sendText(
        "\tWidest-Shortest Path routing algorithm not implemented");
      break;

    default:
      display.sendText("\t\tInvalid type of routing algorithm");
  }
  return resultPath;
} // end of findPath()

/**
 * Searches the PIB path structure for a path that fulfills the specified
 * QoS requirements. If interservice borrowing is enabled, a path using
 * interservice borrowing will only be selected if no paths exists (
 * (any hop count) considering no interservice borrowing bandwidth.
 * Different routing algorihtms may be used depending on user option.
 * Currently, only First-Widest Path algorithm (FSP) is implemented.
 * @param sourceRouterID the IPv6 address of the source router.
 * @param destinationRouterID the IPv6 address of the destination router.
 * @param requestedBandwidth the requested bandwidth.
 * @param requestedDelay the requested upper bound on end-to-end delays.
 * @param requestedLossRate requested upper bound on end-to-end loss rate.
 * @param bSL the service level (INT_SERV or DIFF_SERV).
 * @param isBorrowingAllowed whether inter-service borrowing is allowed.
 * @param rotingAlgorithm the routing algorithm to be used.
 * @return the Path object of a suitable path or null if otherwise.
 */
protected Path findPath(
  IPv6Address sourceRouterID,
  IPv6Address destinationRouterID,
  int requestedBandwidth,
  short requestedDelay,
  short requestedLossRate,
  byte bSL,
  boolean isBorrowingAllowed,
  byte routingAlgorithm)
{
  Path resultPath = null;

  switch(routingAlgorithm)
  {
    case FIRST_SHORTEST_PATH:
      resultPath = findPathFSP(
                         sourceRouterID,
                         destinationRouterID,
                         requestedBandwidth,
                         requestedDelay,
                         requestedLossRate,
                         bSL,
                         isBorrowingAllowed);
      break;

    case SHORTEST_WIDEST_PATH:
      display.sendText(
        "\tShortest-Widest Path routing algorithm not implemented");
      break;

    case WIDEST_SHORTEST_PATH:
      display.sendText(
        "\tWidest-Shortest Path routing algorithm not implemented");
      break;

    default:
      display.sendText("\t\tInvalid type of routing algorithm");
  }
  return resultPath;
} // end of findPath()
```

```
/**
 * Implementation of the First Shortest Path with inter-service borrowing
 * capability. Searches the PIB for a path that can provide the required QoS
 * guarantees. Shortest path are first evaluated and the first meeting the
 * QoS requirements is selected. If interservice borrowing is enabled,
 * a path using interservice borrowing will only be selected if no paths
 * exists (any hop count) considering no interservice borrowing bandwidth.
 * This implementation is therefore the shortest in time and the most
 * efficient search algorithm.
 * @param sourceRouterID the IPv6 address of the source router.
 * @param destinationRouterID the IPv6 address of the destination router.
 * @param requestedBandwidth the requested bandwidth.
 * @param requestedDelay the requested upper bound on end-to-end delays.
 * @param requestedLossRate requested upper bound on end-to-end loss rate.
 * @param bSL the service level (INT_SERV or DIFF_SERV).
 * @param isBorrowingAllowed whether inter-service borrowing is allowed.
 * @return the Path object of a suitable path or null if otherwise.
 */
private Path findPathFSP(
  IPv6Address sourceRouterID,
  IPv6Address destinationRouterID,
  int requestedBandwidth,
  short requestedDelay,
  short requestedLossRate,
  byte bSL,
  boolean isBorrowingAllowed)
{
  testMsg("findPathFSP() for IntServS or DiffServ");

  Path foundPath = null;
  Path foundPathWithBorrowing = null;
  Path tempPath = null;

  int availableBW = 0;
  int availableBWIncludingBorrowing = 0;
  short delayBound = 0;
  short lossRateBound = 0;

  InterfaceInfo interfaceInformation =
      (InterfaceInfo) htInterfaces.get(sourceRouterID.toString());

  //Source node
  int sourceNodeID = ((InterfaceInfo)
    htInterfaces.get(sourceRouterID.toString())).getNodeID().intValue();

  //Destination node
  int destinationNodeID = ((InterfaceInfo)
    htInterfaces.get(destinationRouterID.toString())).getNodeID().intValue();

  //There exists at least a physical path, so let's proceed
  Hashtable table = new Hashtable();

  testMsg("Destination node is reachable from source node");

  //labeled compound statement
  stop:
  {

    for (int i = 1 ; i < MAX_HOP_COUNT ; i++ )
    {
      testMsg("Hop count = " + i);
      table = aPI[sourceNodeID][destinationNodeID][i];
      Enumeration enum = table.elements();

      if (enum.hasMoreElements())
      {
        //Cycle through each of the paths of the current hop count, between
        //source and destination nodes
        while (enum.hasMoreElements())
        {
```

```
Integer currentPathID = (Integer) enum.nextElement();

// Extract that path QoS information
tempPath = (Path) htPaths.get(currentPathID);

//Bandwidth
availableBW =
  tempPath.getPathServiceLevelQoS(bSL).getAvailableBandwidth();

//Bandwidth including inter-service borrowing
availableBWIncludingBorrowing =
  tempPath.getPathServiceLevelQoS(bSL)
  .getAvailableBandwidthIncludingBorrowing();

//Delay bound
delayBound = tempPath.getPathServiceLevelQoS(bSL).getPacketDelay();

//Loss Rate bound
lossRateBound =
  tempPath.getPathServiceLevelQoS(bSL).getPacketLossRate();

  testMsg("Evaluating path " + currentPathID);
  testMsg("\tAvailable BW:  \t" + availableBW);
  testMsg("\tAvailable BW-B:\t" + availableBWIncludingBorrowing);
  testMsg("\tDelay:         \t" + availableBWIncludingBorrowing);
  testMsg("\tLoss Rate:     \t" + lossRateBound);

//Check if current path fits request (no inter-service borrowing)
if(
  delayBound <= requestedDelay &&
  lossRateBound <= requestedLossRate)
{
  //Current path meets delay and loss rate QoS demands
  //Now check for bandwidth with no borrowing

  testMsg("Meets delay and loss rate QoS demands");

  if(availableBW >= requestedBandwidth)
  {
    //Current path is the first that meets the QoS demands
    //without considering borrowing. Select it and leave path loop
    testMsg("Meets bandwidth requirements" +
      " with no need for interservice borrowing");
    testMsg("** PATH SELECTED ***");
    foundPath = tempPath;
    break stop;
  }

  // If borrowing is allowed and a path with borrowing has not yet
  // been found, check if current path meets bandwidth
  // requirements, now considering inter-service borrowing
  else if(
    isBorrowingAllowed &
    foundPathWithBorrowing == null &&
    availableBWIncludingBorrowing >= requestedBandwidth)
  {
    //Current path is the first that meets the QoS requirements
    //having considered inter-service borrowing. Select it and
    //continue searching for a suitable path with no borrowing
    foundPathWithBorrowing = tempPath;

    testMsg("Meets bandwidth requirements " +
      "only if interservice borrowing is considered");
    testMsg("Continue searching for other paths...");
  }
  else
  {
    testMsg("There is not enough available bandwidth");
  }

} // End of if structure compare
```

```
              testMsg("Done evaluating path.");

            } // End of while-loop through all paths of same hop-count

            testMsg("No more paths of hop count = " + i);

          } // End of if structure

        } // End of for-loop for increasing hop-count

        testMsg("\tAll paths from source to destination evaluated");

      } // End of labeled stop structure


      // Prepare information for display
      String borrowingStatus = (isBorrowingAllowed) ? "enabled" : "disabled";
      String admissionCondition =
        "There was no need for inter-service borrowing";

      // If a path with no borrowing could not be found and borrowing is enabled,
      // return result will either be a path with borrowing or null
      if (foundPath == null &&
          isBorrowingAllowed &&
          foundPathWithBorrowing != null)
      {
        foundPath = foundPathWithBorrowing;
        admissionCondition =
          "Admission possible only with inter-service borrowing";
      }

      display.sendText("\tRouting algorithm results:");

      if(foundPath == null)
      {
        //No path found information
        display.sendText(
          "\t     There are currently no paths available meeting this QoS demand"
          + "\n\t     Inter-service borrowing: " + borrowingStatus);
      }
      else
      {
        //Path found - display information
        display.sendText(
          "\t     The selected path is:        \t" + foundPath.toString()+"\n" +
          "\t     Available bandwidth:         \t" + availableBW + "\n" +
          "\t     Packet delay upper-bound:    \t" + delayBound + "\n" +
          "\t     Packet loss rate upper-bound:\t" + lossRateBound + "\n" +
          "\t     Inter-service borrowing:     \t" + borrowingStatus + "\n" +
          "\t     " + admissionCondition
        );

      } // End if

      return  foundPath;

    } //End of findPathFSP()

  } // End if RoutingALgorithm class

// End of the inner classes definition ----------------------------------------

 /**
  * 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.
    display = new SAAMRouterGui("PathInformationBase");

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

    myServer = theServer;

    vIFacesTraversed = new Vector();

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

    // Launch PIB tester if in Testing Mode
    if(TESTING_MODE) {
      pibTest = new Thread(new PibTester(this));
      pibTest.start();
    }

    // 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
    display.sendText(this.toString());

    //Set default state for interservice borrowing
    setInterserviceBorrowing(INTERSERVICE_BORROWING_DEFAULT);
    setBorrowingThreshold(DEFAULT_BORROWING_THRESHOLD);

  } // End BasePIB constructor()

  /**
   * Resets the PIB, allowing for efficient clearance of server resources
   * allocated for path status maintenance. Wise to be used during
   * initialization of a SAAM server.
   * @return void.
   */
  public void resetPIB( )
```

```
{
  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++)
      {
        // Clear the hashtable stored in the array element
        aPI[i][j][k].clear();
      }
    }
  }

  // Clear each PIB member hashtable
  htRouterIDtoNodeID.clear();
  htNodeIDtoRouterID.clear();
  htRouterInterfaceMap.clear();
  htInterfaces.clear();
  htPaths.clear();
} // End of resetPIB()

/**
 * Returns a String identifying the version of the PIB.
 * @return the String identifying the current PIB version.
 */
public String toString()
{
  return strPIB_VERSION;
}

/**
 * Returns string to display all paths of the PIB.
 * @return the string with all paths of the PIB.
 */
public String toStringPaths()
{
  // Option for displaying only path id and nothe sequence
  final int OPTION = 1;

  return toStringPaths(OPTION);

} // End og displayPaths()

/**
 * Returns a string with all paths currently in the PIB, for displaying
 * purposes. Several options for display are available through the argument
 * provided, so that different detail is obtained.
 * @param option the available display option (-1 to 2).
 * @return String the string representation of the paths.
 */
public String toStringPaths(int option)
{
  String disp = new String();

  disp = "\n***** Paths in the PIB *****\n";
  if(option < -1 || option > 2)
  {
    option = -1;
  }
  if(option == -1)
  {
    disp +=
      "Following display options are available:\n" +
      "    0 : Total number of paths in the PIB\n" +
      "    1 : List all paths and interfaced transversed\n" +
      "    2 : List all paths, interfaces transversed and QoS information\n" +
      "    n : Display only path ID 'n'\n" +
      "   -1 : Display this help information and the total number of paths\n";
  }

  int nrPaths = htPaths.size();
  disp += "\n\tTotal: " + nrPaths;

  if ( ((option == 1) || (option == 2)) && nrPaths > 0 )
  {
    // Step through each path in the PIB
    Enumeration enum = htPaths.elements();
    disp += "\n\tPath-ID\tNode-sequence\n";

    while (enum.hasMoreElements())
    {
      // Get path
      Path path = (Path)enum.nextElement();

      // Display path ID and node sequence
      disp += "\t" + path.toString() + "\n";

      if (option == 2)
      {
        //Option tp display also path QoS information
        PathQoS pqos[] = path.getPathQoSArray();

        disp += "\n\t\tSL \tAvailable BW \tAvailable BW-B \tDelay \tLoss Rate";
        for (int i = 0 ; i < pqos.length ; i++)
        {
          disp += "\n\t";
          disp += "\t"    + i;
          disp += "\t"    + pqos[i].getAvailableBandwidth();
          disp += "\t\t"  + pqos[i].getAvailableBandwidthIncludingBorrowing();
          disp += "\t\t"  + pqos[i].getPacketDelay();
          disp += "\t"    + pqos[i].getPacketLossRate();
        }

        disp += "\n";

      } // End if of option 2

    } // End loop to step through paths

  } // End if for detailed path information

  return disp;

} // End of toStringPaths()

/**
 * Returns a string will all interfaces currently in the PIB, for displaying
 * purposes.
 * @return String the string with all interfaces of the PIB.
 */
public String toStringInterfaces()
{
  // Option for displaying only the interface addresses
  final int OPTION = 0;

  return  toStringInterfaces(OPTION);

} // End of toStringInterfaces()

/**
 * Returns a string will all interfaces currently in the PIB, for displaying
 * purposes. Several options for display are available through the argument
 * provided. Different detail may be displayed.
 * @param int for the option of display (0, 1, 2 or -1)
 * @return String the string representation of the interfaces
 */
public String toStringInterfaces(int option)
{
  String disp = new String();
  InterfaceInfo interfaceIO = null;

  disp = "\n***** Interfaces in the PIB *****\n";
```

```
if(option < -1 || option > 2)
{
  option = -1;
}
if(option == -1)
{
  disp +=
    "   Following display options are available:\n" +
    "      0 : List all interfaces \n" +
    "      1 : List all interfaces and paths\n" +
    "      2 : List all interfaces and their QoS parameters\n" +
    "     -1 : Display this help information and the total no. of paths\n";
}

disp += "\n\tTotal: " + htInterfaces.size() + "\n";

if(option != -1) {

  Enumeration enum = htInterfaces.keys();
  disp += "\nInterface Address\t\tNode ID";

  while (enum.hasMoreElements())
  {
    //IPv6Address interfaceAddress = (IPv6Address) enum.nextElement();
    String addressString = (String) enum.nextElement();
    disp += "\n\t" + addressString;

    interfaceIO =
      (InterfaceInfo) htInterfaces.get(addressString);

    disp += "\t" + interfaceIO.getNodeID();

    if (interfaceIO ==null) {
      disp += "\n\n interfaceIO == null";
      return disp;
    }
    //Display paths transversing this interface (option 1)
    if (option == 1)
    {
      Hashtable table = interfaceIO.getPathIDs();
      Enumeration enumTable = table.elements();

      while (enumTable.hasMoreElements())
      {
        Integer currentPathID =(Integer) enumTable.nextElement();
        disp += "\n\t\tPath ID: " + currentPathID.intValue();
      } // End of pathID display block
    }//end if

    //Display QoS parameters (option 2 or 3)
    if (option == 2)
    {
      disp += "\tTotal BW = " +
        (int)(interfaceIO.getTotalBandwidth()/1000) + " kbps";
      disp += "\tUA BW = " +
        (int)(interfaceIO.getUnallocatedBandwidth()/1000) + " kbps";

      ObsQoS[] qos = interfaceIO.getQoS();

      disp += "\n\n " +
        "SL  Base\tIn use\tAvBW\tAvBW-B\tBC\tUtil\tUtil%\tDelay\tLoss Rate";

      for (byte sl = 0; sl < qos.length; sl++)
      {
        int borrowingCapacity =
          interfaceIO.getServiceLevelBorrowingCapacity(sl);
        int avBWwBorrowing =
          interfaceIO.getServiceLevelAvailableBandwidth(sl);
        if(sl == INT_SERV) {
            avBWwBorrowing +=
              interfaceIO.getServiceLevelBorrowingCapacity(DIFF_SERV);
          }
          if(sl == DIFF_SERV) {
            avBWwBorrowing +=
              interfaceIO.getServiceLevelBorrowingCapacity(INT_SERV);
          }
          String utilPct = String.valueOf(
            Math.abs(qos[sl].getUtilization() * ServiceSA.UTIL_UNIT * 100)
          );

          disp += "\n " +

            //column 1 - service level
            sl + "   " +

            //columnn 2 - base allocation
            ((int)((double)interfaceIO.getTotalBandwidth() / 1000 *
            afBASE_ALLOCATION[sl] * afLOAD_FACTOR[sl])) + " \t" +

            //columnn 3 - current utilization
            (int)((double)qos[sl].getUtilization() * ServiceSA.UTIL_UNIT *
              (double)interfaceIO.getTotalBandwidth() / 1000) + "\t" +

            //columnn 4 - available bandwidth
            (int)(interfaceIO.getServiceLevelAvailableBandwidth(sl) / 1000) +
              "\t" +

            //column 5 - available bandwidth with borrowing
            (int)(avBWwBorrowing / 1000) + "\t" +

            //column 6 - capacity available for borrowing
            (int)(borrowingCapacity / 1000) + "\t" +

            //column 7 - utilization as stored in pib
            qos[sl].getUtilization() + "\t" +

            //column 8 - utilization as percentage of total bandwidth
            (((int)(1000 * (qos[sl].getUtilization() *
              ServiceSA.UTIL_UNIT)))/10) + "\t" +

            //column 9 - delay as stored in pib
            qos[sl].getDelay() + "\t" +

            //columnc 10 - loss rate as stores
            qos[sl].getLossRate();

        } //End of iterface QoS display block

        disp += "\n\n";

      }//End if

    } //End of loop traversing the interfaces hashtable
  }
  return  disp;

} // End toStringInterfaces()

/**
 * Returns a string with the observed QoS parameters of a interface and for a
 * given service level. Appropriate for displaying purposes.
 * @param obs the obsQoS object.
 * @param interfaceAddress the IPv6 address.
 * @param bSL the service level.
 * @return the string wiht observed QoS parameters for display.
 */
public String toStringObservedQoS (
  ObsQoS obs,
  IPv6Address interfaceAddress,
  byte bSL)
```

```
{
  String str =
    "Observed Quality of Service for Interface " +
    interfaceAddress + " Service Level " + bSL + " is: \n";
  str += "Utilization: " + obs.getUtilization() + "\n";
  str += "Delay: " + obs.getDelay() + "\n";
  str += "LossRate: " + obs.getLossRate() + "\n\n";
  return str;
} // End of toStringObservedQoS()

/**
 * Returns a string representation QoS attributes of a given path and for a
 * given service level.
 * @param pathQoS the path QoS object for the given service level.
 * @param pathID the path ID.
 * @param bSL the service level.
 * @return the string representating the QoS attributes of the path for the
 * given servicel level.
 */
public String toStringPathQoS(PathQoS pathQoS, Integer pathID, byte bSL)
{
  String str =
    "Path Quality of Service for Path " +
    pathID.intValue() + " Service Level " + bSL + " is: \n";
  str += "Path Available Bandwidth: " + pathQoS.getAvailableBandwidth() + "\n";
  str += "Delay: " + pathQoS.getPacketDelay() + "\n";
  str += "LossRate: " + pathQoS.getPacketLossRate() + "\n\n";

  return str;
} // End toStringPathQoS()

//[PS] - redesigned to support of inter-service borrowing and to improve
//efficiency
/**
 * Receives and processes a LSA - extracts the vector of ISAs, determines the
 * type of each ISA (Add, Remove or Update) and processes each of them in
 * sequence according to their type.
 * @param LSA a LinkStateAdvertisement object.
 * @return void.
 */
public void processLSA (LinkStateAdvertisement LSA)
{
  // Increase the LSA counter
  iLsaCounter++;
  Vector interfaceSAs = LSA.getInterfaceSAs();
  IPv6Address routerID = LSA.getMyIPv6();

  testMsg("processLSA()");

  boolean isNewRouter = false;
  Integer thisNodeID = (Integer) htRouterIDtoNodeID.get(routerID.toString());

  String strNodeID =
    (thisNodeID == null) ? "new router" : thisNodeID.toString();
  display.sendText(
    "\nProcess LSA number " + iLsaCounter + "\n" +
    "\tRouter / Node ID:  \t" + routerID + "  /  " + strNodeID);

  if (thisNodeID == null) // LSA is from a new NODE
  {
    // If it is a New Node, assign it a new NodeID
    // then place the router in the router/node and node/router look-up tables
    thisNodeID = new Integer(iNextNodeID++);

    htRouterIDtoNodeID.put(routerID.toString(), thisNodeID);
    htNodeIDtoRouterID.put(thisNodeID, routerID);
    isNewRouter = true;
  }// End if for new router detection

  // Step through the list of ISA's and process each according to its type
  Enumeration enumISAs = interfaceSAs.elements();
  while (enumISAs.hasMoreElements())  // Step through each ISA in turn
  {
    testMsg("Start processing new ISA");
    InterfaceSA thisISA  = (InterfaceSA) enumISAs.nextElement();
    byte type = thisISA.getInterfaceSAType();

    // Use simple if structure to determine the type of ISA
    switch(type)
    {
      case InterfaceSA.UPDATE: //Update Interface type

        try
        {
          testMsg("ISA of type update. Going to update interface...");
          updateInterface(thisISA);
        }
        catch (Exception e)
        {
          display.sendText("!!!An exception occurred in updating interface!");
          e.printStackTrace();
        }
        break;

      case InterfaceSA.REMOVE: // Remove Interface type
        try
        {
          testMsg("ISA of type remove. Going to remove interface");
          removeInterface(thisISA);
          displayPIB();
        }
        catch (Exception e)
        {
          display.sendText("!!!An exception occurred in removing interface!");
          e.printStackTrace();
        }
        break;

      case InterfaceSA.ADD: // Add Interface type
        try
        {
          testMsg("ISA of type add. Going to add an interface");
          addInterface(thisISA, thisNodeID, routerID, isNewRouter);
          isNewRouter = false;  //no longer new router for additional interfs
        }
        catch (Exception e)
        {
          display.sendText("!!!An exception occurred in adding interface!");
          e.printStackTrace();
        }
        break;

      default:  // Undefined type of ISA
        display.sendText("Undefined type of ISA.");
    }

  } // End while statement processing vector of ISAs

} // End processLSA()

/**
 * Adds a new interface to the PIB.
 * @param thisISA the ISA object containing the interface description.
 * @param thisNodeID the ID of the node hosting the interface.
 * @param routerID the router ID (IPv6Address) of the hosting rounter.
 * @param isNewRouter whether this is the first interface of the hosting
 * router.
 * @return void.
 */
protected void addInterface(
  InterfaceSA thisISA,
  Integer thisNodeID,
  IPv6Address routerID,
  boolean isNewRouter)
{
  int iBandwidth          = thisISA.getBandwidth();
  byte subnetMask         = thisISA.getInterfaceSubnetMask();
  IPv6Address interfaceID = thisISA.getInterfaceIP();

  testMsg("addinterface()");

  display.sendText("\tAdd interface:\t\t" + interfaceID);

  // Determine if the interface already exists in the Interfaces hashtable
  // -- Can't use interfaceID object directly as key because objects with
  // -- same content may still have different hash codes.

  // Process only new interfaces
  if (!htInterfaces.containsKey(interfaceID.toString()))
  { // If its a new interface, it must have a new information object
    // created to contain its key info:
    //  - host node ID
    //  - total bandwidth capacity
    //  - subnet mask for the network to which it belongs
    //  - and a hashtable to store the path IDs of all paths that traverse it
    InterfaceInfo interfaceInformationObject = new
        InterfaceInfo(thisNodeID, iBandwidth, subnetMask, new Hashtable());

    // Place the new interface in the table of all interfaces
    htInterfaces.put(interfaceID.toString(), interfaceInformationObject);

    if (isNewRouter)
    { //Create an entry into RouterInterfaceMap
      Hashtable htThisNodeInterfaces = new Hashtable();
      htThisNodeInterfaces.put(interfaceID.toString(), interfaceID);
      htRouterInterfaceMap.put(routerID.toString(), htThisNodeInterfaces);

      display.sendText("\t      This is a new router:  \t" + routerID);
    }
    else
    { // Extract the sequence of interfaces hosted on a specific router and
      // append the new interface to the referenced sequence
      ((Hashtable) htRouterInterfaceMap.get(routerID.toString())).
          put(interfaceID.toString(), interfaceID);
    }

    // Determine neighbor interfaces for the interface we are adding.
    // This is accomplished by comparing the network addr of the new interface
    // with the network address of each known interface until a match is found.
    IPv6Address thisNetwork = interfaceID.getNetworkAddress(subnetMask);
    Enumeration enumRouters = htRouterInterfaceMap.elements();

    // Step through each router
    while (enumRouters.hasMoreElements())
    {
      Hashtable NextRouter = (Hashtable) enumRouters.nextElement();
      Enumeration enumInterfaces = NextRouter.elements();

      // Step through each of the candidate router's interfaces
      while (enumInterfaces.hasMoreElements())
      {
        IPv6Address neighborInterfaceID =
          (IPv6Address) enumInterfaces.nextElement();
        IPv6Address nextNetwork =
          neighborInterfaceID.getNetworkAddress(subnetMask);

        // We now compare networkIDs. If they are equal we know interfaces
        // are neighbors and commence updating the aPI, PathIDs, and
        // RouterInterfaceMap hashtable.
        if (nextNetwork.equals(thisNetwork))
        { //Extract the interface object that describes the interface
          InterfaceInfo neighborInfoObj =
              (InterfaceInfo) htInterfaces.get(neighborInterfaceID.toString());
          Integer neighborNodeID = neighborInfoObj.getNodeID();

          display.sendText(
            "\t      Matching subnet ok!\n" +
            "\t      My node ID:        \t" + thisNodeID + "\n" +
            "\t      My neighbor node ID:\t" + neighborNodeID);

          if (!neighborNodeID.equals(thisNodeID))
          {
            // modify the aPI and hashtables to add new paths
            createPaths(
              thisNodeID,
              neighborNodeID,
              interfaceID,
              neighborInterfaceID,
              iBandwidth,
              isNewRouter);
            // exit from while loop assuming point-to-point link;
            // each interface has only one neighbor
            break;
          }
        }// End if-statement processing new subnet pair

      }// End while-statement stepping through a candidate router's interfaces

    }// End while-statement stepping through routers

    displayPIB();

  }
  else
  {
    display.sendText("\t      " +
      "Can't add the interface because it is already in htInterfaces table!");
  } // End if-statement preventing ADD of an existing interface

}// End addInterface();

/**
* Created new paths after adding the new interface to the PIB.
* @param newInterfaceNodeID the IPv6 address ID of the router providing the
* new interface.
* @param neighborNodeID the ID of the neighbor node.
* @param newInterfaceID the IPv6 address of the new interface.
* @param neighborInterfaceID the IPv6 address of the neighbor interface.
* @param iBandwidth the bandwidth of the new interface
* @param boolean whether the interface is the first of the router
* @return void
*/
protected void createPaths (
  Integer newInterfaceNodeID,
  Integer neighborNodeID,
  IPv6Address newInterfaceID,
  IPv6Address neighborInterfaceID,
  int iBandwidth,
  boolean isNewRouter)
{
  // Create one-hop paths between the neighbor nodes
  createOneHopPath (newInterfaceNodeID, neighborNodeID, newInterfaceID,
                    neighborInterfaceID, iBandwidth);

  // Create multi-hop paths eminating from each of the neighbor nodes
  createMultiHopPath (newInterfaceNodeID, neighborNodeID, newInterfaceID,
                      neighborInterfaceID, iBandwidth, isNewRouter);

  // Create combined paths originating and terminating at nodes other than
  // the neighbors, but which include the neighbors
  if (!isNewRouter)
  {
    createCombinedPath (newInterfaceNodeID, neighborNodeID, newInterfaceID,
                        neighborInterfaceID, iBandwidth);
```

```
   }
 } // End of createPaths()

/**
 * Creates a single hop path between two given interfaces.
 * @param newInterfaceNodeID the ID of the node providing the new interface
 * @param neighborNodeID the ID of the neighbor node.
 * @param newInterfaceID the IPv6 address of the new interface.
 * @param neighborInterfaceID the IPv6 address of neighbor interface.
 * @param iBandwidth the bandwidth of the new interface.
 * @return void
 */
 protected void createOneHopPath(
   Integer newInterfaceNodeID,
   Integer neighborNodeID,
   IPv6Address newInterfaceID,
   IPv6Address neighborInterfaceID,
   int iBandwidth)
 {
   // Generate newPathID to aPI Index for intitial route (i.e. A->B)
   short newPathID = iNextPathID++;

   // Append the new path ID to the hashtable in the corresponding aPI element
   aPI[newInterfaceNodeID.intValue()][neighborNodeID.intValue()][1].
       put(new Integer(newPathID), new Integer(newPathID));

   // Create a new Path with the new PathID
   Integer x = new Integer(newPathID);
   aPIIndex y = new aPIIndex(newInterfaceNodeID, neighborNodeID, 1);

   Path newPath =
     new Path(x, y, newInterfaceID, neighborInterfaceID, iBandwidth);

   // Add path to the htPaths hashtable
   htPaths.put(new Integer(newPathID), newPath);

   // Extract interface object for new interface ID
   InterfaceInfo outboundInterfaceObj =
       (InterfaceInfo) htInterfaces.get(newInterfaceID.toString());

   // Add newPathID to the pathID table contained in the htInterfaces hashtable
   // which is contained in the information object for interface newInterfaceID
   Hashtable pathsThruInterface  = outboundInterfaceObj.getPathIDs();
   pathsThruInterface.put(new Integer(newPathID), new Integer(newPathID));

   // Create a path in the opposite direction (the Path constructor updates the
   // necessary tables):

   Path mirrorNewPath = new Path (newPath);

   // [GX-mirror] for single hop paths, no optimization; have to
   // rely on network id matching to find outbound interfaces for
   // mirrored paths
   /*
     mirrorNewPath.setMirror(newPath);
     newPath.setMirror(mirrorNewPath);
   */

 } // End createOneHopPath()

/**
 * Creates a multi-hop path, originating at each of the two interfaces, i.e.,
 * the newly added interface and its neighbor interface in the neighbor node.
 * Appends one interface's node and interface to each path originating at the
 * other interface and going away from the new link.
 * @param newInterfaceNodeID the ID of the node hosting the new interface.
 * @param neighborNodeID the ID of the neighbor node.
 * @param newInterfaceID the IPv6 address of the new interface.
 * @param neighborInterfaceID the IPv6 address of the neigbor interface.
 * @param iBandwidth the interface bandwidth.
 * @param whether the new interface is the first interface of the
 * rounter and so is a new router.
 * @return void.
 */
 protected void createMultiHopPath (
   Integer newInterfaceNodeID,
   Integer neighborNodeID,
   IPv6Address newInterfaceID,
   IPv6Address neighborInterfaceID,
   int iBandwidth,
   boolean isNewRouter)
 {
   // Step 1 - Create path originating at the newInterfaceNode and going
   //          through the neighbor node on the first hop:

   //[GX]: Need to associate a flag with each node id to indicate that it is a
   //live node
   for(  int DestinationIndex = 0;
         DestinationIndex < MAX_NODE_NUMBER;
         DestinationIndex++)
   {
     // Verify that Destination Index is not the same as Neighbor Node or the
     // new interface node.  This prevents process from searching for a path
     // that has a source and destination index as the
     // same. (i.e., aPI[A][A][h])
     if (neighborNodeID.intValue() == DestinationIndex ||
         newInterfaceNodeID.intValue() == DestinationIndex)
     {
       continue;
     }

     for (int hopCount = 1; hopCount < MAX_HOP_COUNT - 1; hopCount++)
     {
       // Extract PathIDs from aPI array containing all Path IDs
       Enumeration enumPathIDs =
         aPI[neighborNodeID.intValue()][DestinationIndex][hopCount].elements();

       while (enumPathIDs.hasMoreElements())  // Step through paths
       {
         // Extract each path based on the pathIDs in the aPI element hashtable
         Integer thisPathID = (Integer) enumPathIDs.nextElement();
         Path thisPath = (Path) htPaths.get(thisPathID);

         // Verify that added node is not already in vNodeSquence which
         // would create a closed loop if added to the sequence.
         if (!thisPath.vNodeSequence.contains(newInterfaceNodeID))
         {
           // Copies references contained in the original node sequence
           // vector(shallow copy).  Does not duplicate the actual sequence.
           // Vector nodeSequence = (Vector) thisPath.vNodeSequence.clone();
           Vector nodeSequence = (Vector) thisPath.getNodeSequence().clone();

           short multiHopPathID = iNextPathID++;  //Create new multiHopPath ID

           // Create the multihop Path
           System.out.println(
             "[createMultiHopPath 1]: path_id = " + multiHopPathID);
           Path multiHopPath =
             new Path( new Integer(multiHopPathID),
                       thisPathID,
                       new aPIIndex(
                           newInterfaceNodeID,
                           new Integer(DestinationIndex),
                           hopCount + 1),
                       newInterfaceID,
                       neighborInterfaceID,
                       iBandwidth);

           // Add Path to htPaths hashtable
           htPaths.put(new Integer(multiHopPathID), multiHopPath);

           // Add pathID to aPI entry for source
           aPI[newInterfaceNodeID.intValue()][DestinationIndex][hopCount + 1].
               put(new Integer (multiHopPathID), new Integer (multiHopPathID));

           // Extract list of interfaces the new path traverses and add the
           // hashtable of path's ID for each interface traversed then replace
           // the table entry
           Enumeration enumInterfacesTraversed =
             multiHopPath.getInterfaceSequence().elements();
           while (enumInterfacesTraversed.hasMoreElements())
           {
             IPv6Address address =
               (IPv6Address) enumInterfacesTraversed.nextElement();
             InterfaceInfo interfaceObject =
                 (InterfaceInfo) htInterfaces.get(address.toString());
             Hashtable pathTable =
               (Hashtable) interfaceObject.getPathIDs();
             pathTable.put(
               new Integer(multiHopPathID),
               new Integer(multiHopPathID));

           } // End loop through interfaces traversed

           // Step 2 - Create a path in the opposite direction (the constructor
           //          updates the necessary tables):

     // [GX-mirror] we can do more efficiently since we can retrieve outbound
     //  interfaces from mirror path of the given subpath

           Path mirrorMultiHopPath = new Path (multiHopPath);

         }// end if statement preventing creation of loop paths

       }// end while for enumPathsIDs

       // Step 3 - Create paths originating at the neighborNode and going
       //          through the newInterfaceNode node on the first hop:
       // Get hashtable of pathIDs for paths eminating from the new interface
       // away from the direction of the neighbor
       if (!isNewRouter)
       {
         Enumeration enumReversePathIDs =
             aPI [newInterfaceNodeID.intValue()]
                 [DestinationIndex]
                 [hopCount].elements();

         while (enumReversePathIDs.hasMoreElements())
         { // Extract each path using the pathID extracted from the hashtable
           Integer thisReversePathID =
             (Integer) enumReversePathIDs.nextElement();
           Path thisReversePath = (Path) htPaths.get(thisReversePathID);

           // Verify that added node is not already in vNodeSquence which
           // would create a closed loop if added to the sequence.
           if (!thisReversePath.vNodeSequence.contains(neighborNodeID))
           { // Copy node references contained in the original node sequence
             // vector(shallow copy).  Does not duplicate the actual sequence.
             Vector reverseNodeSequence =
               (Vector) thisReversePath.vNodeSequence.clone();

             //append added nodeID to vReverseNodeSequence
             reverseNodeSequence.addElement(neighborNodeID);

             //Create new reverseMultiHopPath ID
             short reverseMultiHopPathID = iNextPathID++;

             // Create reverseMultihop Path
             Path reverseMultiHopPath =
               new Path(
                   new Integer(reverseMultiHopPathID),
                   thisReversePathID,
                   new aPIIndex(
                       neighborNodeID,
                       new Integer(DestinationIndex),
                       hopCount + 1),
                   neighborInterfaceID,
                   newInterfaceID,
                   iBandwidth);

             // Add Path to htPaths hashtable
             htPaths.put(
               new Integer(reverseMultiHopPathID),
               reverseMultiHopPath);

             // Add pathID to aPI entry for source
             aPI [neighborNodeID.intValue()]
                 [DestinationIndex]
                 [hopCount + 1].put(
                     new Integer(reverseMultiHopPathID),
                     new Integer(reverseMultiHopPathID));

             // Update each traversed Interface's InterfaceInfo.
             Enumeration enumReverseEffectedInterfaces =
               reverseMultiHopPath.vInterfaceSequence.elements();

             while (enumReverseEffectedInterfaces.hasMoreElements())
             {
               IPv6Address nextAddress =
                   (IPv6Address) enumReverseEffectedInterfaces.nextElement();

               // Extract interface object for nextEffectedInterfaceID
               InterfaceInfo effectedReverseInterfaceObject =
                   (InterfaceInfo) htInterfaces.get(nextAddress.toString());

               // Extract the pathID hashtable for the effected interface
               // and add the new pathID to it
               Hashtable tempReverseEffectedPathIDTable =
                 effectedReverseInterfaceObject.getPathIDs();

               tempReverseEffectedPathIDTable.put(
                   new Integer(reverseMultiHopPathID),
                   new Integer(reverseMultiHopPathID));

             } // End loop to update interface information objects with
               // new path ID

             // Step 4 - Create a path in the opposite direction (the
             //          constructor updates the necessary tables):
             Path MirrorReverseMultiHopPath = new Path ( reverseMultiHopPath );

           } // End if statement preventing closed loop paths

         } // End while for enumReversePathsIDs

       } // End if for new node check

     } // End for loop indexed by hop count

   } // End for loop indexed by Max_Node_Number

 } // End of createMultiHopPath()

/**
 * Creates a combined path by concatenating pairs of paths terminating at
 * respectively the new interface and the neighbor interface.
 * Algorithm:
 *   For any pair of paths such that Path(i) terminates at node(i) and
 *   Path(j) begins at node(j) and does not terminate at node(i),
 *   if no element of Path(i).vNodeSequence is contained in
 *   Path(j).vNodeSequence then create new Path(k) = Path(i) + Path(j)
 * @param newInterfaceNodeID the node ID of the node hosting thenew interface.
 * @param neighborNodeID the node ID of the neighbor node.
 * @param newInterfaceID the IPv6 address of the new interface.
```

```
* @param neighborInterfaceID the IPv6 address of the neighbor interface.
* @param iBandwidth the inteface bandwidth.
* @return void
*/
protected void createCombinedPath (
  Integer newInterfaceNodeID,
  Integer neighborNodeID,
  IPv6Address newInterfaceID,
  IPv6Address neighborInterfaceID,
  int iBandwidth)
{
  // Step 1 - find a pair of candidate paths to be concatenated:
  // For all possible source nodes not equal to the newInterfaceNodeID
  // nor the neighborNodeID:
  for (int SourceIndex = 0; SourceIndex < MAX_NODE_NUMBER; SourceIndex++)
  {
    // Do not consider paths originating at either the new iFace
    // or its neighbor
    if (SourceIndex != newInterfaceNodeID.intValue() &&
      SourceIndex != neighborNodeID.intValue())
    {
      // For all hop counts for the source path:
      for (int SourceHops = 1; SourceHops < MAX_HOP_COUNT - 1; SourceHops++)
      {
        // Extract the aPI element (hashtable of pathIDs)
        // for this source, the new interface, and this hop count

        Hashtable htSourcePaths =
          aPI[SourceIndex][newInterfaceNodeID.intValue()][SourceHops];

        // For all possible destination nodes not equal to either the
        // newInterfaceNodeID or neighborNodeID, and where the combined hop
        // count is less than the maximum so that the addition of the new
        // link will not result in too long of a path:
        for ( int DestinationIndex = 0;
              DestinationIndex < MAX_NODE_NUMBER;
              DestinationIndex++ )
        {
          // Do not consider paths ending at either the new interface
          // or its neighbor
          if (DestinationIndex != newInterfaceNodeID.intValue() &&
              DestinationIndex != neighborNodeID.intValue())
          {
            // For all hop counts for the destination path:
            for ( int DestinationHops = 1;
                  DestinationHops + SourceHops < MAX_HOP_COUNT - 1;
                  DestinationHops++)
            {
              // Extract the pathID hashtables for the candidate destination
              // paths
              // where the aPI indexes are:

              Hashtable htDestinationPaths =
                 aPI  [neighborNodeID.intValue()]
                      [DestinationIndex]
                      [DestinationHops];

              // Traverse the pathID hashtables in pairs such that each
              // potential path combination is considered.  For each pair
              // extract the path pairs from the paths hashtable and
              // extract the node sequences for each path.  Ensure that each
              // node of the second path's vector is not included in the first
              // path's node sequence vector.
              Enumeration enumSourcePathIDs = htSourcePaths.elements();
              while (enumSourcePathIDs.hasMoreElements() )
              {
                Integer iNextSourceCandidate =
                  (Integer) enumSourcePathIDs.nextElement();
                Path pCandidateSource =
                  (Path) htPaths.get(iNextSourceCandidate);
                Enumeration enumDestinationPathIDs =
                  htDestinationPaths.elements();

                while (enumDestinationPathIDs.hasMoreElements())
                {
                  Integer iNextDestinationCandidate =
                    (Integer) enumDestinationPathIDs.nextElement();
                  Path pCandidateDestination =
                    (Path) htPaths.get(iNextDestinationCandidate);

                  // Determine if the candidate paths have any common nodes
                  // by checking each node vector of the candidate source path
                  // for inclusion in the node vector of the candidate
                  // destination path

                  //assume no common node between the path pair
                  boolean boolNoCommonNode = true;
                  Enumeration enumSourcePathNodes =
                    pCandidateSource.getNodeSequence().elements();

                  while (enumSourcePathNodes.hasMoreElements() )
                  {
                    // If a common node is found the source and destination
                    // pair of paths can not be  concatenated without forming
                    // a closed loop.  Thus, if a common node is found do not
                    // continue checking the remaining nodes in the sequence.
                    // Set a flag to ensure a new path is not created for the
                    // path pair.

                    if (pCandidateDestination.getNodeSequence().
                        contains(enumSourcePathNodes.nextElement()) )
                    {
                      boolNoCommonNode = false;
                      break;
                    }
                    // Current path pair contains a common node - procede to
                    // next destination candidate

                  } // End of while loop to test for common nodes

                  // Step 2 - If no common node is found generate a new path
                  // by concatenating the two candidate paths:
                  if (boolNoCommonNode)
                  {
                    Path pConcatenatedPath =
                      new Path (
                        pCandidateSource,
                        pCandidateDestination,
                        newInterfaceID,
                        neighborInterfaceID,
                        iBandwidth );

                    // Add path to the htPaths hashtable
                    Integer iConcatenatedPathID =
                      pConcatenatedPath.getPathID();
                    htPaths.put(iConcatenatedPathID, pConcatenatedPath);

                    System.out.println("[createCombinedPath]: path_id = " +
                      iConcatenatedPathID);

                    // Add pathID to aPI entry for concatenated path:
                    // note that the hop count must include the link between
                    // the two paths
                    aPI [SourceIndex]
                        [DestinationIndex]
                        [SourceHops + DestinationHops + 1].
                        put(iConcatenatedPathID, iConcatenatedPathID);

                    // Add the pathID to each traversed interface's hashtable
                    // of pathIDs
                    Enumeration enumpConcatenatedPathInterfacesTraversed =
                        pConcatenatedPath.getInterfaceSequence().elements();
```

```
                    while (enumpConcatenatedPathInterfacesTraversed.
                      hasMoreElements())
                    {
                      IPv6Address concatAddress =
                        (IPv6Address)enumpConcatenatedPathInterfacesTraversed.
                        nextElement();
                      InterfaceInfo concatObject =
                          (InterfaceInfo)htInterfaces.
                          get(concatAddress.toString());
                      Hashtable concatTable =
                        (Hashtable) concatObject.getPathIDs();

                      concatTable.put(iConcatenatedPathID,iConcatenatedPathID);
                    } // End of while loop to update interface hashtables

                    // Step 3 - Create mirror path from destination to source
                    // (constructor updates necessary tables):
                    Path pMirrorConcatenatedPath = new Path(pConcatenatedPath);

                  } // End if to create concatenated path

                } // End of destination path IDs

              } // End of source path IDs

            } // End destination hop indexed loop

          } // End destination test (IF) for equality of destination node ID
            // to the new interface's node or its neighbor

        } // End destination indexed loop

      } // End source hop indexed loop

    } // End source test (IF) for equality of source node ID to the new
      //interface's node or its neighbor

  } // End source indexed loop for concatenating two paths joined by the newly
    // added interface

} // End of createCombinedPath()

/**
 * Removes an interface from the PIB, given an ISA object.
 * @param thisISA the InterfaceSA object with the information about the
 * interface to be removed.
 * @return void
 */
protected void removeInterface(InterfaceSA thisISA)
{
  testMsg("removeInterface()");
  IPv6Address interfaceAddress = thisISA.getInterfaceIP();

  display.sendText("Removing interface: " + interfaceAddress);

  // Get the InterfaceInformation object from the htInterfaces
  InterfaceInfo interfaceInformation =
      (InterfaceInfo) htInterfaces.get(interfaceAddress.toString());

  // Extract the table of path IDs for paths traversing the interface
  Hashtable interfacePathIDs = interfaceInformation.getPathIDs();
  Enumeration enum = interfacePathIDs.elements();

  // Step through each path in the interface's pathID table
  while (enum.hasMoreElements())
  {
    Integer pathID = (Integer) enum.nextElement();
    // Delete the path from the PIB path table
    Path path = (Path) htPaths.remove(pathID);
    // Extract the node sequence and determine the associated PIB aPI element
    Vector nodeSequence = path.getNodeSequence();

    Integer source = (Integer) nodeSequence.elementAt(nodeSequence.size() - 1);
    Integer destination = (Integer) nodeSequence.elementAt(0);
    int hopCount = nodeSequence.size() - 1;

    // Delete this pathID from aPI[][][]
    Integer ipath =
      (Integer) aPI [source.intValue()]
                    [destination.intValue()]
                    [hopCount].remove(pathID);

    // Get all the other interfaces this path traverses
    Vector interfaceSequence = path.getInterfaceSequence();
    Enumeration enumInterface = interfaceSequence.elements();

    // Step through each interface and delete the path ID from its table
    while (enumInterface.hasMoreElements())
    {
      IPv6Address nextAddress = (IPv6Address) enumInterface.nextElement();
      // Remove the interface object from the hashtable for updating it
      InterfaceInfo interfaceInfo =
          (InterfaceInfo) htInterfaces.get(nextAddress.toString());

      // Extract the path ID hashtable from the interface object
      Hashtable interfacePaths = interfaceInfo.getPathIDs();

      // Delete the pathID from the path ID table and replace the iFace obj.
      Integer x = (Integer) interfacePaths.remove(pathID);

    }//end of while for all the interfaces which the path goes through

  }//end of while for all the paths going through this deleted interface

  // Remove this InterfaceInformation object from the htInterfaces
  InterfaceInfo interfaceInformationRemove =
      (InterfaceInfo) htInterfaces.remove(interfaceAddress);

  display.sendText("Finished removing interface!");

}//end of removeInterface()

// [PS] - redesigned to support inter-service borroiwng and to improve
// efficiency
/**
 * Processes an Interface Status Advertising information (ISA) of type update.
 * Strips the ISA and updates the referenced interface with the new status
 * information retrieved from the one or more ServiceSAs contained in the ISA.
 * @param interfaceSA the InterfaceSA object containing the interface info.
 * @return void.
 */
protected void updateInterface(InterfaceSA interfaceSA)
{
  testMsg("updateInterface()");

  // [PS] June 2001
  // Flag variable used to display message about interface propagation updates
  // after receiving new utilization updates for IntServ or DiffServ. With
  // currentIt SAAM prototype it is possible for the Link State Monitor to
  // advertise 0% utilization, event thought traffic is being routed. While in
  // test mode, because all traffic is virtually generated within the tester,
  // LSA updates for IntServ and DiffServ won't propagate.
  boolean isUtilizationChangedForISorDS = false;
  //</PS>

  //Flag variable - true to flag that current interface has changed
  boolean isInterfaceChanged = false;

  //Local support variables (initalized with value of zero)
  short[] deltaDelay = new short[NUM_OF_SERVICE_LEVELS];
```

```
short[] deltaLossRate = new short[NUM_OF_SERVICE_LEVELS];

//Retrieve general information from this ISA
IPv6Address interfaceAddress = interfaceSA.getInterfaceIP();
byte numberOfSSA = interfaceSA.getNumOfServiceSA();
Vector serviceLevelSAs = interfaceSA.getServiceSAs();

//Retrieve the interface's information object from the interface hashtable
InterfaceInfo interfaceInformation =
  (InterfaceInfo)htInterfaces.get(interfaceAddress.toString());

//Check if the interface indeed exists
if (interfaceInformation == null)
{
  //Interface does not exist in the PIB - display message and return
  display.sendText(
    "\tInterface object not found from htInterfaces hashtable: " +
    "can't process this ISA."
  );
  return;
}

//Interface exists, lets proceed

display.sendText("\tUpdating interface:\t" + interfaceAddress);

//Retrieve array of QoS parameters about the current interface
ObsQoS[] qosArray = interfaceInformation.getQoS();

//Cycle through the SSAs contained in the ISA, one at a time
for (byte i = 0; i < numberOfSSA; i++)
{
  testMsg("Processing SSA # " + (i + 1) + " of " + (numberOfSSA));
  //Retrieve an SSA and its service level
  ServiceSA thisServiceLevelSA = (ServiceSA) serviceLevelSAs.elementAt(i);
  byte bSL = thisServiceLevelSA.getServiceLevel();

  //Currently, we do not handle updates for this service level.
  if (bSL == OUT_PROFILE)
  {
    continue;   //jump over this iteration: no updates for Out of Profile SL
  }

  //Retrieve the metric type
  byte metricType = thisServiceLevelSA.getMetricType();

  switch(metricType)
  {
    case ServiceSA.UTILIZATION_TYPE:                //utilization update

      short oldUtilization = qosArray[bSL].getUtilization();
      short utilization = thisServiceLevelSA.getUtilization();

      if(Math.abs(utilization - oldUtilization) >= thresholdUtilization)
      {
        //Utilization has changed significantly - must update
        isInterfaceChanged = true;
        qosArray[bSL].setUtilization(utilization);
        //new bandwidth calculation
        testMsg("qosArray[" + bSL + "].setUtilization(" + utilization + ")");

        // Set flag to display warning message
        if((bSL == INT_SERV)||(bSL == DIFF_SERV))
        {
          isUtilizationChangedForISorDS = true;
        }

      }
      break;

    case ServiceSA.DELAY_TYPE:                      //delay update
```

```
      short oldDelay =  qosArray[bSL].getDelay();
      short delay = thisServiceLevelSA.getDelay();
      short delayVariation = (short)(delay - oldDelay);

      if(Math.abs(delayVariation) >= thresholdDelay)
      {
        //Delay has changed significantly - must update
        isInterfaceChanged = true;

        qosArray[bSL].setDelay(delay);

        testMsg("qosArray[" + bSL + "].setDelay(" + delay + ") delta = " +
          delayVariation);

        //Keep track of changes for the path delay update
        deltaDelay[bSL] = delayVariation;
      }
      break;

    case ServiceSA.LOSSRATE_TYPE:                   //loss rate update

      short oldLossRate =  qosArray[bSL].getLossRate();
      short lossRate = thisServiceLevelSA.getLossRate();
      short lossRateVariation = (short)(lossRate - oldLossRate);

      if(Math.abs(lossRateVariation) >= thresholdLossRate)
      {
        //Loss Rate has changed significantly - must update
        isInterfaceChanged = true;

        qosArray[bSL].setLossRate(lossRate);

        //Keep track of changes for the path loss rate update
        deltaLossRate[bSL] = lossRateVariation;

        testMsg("qosArray[" + bSL + "].setLossRate(" + lossRate +
          ") delta = " + lossRateVariation);
      }
      break;

    default:                                        //unsupported metric
      display.sendText(
        "[UpdateInterface] Unsupported metric type: " + metricType);

  }//end of switch

}//end of for-loop through each SSA

//Now all SSAs are processed, if the interface has actually changed,
//must evaluate available bandwidth for each SL

//If the interface did not change, no more work to do
if(!isInterfaceChanged)
{
  testMsg("Interface did not change, after processing all SSA of this ISA");
  return;
}

if(isUtilizationChangedForISorDS)
{
  // An ISA was received for updating utilization of IntServ or DiffServ.
  // Must not propagate interface utilization until utilization report
  // contains realistic information.
  display.sendText(
    "PIB RESOURCE MANAGEMENT WARNING:\n" +
    "An ISA changed utilization of IntServ or DiffServ of an Interface.\n" +
    "PIB reservation mechanism may not functionally correctly if routers\n" +
    "advertise inaccurate link utilization. As of Sep 2001, Link \n" +
    "State Monitor was not adversing correct utilization.\n");
}
```

```
  //Refresh the interface QoS parameters
  refreshInterfaceQoS(interfaceInformation);

  //Now the interface has been updated, must update the QoS parameters of
  //all paths transversing this interface

  Hashtable pathIDs = interfaceInformation.getPathIDs();
  Enumeration enumPathIDs = pathIDs.elements();

  testMsg("Update paths transversing changed interface");

  while( enumPathIDs.hasMoreElements() )
  {
    Integer nextPathID = (Integer) enumPathIDs.nextElement();
    Path thisPath = (Path) htPaths.get(nextPathID);

    refreshPathQoS(thisPath, deltaDelay, deltaLossRate);

  } // end while-loop through all paths

display.sendText("Finished updating interface!");

} // End of ()


//[PS] - created to support inter-service borrowing
/**
 * Refreshes the interface QoS data, after processing one or more LSAs which
 * incurred an interface change. This method gets also called from
 * InterfaceInfo constructor and from the interface resetQoS method.
 * @param interfaceIO the InterfaceInfo object.
 * @return void.
 */
protected void refreshInterfaceQoS(InterfaceInfo interfaceIO)
{
  testMsg("refreshInterfaceQoS()");

  // Interface bandwidth
  int totalBandwidth = interfaceIO.getTotalBandwidth();

  // Interface QoS information
  ObsQoS[] qosArray = interfaceIO.getQoS();

  //Determine the portion on the total bandwidth currently not allocated

  int unallocatedBandwidth = totalBandwidth;
  int baseUnallocatedBW = totalBandwidth;
  for(byte sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
    int slBaseAllocation =
      (int)((float)totalBandwidth * afBASE_ALLOCATION[sl]);

    baseUnallocatedBW -= slBaseAllocation;

    int slInUseBandwidth = (int)(
      (double)totalBandwidth *
      (double)qosArray[sl].getUtilization() * ServiceSA.UTIL_UNIT
    );

    switch(sl)
    {
      case INT_SERV:    //Special case (IS) - may grow dynamically
      case DIFF_SERV:   //Special case (DS) - may grow dynamically

        unallocatedBandwidth -= Math.max(slBaseAllocation,slInUseBandwidth);
        break;

      default:
        unallocatedBandwidth -= slBaseAllocation;
    }//end of switch

  }//end of for-loop for determining unallocated bandwidth

  //Ensures unallocated BW is not negative (case with inter-service borrowing)
  unallocatedBandwidth = Math.max(0, unallocatedBandwidth);

  interfaceIO.setUnallocatedBandwidth(unallocatedBandwidth);
  testMsg("interfaceIO.setUnallocatedBandwidth( "+unallocatedBandwidth +" )");

  int borrowingIS = 0;
  int borrowingDS = 0;

  //Determine and set the available bandwidth for each service level
  for(byte sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
    // The initial base allocation
    int slBaseAllocation =
      (int)( (float)totalBandwidth * afBASE_ALLOCATION[sl] );

    // The initial base allocation affected by the SL load factor
    int slTrueBaseAllocation = (int)(
      (float)slBaseAllocation * afLOAD_FACTOR[sl]);

    // The current total service level allocation
    int slInUseBandwidth = (int)(
      (double)totalBandwidth *
      (double)qosArray[sl].getUtilization() * ServiceSA.UTIL_UNIT);

    int slAvailableBandwidth;
    int step1 = 0;
    int step2 = 0;
    int step3 = 0;

    //Differentiate for special cases
    switch(sl)
    {
      case INT_SERV:    //Special case (IS) - may grow dynamically
        step1 = Math.max(0,slTrueBaseAllocation - slInUseBandwidth);
        step2 = unallocatedBandwidth;
        step3 = Math.max(0,slInUseBandwidth
          - baseUnallocatedBW - slTrueBaseAllocation);
        slAvailableBandwidth = step1 + step2 - step3;
        testMsg("step 1 + 2 + 3 : " + step1 + " + " + step2 + " + " + step3);
        borrowingIS = Math.abs(Math.min(0,slAvailableBandwidth));
      break;

      case DIFF_SERV:   //Special case (DS) - may grow dynamically
        step1 = Math.max(0,slTrueBaseAllocation - slInUseBandwidth);
        step2 = unallocatedBandwidth;
        step3 = Math.max(0,slInUseBandwidth
          - baseUnallocatedBW - slTrueBaseAllocation);
        slAvailableBandwidth = step1 + step2 - step3;
        testMsg("step 1 + 2 + 3 : " + step1 + " + " + step2 + " + " + step3);
        borrowingDS = Math.abs(Math.min(0,slAvailableBandwidth));
      break;

      default:          //All other cases (BE, CC and OP)
        slAvailableBandwidth = slTrueBaseAllocation - slInUseBandwidth;
    } // end switch

    // set the new value of available bandwidth
    interfaceIO.setServiceLevelAvailableBandwidth(
      slAvailableBandwidth, sl);
    testMsg("interfaceIO.setServiceLevelAvailableBandwidth( " +
      slAvailableBandwidth + " , " + sl +" )" );

    //Determine and set the new borrowing capacity for this Service Level
    interfaceIO.setServiceLevelBorrowingCapacity(
      borrowingCapacity(slBaseAllocation, slInUseBandwidth, sl), sl);
  }//end of for-loop through each service level for setting available BW

  //Correct available bandwidth if borrowing is taking place - IS
  if(borrowingIS > 0)
  {
    int newSLAvBW =
      interfaceIO.getServiceLevelAvailableBandwidth(DIFF_SERV) - borrowingIS;
    interfaceIO.setServiceLevelAvailableBandwidth(newSLAvBW, DIFF_SERV);
  }

  //Correct available bandwidth if borrowing is taking place - IS
  if(borrowingDS > 0)
  {
    int newSLAvBW = interfaceIO.getServiceLevelAvailableBandwidth(INT_SERV)
      - borrowingDS;
    interfaceIO.setServiceLevelAvailableBandwidth(newSLAvBW, INT_SERV);
  }
} //end of refreshInterfaceQoS()

//[PS] - added to support inter-service borrowing
/**
 * Refreshes the interface bandwidth, after admiting a new flow to a path
 * passing through this interface.
 * @param interfaceIO the interface.
 * @param bandwidthReduction the bandwidth assigned to the flow just admited.
 * @param bSL the service level the bandwidth changes applies to.
 * @return void.
 */
protected void refreshInterfaceBW(
  InterfaceInfo interfaceIO,
  int bandwidthReduction,
  byte bSL)
{
  testMsg("refreshInterfaceBW(" + bandwidthReduction + "," +
    bSL + ")");

  int oldSLBandwidth;
  int newSLAvailableBW = 0;

  // For service levels other than those where Dynamic Growing and Interservice
  // borrowing is allowed (IntServ and DiffServ), simply reduce the available
  // bandwidth by the amount requested
  if(bSL != INT_SERV && bSL != DIFF_SERV)
  {
    oldSLBandwidth = interfaceIO.getServiceLevelAvailableBandwidth(bSL);
    interfaceIO.setServiceLevelAvailableBandwidth(
      oldSLBandwidth - bSL, bSL);
    return;
  }

  // Cases of IntServ or DiffServ (Dynamic Growing and Interservice Borrowing)

  int oldUnallocatedBW = interfaceIO.getUnallocatedBandwidth();
  int oldSLAvailableBW = interfaceIO.getServiceLevelAvailableBandwidth(bSL);
  newSLAvailableBW = oldSLAvailableBW - bandwidthReduction;

  // Set the new service level available bandwidth
  interfaceIO.setServiceLevelAvailableBandwidth(newSLAvailableBW, bSL);
    testMsg("iFaceIO.setServiceLevelAvBW(" + newSLAvailableBW+"," + bSL + ")");

  // Service level base allocation (initial allocated capacity)
  int baseAllocation =
    (int)(afBASE_ALLOCATION[bSL] * (float)interfaceIO.getTotalBandwidth());

  if(newSLAvailableBW > oldUnallocatedBW)
  {
    // Step 1 - direct admission (unallocated space was not yet required)
    // Determine and set the new borrowing capacity
    // To calculate the new borrowing capacity, it is necessary to first
    // determine the SL base allocation and current utilization
    int currentBWUtilization =
      baseAllocation -(newSLAvailableBW - oldUnallocatedBW);

    int bc = borrowingCapacity(baseAllocation, currentBWUtilization, bSL);
    interfaceIO.setServiceLevelBorrowingCapacity(bc, bSL);
    testMsg("Phase 1 (Direct Admission): " +
      "IFace.setServiceLevelBC("+bc+","+bSL+")");
  }
  else
  {
    // Step 2 - dynamic growing (unallocated bandwidth has been claimed) or
    // Step 3 - interservice borrowing

    // Determine and set the new unallocated bandwidth
    int newUnallocatedBW = Math.max(0,oldSLAvailableBW - bandwidthReduction);
    interfaceIO.setUnallocatedBandwidth(newUnallocatedBW);
    testMsg("Phase 2/3 (Dynamic growing/borrowing) " +
      "IFace.setUnallocatedBW(," + newUnallocatedBW + ")");

    // Set the SL borrowing capacity to 0 (zero)
    interfaceIO.setServiceLevelBorrowingCapacity(0, bSL);
    testMsg("Phase 2/3 (Dynamic growing/borrowing) " +
      "IFace.setServiceLevelBC(0," + bSL + ")");

    // Reduce the available BW for the other dynamic SL by the delta UA
    int deltaUA = oldUnallocatedBW - newUnallocatedBW;
    int oldAvBW;
    switch(bSL)
    {
      case INT_SERV:
        oldAvBW = interfaceIO.getServiceLevelAvailableBandwidth(DIFF_SERV);
        interfaceIO.setServiceLevelAvailableBandwidth(
          oldAvBW - deltaUA, DIFF_SERV);
          testMsg("Phase 2/3 (Dynamic growing/borrowing) " +
          "IFace.setServiceLevelAvBW(" + (oldAvBW - deltaUA)+
          "," + DIFF_SERV + ")");
        break;

      case DIFF_SERV:
        oldAvBW = interfaceIO.getServiceLevelAvailableBandwidth(INT_SERV);
        interfaceIO.setServiceLevelAvailableBandwidth(
          oldAvBW - deltaUA, INT_SERV);

          testMsg("Phase 2/3 (Dynamic growing/borrowing) " +
            "IFace.setServiceLevelAvBW(" + (oldAvBW - deltaUA) + "," +
            INT_SERV + ")");

        break;
    }
  }

} // end of refreshInterfaceBW()

/**
 * Refreshes the QoS information of a given path. Whenever a path is newly
 * created, or a changed in one or more of its interfaces happens, the
 * path QoS need to be refreshed so that the path QoS information takes
 * those changes into account.
 * @param path the Path to update.
 * @return void
 */
protected void refreshPathQoS(Path path)
{
  testMsg("refreshPathQoS("+path.getPathID()+")");

  PathQoS pathQoS[] = path.getPathQoSArray();

  short[][] delayAndLossRateArray = findPathDelayAndLossRate(path);
  int[][] pathAvailableBW = pathBandwidth(path);

  //For every Service Level, check the QoS attributes of this Path
  for(byte sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
     //Set path QoS parameters
    pathQoS[sl].setAvailableBandwidth(pathAvailableBW[0][sl]);
    pathQoS[sl].setAvailableBandwidthIncludingBorrowing(pathAvailableBW[1][sl]);
    pathQoS[sl].setDelay(delayAndLossRateArray[0][sl]);
    pathQoS[sl].setLossRate(delayAndLossRateArray[1][sl]);

  }//end for-loop through all service levels

} //end of refreshPathQoS()

// [PS] - created to support sinter-service borrowing
/**
 * Refreshes the QoS information of a given path.
 * @param path the path to update.
 * @param deltaDelay[] an array with the delay variations per Service Level.
 * @param deltaLossRate[] an array with the loss rate variations per SLevel.
 * @return void.
 */
protected void refreshPathQoS(
  Path path,
  short[] deltaDelay,
  short[] deltaLossRate)
{
  testMsg("refreshPathQoS(" + path.getPathID().toString() + ")");

  PathQoS pathQoS[] = path.getPathQoSArray();

  int[][] pathAvailableBW = pathBandwidth(path);

  //For every Service Level, check the QoS attributes of this Path
  for(byte sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
    //Set path QoS parameters
    pathQoS[sl].setAvailableBandwidth(pathAvailableBW[0][sl]);
    pathQoS[sl].setAvailableBandwidthIncludingBorrowing(pathAvailableBW[1][sl]);
    pathQoS[sl].updatePathDelay(deltaDelay[sl]);
    pathQoS[sl].updatePathLossRate(deltaLossRate[sl]);
  }//end for-loop through all service levels

} // end of refreshPathQoS(Path, dDelay, dLR)

//[PS] - created to support inter-service borrowing
/**
 * Calculates the bandwidth capacity that might be released for inter-service
 * borrowing by the specified service level.
 * @param bandwidth an integer representing the bandwidth allocation for
 * the service level (base allocation affected by the load factor).
 * @param utilization the current absolute utilization.
 * @bSL sl service level.
 * @return an integer greater or equal to zero, with the amount of bandwidth
 * that the specified service can make available for borrowing.
 */
protected int borrowingCapacity(int bandwidth, int utilization, byte bSL)
{
  int borrowingCapacity = 0;

  if(bSL == DIFF_SERV || bSL == INT_SERV)
  {
    // This applies the formula of inter-service borrowing and ensures
    // that bandwidth bellow the given threshold is never made available
    // for borrowing.
    // x = sumation(Rf) + offset * sumation(Rf)
    // where x is the amount of bandwidth that cannot be borrowed.
    borrowingCapacity = Math.round(
      afLOAD_FACTOR[bSL] * (float)(bandwidth - Math.max(
        (float)bandwidth * borrowingThreshold,
        (float)utilization * (1.0f + BORROWING_PROBABILITY_OFFSET))));

    // For high levels of utilization or poor settings of the off set factor,
    // it must be ensured the formula does not yield a negative result
    borrowingCapacity = Math.max(0, borrowingCapacity);
  }

  return borrowingCapacity;
}

//[PS] - created to support inter-service borrowing
/**
 * Searches all interfaces traversed by a given path for the minimum
 * available bandwidth of a specified service level. Both values of
 * availability, with and without inter-service borrowing, are calculated and
 * then returned as an array of size 2.
 * @param path the given path.
 * @param bSL the given service level.
 * @return int[] an integer array of size 2, containing respectively
 * the minimum available bandwidth without inter-service borrowing and the
 * minimum available bandwidth including inter-service borrowing.
*/
protected int[] pathBandwidth(Path path, byte bSL)
{
  // Local support variables
  boolean checkInterface;
  int tempBW;

  // Array to hold the result for return. Initialized with 'magic numbers'.
  // The numbers represent the an upper bound for the minimum available BW
  int [] minimumBW = {10000000,10000000};

  Vector interfaceSequence = path.getInterfaceSequence();
  Enumeration e = interfaceSequence.elements();

  // Step through all interfaces traversed by this path
  while (e.hasMoreElements())
  {
    IPv6Address address = (IPv6Address) e.nextElement();
    InterfaceInfo interfaceIO =
        (InterfaceInfo) htInterfaces.get(address.toString());

    // Check to see if a new minimum is found
    tempBW = interfaceIO.getServiceLevelAvailableBandwidth(bSL);
    if (tempBW < minimumBW[0])
    {
      //A new minimum value has been found
      minimumBW[0] = tempBW ;
    }

    // Check to see if a new minimum including borrowing is found
    switch(bSL)
    {
      case INT_SERV:
        tempBW += interfaceIO.getServiceLevelBorrowingCapacity(DIFF_SERV);
        break;

      case DIFF_SERV:
        tempBW += interfaceIO.getServiceLevelBorrowingCapacity(INT_SERV);
        break;

      default:
        // Do nothing
    }

    if (tempBW < minimumBW[1])
    {
      //A new minimum value has been found (including interservice borrowing
      minimumBW[1] = tempBW ;
    }
  }//end of while loop

  return minimumBW;
```

```
}//end of findMinimumAvailableBandwidth(Path, bSL)

//[PS] - created to support inter-service borrowing
/**
 * Searches all interfaces transversed by a given path for the minimum
 * available bandwidth, both with and without interservice borrowing and
 * across all service levels.
 * @param path the given path.
 * @return int[][] a two-dimension array of integers where int[0][i] and
 * int[1][i] refer to the path bandwidth for service level 'i', without and
 * with interservice borrowing, respectively.
*/
protected int[][] pathBandwidth(Path path)
{
  // Local support variables
  boolean checkInterface;
  int tempBW;
  int [][] minimumBW = new int[2][NUM_OF_SERVICE_LEVELS - 1];

  // Array to hold the result for return. Initialized with 'magic numbers'.
  // The numbers represent the an upper bound for the minimum available BW
  for(byte sl = 0; sl < NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
    minimumBW[0][sl] = 10000000;
    minimumBW[1][sl] = 10000000;
  }

  Vector interfaceSequence = path.getInterfaceSequence();
  Enumeration e = interfaceSequence.elements();

  // Ckeck every interface of the path
  while (e.hasMoreElements())
  {
    IPv6Address address = (IPv6Address) e.nextElement();

    InterfaceInfo interfaceIO =
      (InterfaceInfo) htInterfaces.get(address.toString());

    // For this interface, repeat process for every service level
    for(byte bSL = 0; bSL < NUM_OF_SERVICE_LEVELS - 1; bSL++)
    {

      // Check to see if a new minimum is found
      tempBW = interfaceIO.getServiceLevelAvailableBandwidth(bSL);
      if (tempBW < minimumBW[0][bSL])
      {
        //A new minimum value has been found
        minimumBW[0][bSL] = tempBW;
      }

      // Check to see if a new minimum including borrowing is found
      switch(bSL)
      {
        case INT_SERV:
          tempBW += interfaceIO.getServiceLevelBorrowingCapacity(DIFF_SERV);
          break;

        case DIFF_SERV:
          tempBW += interfaceIO.getServiceLevelBorrowingCapacity(INT_SERV);
          break;

        default:
          // Do nothing
      } // end of switch


      if (tempBW < minimumBW[1][bSL])
      {
        //A new minimum value has been found (including interservice borrowing
        minimumBW[1][bSL] = tempBW;
      }
    } // end of the for loop through all service levels

  }//end of while loop

  return minimumBW;

}// end of pathBandwidth(Path)

//[PS] - created to support inter-service borrowing
/**
 * Visits all interfaces transversed by a path to compute the path
 * delay and the loss rate for all service levels.
 * @param path the given Path.
 * @return short [][] a 2-dimension array with the values found, where
 * [0][i] refers to the delay of service level i and [1][i] refers to
 * the loss rate of service level i.
*/
protected short[][] findPathDelayAndLossRate(Path path)
{
  //Local support variables
  short[][] result = new short[2][NUM_OF_SERVICE_LEVELS - 1];

  Vector interfaceSequence = path.getInterfaceSequence();
  Enumeration e = interfaceSequence.elements();

  //Cycle through all interfaces along this path
  while (e.hasMoreElements())
  {
    IPv6Address address = (IPv6Address) e.nextElement();

    InterfaceInfo interfaceIO =
        (InterfaceInfo) htInterfaces.get(address.toString());

    ObsQoS[] qosArray = interfaceIO.getQoS();

    for(int i = 0; i < NUM_OF_SERVICE_LEVELS -1; i++)
    {
      result[0][i] +=  qosArray[i].getDelay();
      result[1][i] +=  qosArray[i].getLossRate();
    }
  }//end of while loop

  return result;

}//end of findPathDelayAndLossRate(path)

//[PS] - created
/**
 * Resets all QoS dynamic information contained in PIB.
 * @return void.
 */
private void resetQoS()
{
  testMsg("resetQoS()");
  int count = 1;
  Enumeration eInterfaces = htInterfaces.elements();
  while(eInterfaces.hasMoreElements())
  {
    InterfaceInfo iface = (InterfaceInfo)eInterfaces.nextElement();
    testMsg("\tInterface # " + (count++));
    iface.resetQoS();
  }
}//end of resetQoS()

//[PS] - reviewd
/**
 * Processes a new flow request. This method is only used by the PIB tester.
 * The method is also used to reset the pib QoS data.
 * @param flowRequest the new FlowRequest. If null is passed, pib QoS data are
 * reset.
 * @return a Vector containing the list of interfaces the flow will
 * traverses or null if the flow cannot be admited.
 */
public Vector processFlowRequest_test(FlowRequest flowRequest)
{
  if(flowRequest == null)
  {
    //Reset pib QoS data and return null
    resetQoS();
    return null;
  }

  vIFacesTraversed.clear();
  processFlowRequest(flowRequest);

  //Must return the routerID of each of the interfaces traversed
  //The return vector will have intercaleted, interface traversed and routerID
  Enumeration e = vIFacesTraversed.elements();
  Vector vResult = new Vector();
  while(e.hasMoreElements())
  {
    //add to result vector, the interface
    IPv6Address iFaceAddress = (IPv6Address)e.nextElement();
    vResult.add(iFaceAddress);

    //add to result vector, the routerID of the interface
    InterfaceInfo ifInfo =
      (InterfaceInfo)htInterfaces.get(iFaceAddress.toString());
    Integer nodeID = ifInfo.getNodeID();
    IPv6Address routerID = (IPv6Address)htNodeIDtoRouterID.get(nodeID);
    vResult.add(routerID);

    //add to result vectot, the interface total bandwidth
    vResult.add(new Integer(ifInfo.getTotalBandwidth()));
  }
  return ((Vector)vResult.clone());
}

//[PS] - redesigned
/**
* Processes a new flow request.
* @param flowRequest the new FlowRequest.
* @return an integer flow label, in which the lower 16 bits carry path-id
* to be used for routing. 0 signifies rejection of flow request.
*/
public int processFlowRequest(FlowRequest flowRequest)
{
  testMsg("processFlowRequest()");
  display.sendText("\nNew flow request arrived to PIB");

  int serviceLevel;
  int flowLabel = 0;

  // Get service level
  serviceLevel = flowRequest.getServiceLevel();

  if(htInterfaces == null)
  {
    // PIB is not ready yet
    display.sendText("\tPIB is not ready. No interfaces are defined!");
  }
  else
  {
    // Select appropriate
    switch(serviceLevel)
    {

      case INT_SERV:
        display.sendText("\tService Level " + serviceLevel + ": IntServ");
        flowLabel =
          admissionControl_IS(flowRequest);
        break;

      case DIFF_SERV:
        display.sendText("\n\tService Level " + serviceLevel + ": DiffServ");
        flowLabel =
          admissionControl_DS(flowRequest);
        break;

      case BEST_EFFORT:
        display.sendText("\n\tService Level " + serviceLevel+": Best Effort");
        flowLabel =
          admissionControl_BE(flowRequest);
        break;

      default:
        display.sendText("\n\tService Level " + serviceLevel+ ": Unsuported");
        flowLabel = 0;

    } // End of switch

  } // end of if PIB not ready

  return flowLabel;

} //End of processFlowRequest

//[PS] - redesigned
/**
 * Checks for the feasibility of a given source node to reach a given
 * destination node, i.e., whether a path between two nodes exists in the PIB.
 * @param sourceNodeID the source node ID.
 * @param destinationNodeID the destination node ID.
 * @return true if it exists a path between the given nodes and false if
 * otherwise.
 */
protected boolean isRouteFeasible(int sourceNodeID, int  destinationNodeID)
{
  //determine whether the source node can reach the destination node
  for (int i = 1; i < MAX_HOP_COUNT ; i++)
  {
    if (aPI[sourceNodeID][destinationNodeID][i] != null)
    {
      return true;
    }
  }//End of for-loop
  return  false;

 } //End of isRouteFeasible()

//[PS] - redesigned
/**
 * Updates the remaining available bandwidth of a given path after admiting
 * a new flow.
 * @param currentPath the path to update.
 * @param requestedBandwidth the requested bandwidth.
 * @param bSL the given service level.
 * @return void.
 */
protected void updateAvailableBandwidth(
  Path currentPath,
  int requestedBandwidth,
  byte bSL)
{
  HashSet changedPaths = new HashSet();
  Integer currentPathID = currentPath.getPathID();

  testMsg("updateAvailableBW(" + currentPathID + "," + requestedBandwidth +
    "," + bSL + ")");
  testMsg("Loop through all iFaces of path" + currentPathID);

  // Change available bandwidth of every  interface traversed by
  // this path (outbound interfaces).
  Enumeration eInterface = currentPath.getInterfaceSequence().elements();

  while (eInterface.hasMoreElements())
  {
    // Retrieve next interface
    IPv6Address iFaceAddress = (IPv6Address) eInterface.nextElement();
    InterfaceInfo interfaceIO =
        (InterfaceInfo) htInterfaces.get(iFaceAddress.toString());

    // If in test mode, add this interface to the list of interfaces this
    // new flow traverses

    if(TESTING_MODE)
    {
      vIFacesTraversed.add(iFaceAddress);
    }

    // Update interface BW to reflect the BW reduction in one service level.
    // Due to interservice borrowing, it may also affect other services BW
    refreshInterfaceBW(interfaceIO, requestedBandwidth, bSL);

    // Must update all paths that traverse this interface, outbound direction
    // For now, I will only flag those paths. When done with all interfaces
    // associated with the path that admited the new flow, then we will
    // refresh all flaged paths
    Hashtable pathIDs = interfaceIO.getPathIDs();
    Enumeration enumPathIDs = pathIDs.elements();

    // Step through all paths that traverse this interface
    while (enumPathIDs.hasMoreElements())
    {
      // Add the path ID (Integer) to the set paths that need to be updated
      changedPaths.add(enumPathIDs.nextElement());
    } // end of while loop through all paths

  } // end of while loop through the interface sequence of a path

  // Now all interfaces traversed by this path have been updates, lets refresh
  // all afected paths, so that they reflec the interface changes
  Iterator pathIDs = changedPaths.iterator();
  while(pathIDs.hasNext())
  {
    // Get next path
    Integer pathID = (Integer)pathIDs.next();
    Path thisPath = (Path) htPaths.get(pathID);
    testMsg("Refreshing path ID " + pathID);

    // Retrieve path QoS array
    PathQoS[] qos = thisPath.getPathQoSArray();

    // Determine and set the new path avaliable bandwidth for this SL
    int[] minAvailableBW = pathBandwidth(thisPath, bSL);
    qos[bSL].setAvailableBandwidth(minAvailableBW[0]);
    qos[bSL].setAvailableBandwidthIncludingBorrowing(minAvailableBW[1]);

    // For service levels with interservice borrowing, must also
    // determine and set any changes in their respective available BW

    switch(bSL)
    {
      case INT_SERV:
        minAvailableBW = pathBandwidth(thisPath, DIFF_SERV);
        qos[DIFF_SERV].setAvailableBandwidth(minAvailableBW[0]);
        qos[DIFF_SERV].setAvailableBandwidthIncludingBorrowing(
          minAvailableBW[1]);
        break;

      case DIFF_SERV:
        minAvailableBW = pathBandwidth(thisPath, DIFF_SERV);
        qos[INT_SERV].setAvailableBandwidth(minAvailableBW[0]);
```

```
        qos[INT_SERV].setAvailableBandwidthIncludingBorrowing(
          minAvailableBW[1]);
        break;

      default:
        // Do nothing if any other service level

    } // end switch

  } // End of while loop through all paths

} // End of updateAvailableBandwidth()


//[PS] - redesiged
/**
 * Performs the admission control sequence for a new IntServ flow request.
 * @param flowRequest the new flow request object.
 * @return the flow label of the new flow. 0 if the flow is rejected.
 */
protected int admissionControl_IS(FlowRequest flowRequest)
{
  testMsg("admissionControl_IS()");

  int flowLabel = 0;
  int flowID;
  FlowResponse response;

  // Retrieve flow request data
  IPv6Address sourceRouterID = flowRequest.getSourceRouterID();
  IPv6Address destinationRouterID = flowRequest.getDestinationRouterID();
  long timeStamp          = flowRequest.getTimeStamp();
  int requestedBandwidth  = flowRequest.getRequestedBandwidth();
  short requestedDelay    = flowRequest.getRequestedDelay();
  short requestedLossRate = flowRequest.getRequestedLossRate();

  // Obtain nodeIDs (int) from the RouterIDs (IPv6Address)
  int sourceNodeID = ((InterfaceInfo)htInterfaces.get(
    sourceRouterID.toString())).getNodeID().intValue();
  int destinationNodeID = ((InterfaceInfo)htInterfaces.get(
    destinationRouterID.toString())).getNodeID().intValue();

  // Display flow request data
  display.sendText(
    "\tFlow requirement information:\n" +
    "\t    Source node:     \t" + sourceNodeID + "\n" +
    "\t    Destination node:\t" + destinationNodeID+ "\n" +
    "\t    Bandwidth:       \t" + requestedBandwidth + "\n" +
    "\t    Delay bound:     \t" + requestedDelay + "\n" +
    "\t    Loss rate bound: \t" + requestedLossRate);

  if( !isRouteFeasible(sourceNodeID, destinationNodeID) )
  {
    // Destination is not reachable from source
    display.sendText("\tDestination is not reachable from source");

    // Generate negative response
    response = new FlowResponse(timeStamp, FlowResponse.REJECTED);
  }
  else
  {
    // Route is feasable

    display.sendText("\tRoute is feasible");

    // Call routing algorithm to select the appropriate path

    Path currentPath = routingAlgorithm.findPath(
                sourceRouterID,
                destinationRouterID,
                requestedBandwidth,
              requestedDelay,
              requestedLossRate,
              INT_SERV,
              isBorrowingEnabled,
              RoutingAlgorithm.FIRST_SHORTEST_PATH);
    if (currentPath != null)
    {
      // A path exists to accomodate this flow request

      // Update affected path and interfaces after flow admission
      updateAvailableBandwidth(currentPath, requestedBandwidth, INT_SERV);

      Integer currentPathID = currentPath.getPathID();

      // assign a flow label to the new connection:
      // flow label = | flow-id (8bits) | path-id (12bits)|
      flowID = currentPath.getNewFlowID();
      flowLabel = flowID * 4096 + currentPathID.intValue(); // 2^12 = 4096

      display.sendText(
        "\tFlow id    \t= " + flowID + "\n" +
        "\tPath id    \t= " + currentPathID.intValue() + "\n" +
        "\tFlow Label \t= " + flowLabel);

     //Add this flow into the ahtFlows object in case rerouting is needed later
      currentPath.AddFlow(
        INT_SERV, flowLabel, new FlowQoS(
                                timeStamp,
                                requestedBandwidth,
                                requestedDelay,
                                requestedLossRate));

      // Send RoutingTableUpdates to routers if path has not yet been created.
      if (!currentPath.bCreated)
      {
        setupPath(currentPath, currentPathID.intValue());
        currentPath.bCreated = true;
      }

      // Generate flow response and send it to the sender
      response =
        new FlowResponse(timeStamp, FlowResponse.IS_ACCEPTED, flowLabel);

    }
    else
    {
      // No path exist that can accomodate this flow request

      display.sendText(
        "\tThere is no path which can support this flow request");

      // Generate flow response and send it to the sender
      response = new FlowResponse(timeStamp, FlowResponse.REJECTED);

    } // end if path created

  } // end of if rounteFeasable

  // Send response back to originator
  myServer.sendFlowResponse(response, sourceRouterID);

  return flowLabel;

} // End of admissionControl_IS()

/**
 * Performs the admission control sequence for a new DiffServ flow request.
 * @param FlowRequest flowRequest.
 * @return the flow label of the new flow. 0 if the flow is rejected.
 */
protected int admissionControl_DS(FlowRequest flowRequest)
```

```
{
  testMsg("admissionControl_DS()");

  int flowLabel = 0;
  FlowResponse response = null;

  // Validate user requesting the flow

  // Get user ID
  int userID = flowRequest.getUserID();

  // Get the Service Level Specification
  SLS userSLS = (SLS) htUserSLSs.get(new Integer(userID));

  // Get flow request data
  IPv6Address sourceRouterID = flowRequest.getSourceRouterID();
  IPv6Address destinationRouterID = flowRequest.getDestinationRouterID();
  long timeStamp = flowRequest.getTimeStamp();

  // Obtain nodeIDs (int) from the RouterIDs (IPv6Address)
  int sourceNodeID = ((InterfaceInfo)htInterfaces.get(
    sourceRouterID.toString())).getNodeID().intValue();
  int destinationNodeID = ((InterfaceInfo)htInterfaces.get(
    destinationRouterID.toString())).getNodeID().intValue();

  // Check to see if the requestor is a valid customer
  if (userSLS == null)
  {
   // Not a valid customer
   display.sendText("DS request is from an invalid customer. Discarding...\n");
  }
  else
  {
    // Valid customer
    // Get the QoS parameter of this user
    int requestedBandwidth  = userSLS.getProfile();
    short requestedDelay    = userSLS.getDelay();
    short requestedLossRate = userSLS.getDelay();

    display.sendText(
      "\tFlow requirement information for DiffServ:\n" +
      "\t    Customer ID:    \t" + userID + "\n" +
      "\t    Source node:     \t" + sourceNodeID + "\n" +
      "\t    Destination node:\t" + destinationNodeID+ "\n" +
      "\t    SLS bandwidth:  \t" + requestedBandwidth + "\n" +
      "\t    SLS delay bound:\t" + requestedDelay + "\n" +
      "\t    SLS loss rate:  \t" + requestedLossRate);

    if( !isRouteFeasible(sourceNodeID, destinationNodeID) )
    {
      // Destination is not reachable from source
      display.sendText("\tDestination is not reachable from source");

      // Generate negative response
      response = new FlowResponse(timeStamp, FlowResponse.REJECTED);
    }
    else
    {
      // Route is feasable

      display.sendText("\tRoute is feasible");

      // Call routing algorithm to select the appropriate path

      Path currentPath = routingAlgorithm.findPath(
        sourceRouterID,
        destinationRouterID,
        requestedBandwidth,
        requestedDelay,
        requestedLossRate,
        DIFF_SERV,
        isBorrowingEnabled,
        RoutingAlgorithm.FIRST_SHORTEST_PATH);

      if (currentPath != null)
      {
        // A path exists to accomodate this flow request
        updateAvailableBandwidth(currentPath, requestedBandwidth, DIFF_SERV);

        Integer currentPathID = currentPath.getPathID();

        // Assign a flow label to the new connection:
        // Flow label = | flow-id | path-id |;
        // in this case, flow-id = 0.
        flowLabel = currentPathID.intValue();

        display.sendText(
          "\tPath id    \t= " + currentPathID.intValue() + "\n" +
          "\tFlow Label \t= " + flowLabel);

      //Add this flow into the ahtFlows object
      FlowQoS flowQoS = new FlowQoS(
        timeStamp,
        requestedBandwidth,
        requestedDelay,
        requestedLossRate);

        currentPath.AddFlow(
          DIFF_SERV, flowLabel, new FlowQoS(
                                  timeStamp,
                                  requestedBandwidth,
                                  requestedDelay,
                                  requestedLossRate));

        // Send RoutingTableUpdates to routers if path has not been created.
        if (!currentPath.bCreated)
        {
          setupPath(currentPath, currentPathID.intValue());
          currentPath.bCreated = true;
        }

        // generate flow response and send it to the sender
        response =
          new FlowResponse(timeStamp,FlowResponse.DS_ACCEPTED,userID,userSLS);
        display.sendText("\tSending positive flow response to DS user");
      }
      else
      {
        display.sendText(
          "\tThere is no path which can support this flow request");

        // Generate flow response
        response = new FlowResponse(timeStamp, FlowResponse.REJECTED);

      } // end of if path created

    } // end of if route is feasable

  }//End else valid customer

  // Send response back to originator
  myServer.sendFlowResponse(response, sourceRouterID);

  return flowLabel;

}//End of admissionControl_DS()

//[PS] - reviewd
/**
 * Performs the admission control sequence for a Best Effort flow request.
 * @param flowRequest the flow request.
 * @return and integer with the flow label, or ) if not admited.
 */
protected int admissionControl_BE(FlowRequest flowRequest)
{
  /* [PS]
    Previous version of PIB had BE implemented in such a way resources
    actually got allocated after admiting a new BE flow, on a fixed 50kbps per
    flow basis. This is contrary to the logic of a the BE service model.
    The whole code was deleted to avoid confusion with current clean resource
    allocation algorithm.
  */
  display.sendText("\n\nNew flow request: Best Effort");

  display.sendText(
    "Best Effort admission control is currently not implemented in PIB.\n" +
    "BE flow request will be rejected!");

  return 0;
} //End of admissionControl_BE()

//[PS] - created to support inter-service borrowing
/**
 * Sets the state of the interservice borrowing capability.
 * @param the new state of the interservice borrowing. True to turn it on and
 * false to turn it off.
 * @return void
 */
public void setInterserviceBorrowing(boolean isBorrowingEnabled)
{
  this.isBorrowingEnabled = isBorrowingEnabled;
  display.sendText("\nInter-service borrowing status: " +
      ((isBorrowingEnabled) ? "ENABLED.\n" : "DISABLED.\n"));
}

//[PS] - created to support inter-service borrowing
/**
 * Sets the borrowing threshold. Inter-service borrowing capable service
 * levels, have a threshold bellow which borrowing is not allowed. Only
 * capacity above this limit is made available for inter-service borrowing;
 */
public void setBorrowingThreshold(double newThreshold)
{
  if((newThreshold >= 0.0d) && (newThreshold <= 1.0d))
  {
    borrowingThreshold = newThreshold;
  }
  display.sendText("\nBorrowing threshold is " +
    Math.round(borrowingThreshold * 100) +
    " % of service level base allocation");
}

/**
 * Sets up a path by sending routing table update messages to routers.
 * @param newPath the new Path.
 * @param pathID the ID of the new path.
 * @return void.
 */
protected void setupPath(Path newPath, int pathID)
{
  display.sendText("Send routing table updates to routers in selected path\n");

  Vector vNodeSeq = newPath.getNodeSequence();
  IPv6Address targetRouterID = null, nextRouterID = null;

  Vector vInterfaceSeq = newPath.getInterfaceSequence();
  IPv6Address nextHopAddress = null;

  nextRouterID =
    (IPv6Address) htNodeIDtoRouterID.get((Integer) vNodeSeq.elementAt(0));
  // Both node and interface sequences are from *destination to source*

  for (int ni = 1; ni < vNodeSeq.size(); ni++)
  {
    // targetRouterID is where this routing entry is going to be intalled
    // the routing entry consists of (path-id, next-hop-interface-address)
    targetRouterID =
      (IPv6Address) htNodeIDtoRouterID.get((Integer) vNodeSeq.elementAt(ni));
    // Have to determine next-hop-interface-address since it is an inbound
    // interface and not part of the vInterfaceSequence of the path

    // First find the outbound interface of the target router
    // Currently vInterfaceSeq contains inbound interfaces as well.
    // Therefore, the code below won't work until vInterfaceSeq is built
    // correctly
    IPv6Address outbound = (IPv6Address) vInterfaceSeq.elementAt(ni - 1);
    //
    //IPv6Address outbound = (IPv6Address) vInterfaceSeq.elementAt(2 * ni - 1);
    //
    InterfaceInfo outboundInfo
        = (InterfaceInfo) htInterfaces.get(outbound.toString());

    byte subnetMaskLength = outboundInfo.getSubnetMask();

    // Then obtain the next hop router ID, and then all interface addresses
    // on that router
    Hashtable nextNodeInterfaces =
        (Hashtable) htRouterInterfaceMap.get(nextRouterID.toString());
    Enumeration e = nextNodeInterfaces.elements();

    // Now search all these addresses to find the inbound interface
    // (with matching network address.
    while (e.hasMoreElements())
    {
      IPv6Address check = (IPv6Address) e.nextElement();
      if (Interface.isOnSameNetwork(check, outbound, subnetMaskLength))
      {
        nextHopAddress = check;
        break;
      }
    }

    //[GX] Seems no need to include the service level info
    //[GX] 1 just for now.
    myServer.sendFRTUpdate(targetRouterID, pathID, nextHopAddress, (byte) 1);

    nextRouterID = targetRouterID;
  }

  // Wait for the routers to update their routing tables
  try
  {
    // 2-second seems arbitrary; alternatively could rely on ACKs.
    Thread.sleep(2000);
  }
  catch(InterruptedException ie)
  {
    display.sendText(
      "Exception while waiting for path set up. " + ie.toString());
  }

} // End of setupPath()

/**
 * Retrieves a Vector with all interfaces of PIB. The Enumerations contains
 * IPv6 addresses as strings for the interfaces.
 * @return the Enumeration with all interfaces in PIB.
 */
public Enumeration getPibInterfaces()
{
  return htInterfaces.elements();
}

/**
```

```
 * Displays to screen the current PIB.
 * @return void.
 */
public void displayPIB()
{
  String str = "";

  if (false) // Set to true to display path information array
  {
    str = "PIB contains following paths:";
    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++)
        {
          // Skip the case of the same source and destination
          if (i == j) {continue;}

          if (!(aPI[i][j][k].isEmpty()))
          {
            str += "From Source: " + i + " to destination: " +
                                    j + " on " + k + " hops\n";
            // Get enumeration of pathIDs for each element of the 3-D array
            // for example aPI[2][3][3] = {4,6,9} --
            // Remember: aPI[i][j][k] is a hashtable!!
            Enumeration enumPathIds = aPI[i][j][k].elements();

            // For each pathID, get the referenced Path and display the
            // vector of nodes
            while (enumPathIds.hasMoreElements()) // Walk through each path
            {
              Integer currentPathID = (Integer) enumPathIds.nextElement();
              // Append the next path ID
              str += "Path: "+ currentPathID.intValue() + "\n ";

              Path currentPath = (Path) htPaths.get(currentPathID);
              if (currentPath == null) {break;}

              if (false) // Set to true to display Path QoS information
              {
                // Extract that path's information
                PathQoS  x[] = currentPath.getPathQoSArray();

                for (int w = 0; w < currentPath.objPathQoS.length; w++)
                {
                  PathQoS test = x[w];
                  str += "Available Bandwidth: " +
                    test.getAvailableBandwidth() + "\n";
                  str += "Delay: " + test.getPacketDelay() + "\n";
                  str += "LossRate: " + test.getPacketLossRate() + "\n\n";
                }

              } // End if for QoS display

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

              }// End if for Node Sequence display


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

              }// End Interface Sequence Display block
              } // End currentPath while loop

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

          } // End hop-count based loop

        } // 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:
    {
      str += toStringPaths(1);
    }

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

    // Dump string to display
    display.sendText(str);

  } // End of displayPIB()

  /**
   * Used during test mode to display additional information to the demo station
   * gui.
   * @param msg the additional information to be displayed.
   * @return void.
   */
  protected void testMsg(String msg) {
    if (DISPLAY_FULL_DETAIL)
    {
      display.sendText("\t\t\t\t\t[TM]  " + msg);
    }
  } // End testMsg()

} // End of PathInformationBase class
```

APPENDIX C PIB TESTER SOURCE CODE

```
//July 2001 [Paulo Silva]   - created

package saam.server;

import java.io.*;
import java.util.*;
import java.awt.*;
import java.text.*;
import java.awt.event.*;
import javax.swing.*;

import saam.agent.server.ServerAgent;
import saam.message.*;
import saam.net.IPv6Address;
import saam.util.jgl.*;

/**
 * PibTester is a test drive for the PIB (BasePIB). This class is instantiated
 * from within BasePIB whenever the test flag of BasePIB is set to true. The
 * tester, has its own GUI interface and may be used for testing or monitoring
 * PIB behavior. Testing means the ability to model generation of IS and DS flow
 * request messages, associated LSA messages and send them to PIB. Monitoring
 * means ability to query PIB about its state, like Path status or Interface
 * status. Simulation results are dumped to a file in column format for
 * firther analysis. Make a search for //columns to discover the column key.
 * @author Paulo Silva (August 2001)
 */
public class PibTester extends JFrame implements ActionListener, Runnable
{

  //Available types of simulation events
  public final byte TYPE_FLOW_REQUEST = 1;
  public final byte TYPE_FLOW_TERMINATION = 2;
  public final byte TYPE_LSA = 3;

  //Support data members
  public final int
    IS        = 0,  //IntServ
    DS        = 1,  //DiffServ
    REQ       = 0,  //Flow requests
    REJ       = 1,  //Flow rejections
    ACT       = 2,  //Active flows
    RJR       = 0,  //Rejection rate
    RRA       = 1,  //Rejection rate average
    EMA       = 2;  //Weighted exponential moving average

  private boolean isReady = false;
  private PrintWriter dataOut = null;
  private String dataLine = null;
  private byte allowBorrowing = 1;
  private int refreshMode = 0;
  protected Thread timer;
  private int numServiceLevels;
  private Object theLock = new Object();
  private BasePIB pib;
  private Hashtable htInterfaces;
  private JglPriorityQueue queue;

  //Simulation variables
  double borrowingThreshold = 0.5d;
  double alphaFactor = 0.5d;
  long seed;
  Random randomGen;
  boolean runSimulation = false;
  boolean isReleased = false;
  boolean abortRun = false;
  private double timeFactor = 1.0d;
  int lsaCycle;
  int flowReqIntervalIS;  //sec
  int durationMeanIS;     //sec
  int durationSigmaIS;    //sec
  int flowBwIS;           //bps
  int flowReqIntervalDS;  //sec
  int durationMeanDS;     //sec
  int durationSigmaDS;    //sec
  int flowBwDS;           //bps
  int eventCounter = 0;
  int projActiveFlowsIS = 0;
  int projActiveFlowsDS = 0;
  final int INT = 3;
  final int DBL = 3;
  final int SIM_DATA = INT + DBL;
  int data[][] = new int[2][INT];
  double rrData[][] = new double[2][DBL];
  int simState = -1;

  //For the moving average
  private boolean [][] req;
  private int [] reqCount = new int[2];
  private int movAvgWindowSize = 0;

  //Gui data
  private String [] modeLabel = {"Display mode: refresh","Display mode: normal"};
  private String [] borrowingLabel = {"Turn borrowing ON","Turn borrowing  OFF"};
  private String [] control = {"S T A R T","S T O P ", "C O N T I N U E"};
  private DecimalFormat df = (DecimalFormat)NumberFormat.getCurrencyInstance();
  private int nrButtons;
  private String text = new String();
  private JLabel lblTimer;
  private JButton [] button;
  private JButton bEvent;
  private JButton bGet;
  private JPanel buttonPanel;
  private JPanel topPanel;
  private JTextArea tArea;
  private JTextField tField;
  private JTextField [] txtField;
  private JLabel dataLabel [][]= new JLabel[2][SIM_DATA];

  final int BUTTON_PANEL_ROWS = 18;

  private String[] bLabel = {
    "L O A D",                //0
    control[0],               //1
    "A B O R T",              //2
    modeLabel[refreshMode],   //3
    "Queue",                  //4
    "Interfaces - Sim",       //5
    "Interfaces - PIB",       //6
    "Paths - PIB",             //7
    borrowingLabel[allowBorrowing], //8
    "Reset PIB QoS",          //9
    "Clear Screen",           //10
    "Exit"};                  //11

  final int
    SIM_LOAD = 0,
    SIM_CONTROL = 1,
    SIM_ABORT = 2,
    SIM_REFRESH_MODE = 3,
    SIM_QUEUE = 4,
    SIM_INTERFACES = 5,
    PIB_INTERFACES = 6,
    PIB_PATHS = 7,
    PIB_BORROWING = 8,
    PIB_RESET = 9,
    SIM_CLS = 10,
    SIM_EXIT = 11,
    INPUT = -1;

  final int
    SIM_DURATION = 0,
    SIM_TIME_FACTOR = 1,
    SIM_LSA_CYCLE = 2,
    SIM_MOV_AVG_WINDOW = 3,
    SIM_ALPHA = 4,
    SIM_SEED = 5,
    SIM_THRESHOLD = 6,
    SIM_FILE_NAME = 7,
    IS_ARRIVALS = 8,
    IS_MU = 9,
    IS_SIGMA = 10,
    IS_BW = 11,
    DS_ARRIVALS = 12,
    DS_MU = 13,
    DS_SIGMA = 14,
    DS_BW = 15,
    STATE_UNLOADED = -1,
    STATE_READY_TO_START = 0,
    STATE_RUNNING = 1,
    STATE_STOPPED = 2;

  /**
   * Constructor.
   * @param pib a reference to the BasePIB class.
   */
  public PibTester(BasePIB pib)
  {
    super("Pib Tester v 3.3 (August 2001)");
    this.pib = pib;
    this.setSize(1000,500);
    initializeGUI();
    queue = new JglPriorityQueue(new GreaterThan());
    htInterfaces = new Hashtable();
    df = (DecimalFormat)NumberFormat.getCurrencyInstance();
    df.setMaximumFractionDigits(2);
    df.applyPattern("#0.00%");
    numServiceLevels = pib.NUM_OF_SERVICE_LEVELS;

  }//end of constructor

  /**
   * Processes simulation events of type flow request.
   * @param the simluation event.
   */
  private void processEvent_FlowRequest(SimulationEvent event)
  {
    boolean isAccepted = false;
    String displayStr = event.toString() + "\t -> ";

    FlowRequest flowRq = (FlowRequest)event.getEventObject();
    byte slIndex = (byte)(flowRq.getServiceLevel() - 1);
    int bandwidth = flowRq.getRequestedBandwidth();
    int duration = (int)(event.getStopTime() - event.getStartTime());

    //For the data log
    dataLine += "," + (slIndex+1)+"," + bandwidth + "," + duration;//columns 4-6

    //Update counter
    data[slIndex][REQ]++;  //Increment number of flow request for this svcLevel

    Vector vInterfaces;
    //Process flow request in PIB and retrieve all interfaces traversed if any.
    synchronized(theLock)
    {
      vInterfaces = pib.processFlowRequest_test(flowRq);
```

```
}

//Check if flow was accepted
if(vInterfaces.size() > 0)
{
  displayStr += "ACCEPTED";
  isAccepted = true;
  dataLine += ",1";   //column 7
  data[slIndex][ACT]++;     //Increment number of active flows

  // Step through all interfaces traversed by the flow
  // Keep track of allocated BW per interface/servicel level

  // Construct a vector of interfaces
  Vector vIFaces = new Vector(1);
  Enumeration eInterfaces = vInterfaces.elements();
  while(eInterfaces.hasMoreElements())
  {
    //Get next element (affected interface)
    String address = ((IPv6Address)eInterfaces.nextElement()).toString();

    //Get next element (routerID)
    String routerID = ((IPv6Address)eInterfaces.nextElement()).toString();

    //Get next element (interface bandwidth)
    int interfaceBW = ((Integer)eInterfaces.nextElement()).intValue();

    vIFaces.add(
      new FlowTerminationData(address,(byte)(slIndex + 1), bandwidth));

    // Add the interfaces to the set of affected interfaces
    if(!htInterfaces.containsKey(address.toString()))
    {
      htInterfaces.put(address, new Interface(routerID, interfaceBW));
    }
    Interface iFace = (Interface)htInterfaces.get(address);
    iFace.increaseAllocatedBW((byte)(slIndex + 1), bandwidth);
  }//end of while

  //Create a flow termination event and insert it in the priority queue
  SimulationEvent simEvent =
    new SimulationEvent(TYPE_FLOW_TERMINATION, event.getStopTime(), vIFaces);
  queue.add(simEvent);

}
else
{
  displayStr += "REJECTED";
  dataLine += ",0";   //column 7
  data[slIndex][REJ]++;

  if(tField.getText().equals("beep"))
  {
    this.getToolkit().beep();
  }
}

//Compute new rejection rate moving average

//Limit the maximum
reqCount[slIndex] +=
  (reqCount[slIndex] == movAvgWindowSize) ? 0 : 1;

boolean last = req[slIndex][reqCount[slIndex] - 1];

//Shift all elements one position
for( int i = reqCount[slIndex] - 1; i >= 1; i--)
{
  req[slIndex][i] = req[slIndex][i-1];
}
req[slIndex][0] = isAccepted; //insert new at front
```

```
  //Average
  int sumRejected = 0;
  for(int i = 0; i < reqCount[slIndex]; i++)
  {
    sumRejected += (!req[slIndex][i]) ? 1 : 0;
  }

  //Store the new refection rate average
  rrData[slIndex][RRA] = (double)sumRejected / (double)reqCount[slIndex];

  //Wheighted exponential moving average
  rrData[slIndex][EMA] =
    alphaFactor * rrData[slIndex][RRA]
    + (1 - alphaFactor) * rrData[slIndex][EMA];

  if(refreshMode == 1)
  {
    synchronized(theLock)
    {
      displayStr += "\n" + pib.toStringInterfaces(2);
    }
    display(displayStr, true);  //Reset display first
  }
  else
  {
    display(displayStr);
  }

}//end of processEvent_FLowRequest

/**
 * Process a generic simulation event.
 * @param event the simulation event.
 * @return void.
 */
private void processEvent(SimulationEvent event)
{
  //dataOuteventCounter
  byte type = event.getType();
  eventCounter++;

  //For the data file: columns 1-3
  dataLine = event.getStartTime() + "," + eventCounter + "," + type;

  switch (event.getType())
  {
    case TYPE_FLOW_REQUEST:
      processEvent_FlowRequest(event);
      break;

    case TYPE_FLOW_TERMINATION:
      processEvent_FlowTermination(event);
      break;

    case TYPE_LSA:
      processEvent_LSA(event);
      //For the data file: columns 4-7
      dataLine += ",,,,";
      break;
  } //end switch

  //refresh gui
  refreshSimOutput();

  //Include for data log, all current statistical information
  //columns 8-19
  dataLine += "," +
    data[IS][REQ] + "," +
    data[IS][REJ] + "," +
    data[IS][ACT] + "," +
    rrData[IS][RJR] + "," +
    rrData[IS][RRA] + "," +
    rrData[IS][EMA] + "," +
    data[DS][REQ] + "," +
    data[DS][REJ] + "," +
    data[DS][ACT] + "," +
    rrData[DS][RJR] + "," +
    rrData[DS][RRA] + "," +
    rrData[DS][EMA];
  dataOut.println(dataLine);
}

/**
 * Process a simulation event of type flow termination.
 * @param event the simulation event. In this case is a vector containing flow
 * termination objects. Each of these object contain data about a single
 * interface.
 * @return void.
 */
private void processEvent_FlowTermination(SimulationEvent event)
{
  String displayStr = event.toString();

  byte serviceLevel = 0;
  int bandwidth = 0;

  //Step through all interfaces affected by the terminated flow
  Enumeration eIFaces = ((Vector)event.getEventObject()).elements();
  while(eIFaces.hasMoreElements())
  {
    //Get the flow termination data for one interface at a time
    FlowTerminationData ftData = (FlowTerminationData)eIFaces.nextElement();
    String address = ftData.getAddress();
    serviceLevel = ftData.getServiceLevel();
    bandwidth = ftData.getBandwidth();

    //Update the allocated bandwidth of this interface
    Interface theInterface = (Interface)htInterfaces.get(address);
    theInterface.decreaseAllocatedBW(serviceLevel, bandwidth);

  }//end while

  //Update counter
  data[serviceLevel-1][ACT]--;

  //Prepare a data line for data output
  dataLine += "," + serviceLevel;   //column 4
  dataLine += "," + bandwidth;      //column 5
  dataLine += ",,";                 //column 6-7

  //Refresh gui display
  if(refreshMode == 1)
  {
    synchronized(theLock)
    {
      displayStr += "\n" + pib.toStringInterfaces(2);
    }
    display(displayStr, true);  //Reset display first
  }
  else
  {
    display(displayStr);
  }
}

/**
 * Processes a simulation event of type LSA.
 * @param event the simulation event.
 * @return void.
 */
private void processEvent_LSA(SimulationEvent event)
```

```
{
  String displayStr = event.toString() + " # "
    + ((Integer)event.getEventObject()).toString();

  //Data initialization
  Hashtable htLSAbyRouterID = new Hashtable();

  //Step through all interfaces that habe been affected by flows
  //and create and store the interface ISA which in turn will contain two SSAs,
  //pertaining to utilization of IS and DS, and the routerID
  Enumeration allInterfaces = htInterfaces.keys();
  while(allInterfaces.hasMoreElements())
  {
    //Retrieve address and the interface object
    String address  = (String)allInterfaces.nextElement();
    Interface thisInterface = (Interface)htInterfaces.get(address);
    String hostRouterID = thisInterface.getRouterID();

    //Create interface status advertisement (ISA)
    InterfaceSA isa = null;
    try
    {
      //Using this constructor, it is an ISA of type update
      isa = new InterfaceSA(IPv6Address.getByName(address));
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }

    //Update IS utilization
    double slAllocatedBW = thisInterface.getAllocatedBW(BasePIB.INT_SERV);
    double iFaceBW = thisInterface.getBandwidth();
    short utilization = (short)(slAllocatedBW/iFaceBW/ServiceSA.UTIL_UNIT);

    isa.insertServiceLevelSA(
      new ServiceSA(BasePIB.INT_SERV,ServiceSA.UTILIZATION_TYPE,utilization));

    //Update DS utilization
    slAllocatedBW = thisInterface.getAllocatedBW(BasePIB.DIFF_SERV);
    utilization = (short)(slAllocatedBW/iFaceBW/ServiceSA.UTIL_UNIT);

    isa.insertServiceLevelSA(
      new ServiceSA(BasePIB.DIFF_SERV,ServiceSA.UTILIZATION_TYPE,utilization));

    //Create a new LSA if no lsa exists for this routerID. Add this ISA to it
    if (htLSAbyRouterID.containsKey(hostRouterID))
    {
      //LSA already created for this router. Retrieve it and add ISA
      LinkStateAdvertisement lsa =
        (LinkStateAdvertisement)htLSAbyRouterID.get(hostRouterID);
      lsa.insertInterfaceSA(isa);
    }
    else
    {
      //First LSA for this router. Create LSA and add ISA
      LinkStateAdvertisement lsa = null;
      try
      {
        lsa = new LinkStateAdvertisement(IPv6Address.getByName(hostRouterID));
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
      lsa.insertInterfaceSA(isa);
      htLSAbyRouterID.put(hostRouterID, lsa);
    }
  }//end while loop to step through all interfaces

  //Step through all LSA's and send them to PIB
  int count = htLSAbyRouterID.size();
  int counter = 1;
  Enumeration eRouters = htLSAbyRouterID.elements();
  synchronized(theLock)
  {
    while (eRouters.hasMoreElements())
    {
      LinkStateAdvertisement lsa =
        (LinkStateAdvertisement)eRouters.nextElement();
      pib.processLSA(lsa);
    }
    displayStr += "\t LSA's: " + count;

    if(refreshMode == 1)
    {
      displayStr += "\n" + pib.toStringInterfaces(2);
      display(displayStr, true);  //Reset display first
    }
    else
    {
      display(displayStr);
    }
  }
}

/**
 * Displays the interfaces currently stored in the tester, i.e. those that
 * have changed due to resource allocation.
 * @return void.
 */
private void displayInterfaces()
{
  int count = 0;
  display("Interfaces in the PibTester (traversed by flows)");
  Enumeration keys = htInterfaces.keys();


  while(keys.hasMoreElements())
  {
    String address = (String)keys.nextElement();
    display(("Interface " + (++count) + ": " + address));
    Interface iFace = (Interface)htInterfaces.get(address);
    for(byte sl = 0; sl < numServiceLevels - 1; sl++)
    {
      display((sl + " - " + iFace.getAllocatedBW(sl)));
    }
  }
}

/**
 * Displays the contents of the simulation priority queue.
 * @return void.
 */
private void displayQueue()
{
  String out = "";
  int count = 0;
  Enumeration e = queue.elements();
  while(e.hasMoreElements())
  {
    SimulationEvent event = (SimulationEvent)e.nextElement();
    out += (++count) + ":\t" + event.toString() + "\n";
  }
  display("Simulation Queue - " + count + " events");
  display(out);
}

/**
 * GUI initialization
 * @return void.
 */
```

```
private void initializeGUI()
{
  nrButtons = bLabel.length;
  button = new JButton[nrButtons];

  Container c = getContentPane();
  c.setLayout(new BorderLayout(5,5));

  //center panel
  tArea = new JTextArea(text,25,30);
  tArea.setAutoscrolls(true);
  tArea.setEditable(false);
  tArea.setBackground(new Color(242,182,132));
  tArea.setFont(new Font("Courier New", Font.BOLD, 12));
  c.add(new JScrollPane(tArea),BorderLayout.CENTER);

  //bottom
  tField = new JTextField(30);
  tField.addActionListener(this);
  tField.setBackground(new Color(198,237,176));
  tField.setFont(new Font("Courier New", Font.BOLD, 12));
  c.add(tField, BorderLayout.SOUTH);

  //right panel - button panel
  buttonPanel = new JPanel(new GridLayout(BUTTON_PANEL_ROWS,1));
  buttonPanel.add(new  JPanel());

  lblTimer = new JLabel(formatTime(0), JLabel.CENTER);

  for(int i = 1; i < BUTTON_PANEL_ROWS - nrButtons - 1; i++)
  {
    buttonPanel.add(new JPanel());
  }

  buttonPanel.add(lblTimer);

  for(int i = 0; i <= nrButtons - 1; i++)
  {
    button[i] = new JButton(bLabel[i]);
    button[i].addActionListener(this);
    buttonPanel.add(button[i]);
  }

  button[SIM_CONTROL].setEnabled(false);
  button[SIM_ABORT].setEnabled(false);
  button[SIM_LOAD].setEnabled(false);

  c.add(buttonPanel,BorderLayout.EAST);

  //left panel
  JPanel p;
  JPanel leftPanel = new JPanel(new GridLayout(0,1));
  txtField = new JTextField[18];
  int i = 0;

  //Row
  leftPanel.add(new JLabel("SIMULATION DATA INPUT",JLabel.CENTER));

  //Row
  p = new JPanel(new GridLayout(1,2));
  p.add(new JLabel("Duration (s) ", JLabel.RIGHT));
  txtField[SIM_DURATION] = new JTextField("3600",5);
  p.add(txtField[SIM_DURATION]);
  leftPanel.add(p);

  //Row
  p = new JPanel(new GridLayout(1,2));
  p.add(new JLabel("Time factor ", JLabel.RIGHT));
  txtField[SIM_TIME_FACTOR] = new JTextField("0.2",5);
  p.add(txtField[SIM_TIME_FACTOR]);
  leftPanel.add(p);
```

```
//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("LSA Cycle (ms) ", JLabel.RIGHT));
txtField[SIM_LSA_CYCLE] = new JTextField("300",5);
p.add(txtField[SIM_LSA_CYCLE]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Moving Avg (# flow reqs)", JLabel.RIGHT));
txtField[SIM_MOV_AVG_WINDOW] = new JTextField("20",5);  //default 10
p.add(txtField[SIM_MOV_AVG_WINDOW]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("EWMA - weigth factor", JLabel.RIGHT));
txtField[SIM_ALPHA] = new JTextField("0.5",5);
p.add(txtField[SIM_ALPHA]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Random no. generator seed", JLabel.RIGHT));
txtField[SIM_SEED] = new JTextField("100",5);
p.add(txtField[SIM_SEED]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Borrowing threshold (%)", JLabel.RIGHT));
txtField[SIM_THRESHOLD] = new JTextField("50",5);
p.add(txtField[SIM_THRESHOLD]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Output file name ", JLabel.RIGHT));
txtField[SIM_FILE_NAME] = new JTextField("SimData_1.txt",5);
p.add(txtField[SIM_FILE_NAME]);
leftPanel.add(p);

//Row
leftPanel.add(new JLabel("INTEGRATED SERVICE FLOWS",JLabel.CENTER));

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Interarrival time (s) ", JLabel.RIGHT));
txtField[IS_ARRIVALS] = new JTextField("1",5);
p.add(txtField[IS_ARRIVALS]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Mean duration (s) ", JLabel.RIGHT));
txtField[IS_MU] = new JTextField("100",5);
p.add(txtField[IS_MU]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Variance (s) ", JLabel.RIGHT));
txtField[IS_SIGMA] = new JTextField("10",5);
p.add(txtField[IS_SIGMA]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Bandwidth (kbps) ", JLabel.RIGHT));
txtField[IS_BW] = new JTextField("6",5);
```

```
p.add(txtField[IS_BW]);
leftPanel.add(p);

//Row
leftPanel.add(new JLabel("DIFFERENTIATED SERVICE FLOWS",JLabel.CENTER));

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Interarrival time (s) ", JLabel.RIGHT));
txtField[DS_ARRIVALS] = new JTextField("10",5);
p.add(txtField[DS_ARRIVALS]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Mean duration (s) ", JLabel.RIGHT));
txtField[DS_MU] = new JTextField("200",5);
p.add(txtField[DS_MU]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Variance (s) ", JLabel.RIGHT));
txtField[DS_SIGMA] = new JTextField("20", 5);
p.add(txtField[DS_SIGMA]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,2));
p.add(new JLabel("Bandwidth (kbps) ", JLabel.RIGHT));
txtField[DS_BW] = new JTextField("7",5);
p.add(txtField[DS_BW]);
leftPanel.add(p);

//Row
leftPanel.add(new JLabel("SIMULATION DATA OUTPUT",JLabel.CENTER));

//Row
p = new JPanel(new GridLayout(1,3));
p.add(new JLabel());
p.add(new JLabel("IntServ",JLabel.CENTER));
p.add(new JLabel("DiffServ",JLabel.CENTER));
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,3));
p.add(new JLabel("Flow requests",JLabel.RIGHT));
dataLabel[IS][REQ] = new JLabel("",JLabel.CENTER);
dataLabel[DS][REQ] = new JLabel("",JLabel.CENTER);
p.add(dataLabel[IS][REQ]);
p.add(dataLabel[DS][REQ]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,3));
p.add(new JLabel("Rejected",JLabel.RIGHT));
dataLabel[IS][REJ] = new JLabel("",JLabel.CENTER);
dataLabel[DS][REJ] = new JLabel("",JLabel.CENTER);
p.add(dataLabel[IS][REJ]);
p.add(dataLabel[DS][REJ]);
leftPanel.add(p);

//Row
p = new JPanel(new GridLayout(1,3));
p.add(new JLabel("Active flows",JLabel.RIGHT));
dataLabel[IS][ACT] = new JLabel("",JLabel.CENTER);
dataLabel[DS][ACT] = new JLabel("",JLabel.CENTER);
p.add(dataLabel[IS][ACT]);
p.add(dataLabel[DS][ACT]);
leftPanel.add(p);
```

```
  //Row
  p = new JPanel(new GridLayout(1,3));
  p.add(new JLabel("Rejection rate",JLabel.RIGHT));
  dataLabel[IS][INT + RJR] = new JLabel("",JLabel.CENTER);
  dataLabel[DS][INT + RJR] = new JLabel("",JLabel.CENTER);
  p.add(dataLabel[IS][INT + RJR]);
  p.add(dataLabel[DS][INT + RJR]);
  leftPanel.add(p);

  //Row
  p = new JPanel(new GridLayout(1,3));
  p.add(new JLabel("Rejection rate avg",JLabel.RIGHT));
  dataLabel[IS][INT + RRA] = new JLabel("",JLabel.CENTER);
  dataLabel[DS][INT + RRA] = new JLabel("",JLabel.CENTER);
  p.add(dataLabel[IS][INT + RRA]);
  p.add(dataLabel[DS][INT + RRA]);
  leftPanel.add(p);

  //Row
  p = new JPanel(new GridLayout(1,3));
  p.add(new JLabel("EWMA",JLabel.RIGHT));
  dataLabel[IS][INT + EMA] = new JLabel("",JLabel.CENTER);
  dataLabel[DS][INT + EMA] = new JLabel("",JLabel.CENTER);
  p.add(dataLabel[IS][INT + EMA]);
  p.add(dataLabel[DS][INT + EMA]);
  leftPanel.add(p);

  c.add(leftPanel,BorderLayout.WEST);

  pack();
  show();
}

/**
 * Displays a string to the GUI interface.
 * @param str the String to be displayed.
 * @return void.
 */
private void display(String str)
{
  tArea.append(str + "\n");
}

/**
 * Displays a string to the GUI interface.
 * @param str the String to be displayed.
 * @param isClearScreen optional clear screen
 * @return void.
 */
private void display(String str, boolean isClearScreen)
{
  if(isClearScreen)
  {
    tArea.setText("\n" + str + "\n");
  }
  else
  {
    tArea.append(str + "\n");
  }
}

/**
 * Formats time for display.
 * @param the time for display in miliseconds.
 * @return a String representation of the provided time.
 */
private String formatTime(long time)
{
  time = Math.round(time/1000);

  long hours = time/3600;
  time -= hours * 3600;
  long minutes = time/60;
  time -= minutes * 60;
  String min = String.valueOf(minutes);
  if (min.length() == 1)
  {
    min = "0" + min;
  }
  String secs = String.valueOf(time);
  if (secs.length() == 1)
  {
    secs = "0" + secs;
  }
  return  ( String.valueOf(hours) + ":" + min + ":" + secs );
}

/**
 * GUI event control.
 * @param e the GUI action event
 * @return void.
 */
public void actionPerformed(ActionEvent e)
{
    int index = -1;
    int option = 0;
    String input;
    Object src = e.getSource();

    //Get the index number of the button pressed (if any)
    if(src.getClass() == button[0].getClass())
    {
      for(int i = 0; i < nrButtons; i++)
      {
        if(src.equals( (Object)button[i] ))
        {
          index = i;
          break;
        }
      }
    }

    switch(index)
    {
      case INPUT:
        input = e.getActionCommand();
        display(input);
        tField.setText("");
        if (input.equals("")) {display("blanks");}
        break;

      case SIM_INTERFACES:
        displayInterfaces();
        break;

      case PIB_INTERFACES:
        option = 0;
        input = tField.getText();
        if( !input.equals("") )
        {
          try
          {
            option = Integer.parseInt(input);
          }
          catch(Exception ex){}
        }

        //Default to enable full detail
        option = 2;

        synchronized(theLock)
        {
```

```
    display(pib.toStringInterfaces(option));
  }
  break;

case PIB_PATHS:
  synchronized(theLock)
  {
    display(pib.toStringPaths(2));
  }
break;

case SIM_QUEUE:
  displayQueue();
  break;

case SIM_CLS:
  tArea.setText("Cleared");
  break;

case SIM_LOAD:
  if(simState == STATE_UNLOADED || simState == STATE_READY_TO_START)
  {
    simLoad();
    simState = STATE_READY_TO_START;
    button[SIM_CONTROL].setText(control[simState]);

    button[SIM_CONTROL].setEnabled(true);
    button[SIM_ABORT].setEnabled(false);

    runSimulation = false;
    isReleased = false;
    abortRun = false;
  }
  else
  {
    display("Simulation is running");
    simState = STATE_UNLOADED;
  }
break;

case SIM_ABORT:
  synchronized(theLock)
  {
    runSimulation = false;
    isReleased = true;
    abortRun = true;
  }

  try{
    Thread.sleep(1000);
  }
  catch(Exception ex){}

  //Reset local set of affected interfaces and send an LSA to PIB
  Enumeration enum = htInterfaces.elements();
  while(enum.hasMoreElements())
  {
    Interface iFace = (Interface)enum.nextElement();
    iFace.reset();
  }
  processEvent_LSA(new SimulationEvent(TYPE_LSA, 0, new Integer(0)));

  button[SIM_LOAD].setEnabled(true);
  button[SIM_ABORT].setEnabled(false);
  button[SIM_CONTROL].setEnabled(false);

break;

case SIM_CONTROL:

  switch(simState)
  {
    case STATE_READY_TO_START:  //0
      runSimulation = true;
      simStart();
      simState = STATE_RUNNING;
      button[SIM_LOAD].setEnabled(false);
      button[SIM_CONTROL].setEnabled(true);
      button[SIM_CONTROL].setText(control[simState]);
      button[SIM_ABORT].setEnabled(true);
    break;

    case STATE_RUNNING:           //1
      synchronized(theLock)
      {
        runSimulation = false;
        isReleased = false;
      }
      simState = STATE_STOPPED;
      button[SIM_CONTROL].setText(control[simState]);
    break;

    case STATE_STOPPED:           //2
      simState = STATE_RUNNING;
      synchronized(theLock)
      {
        timeFactor =
          Double.parseDouble(txtField[SIM_TIME_FACTOR].getText());
        isReleased = true;
        abortRun = false;
        runSimulation = true;
      }
      button[SIM_CONTROL].setText(control[simState]);
    break;

  }
break;

case PIB_RESET:
  synchronized(theLock)
  {
    pib.processFlowRequest_test(null);
  }
break;

case SIM_REFRESH_MODE:
  refreshMode = Math.abs(refreshMode - 1);  //toggle refreshMode
  button[SIM_REFRESH_MODE].setText(modeLabel[refreshMode]);
break;

case PIB_BORROWING:
  synchronized(theLock)
  {
    if(allowBorrowing == 0)
    {
      //Turn interservice borrowing on
      allowBorrowing = 1;
      pib.setInterserviceBorrowing(true);
    }
    else
    {
      //Turn interservice borrowing off
      allowBorrowing = 0;
      pib.setInterserviceBorrowing(false);
    }
  }
  button[PIB_BORROWING].setText(borrowingLabel[allowBorrowing]);
break;

case SIM_EXIT:
  System.exit(0);
  break;
```

```
      default:
        display("No action implemented");

    }//end switch

}//end actioPerformed()

/**
 * Opens a file to output simulation results.
 * @return boolean whether the operation succeded.
 */
public boolean fileOpen()
{
  boolean isOpen = false;

  //First, ensure previous file is closed
  fileClose();

  //Open the file
  try
  {
    dataOut = new PrintWriter(
      new FileOutputStream(txtField[SIM_FILE_NAME].getText()), true);
    isOpen = true;
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
  return isOpen;
}

/**
 * Closes current opened file.
 * @return void.
 */
public void fileClose()
{
  try
  {
    dataOut.close();
  }
  catch(Exception e){}
}

/**
 * Simulation timer thread. Controls the timing od the simulation events
 * when the simulation is running.
 * @return void.
 */
public void run()
{
  long simTime = 0;
  long sleepTime = 0;
  SimulationEvent event;
  long eventTime = 0;
  long lastEventTime = 0;

  stopThread:
  while(!queue.isEmpty())
  {
    //Dequeue next event
    event = (SimulationEvent)queue.pop();
    lastEventTime = eventTime;
    eventTime = event.getStartTime();

    long timeInterval =
      (long)((Math.abs(eventTime - lastEventTime)) * timeFactor);

    //wait until event is due
    lblTimer.setText(formatTime(lastEventTime) +
      " - " + String.valueOf(lastEventTime/1000));
    while(timeInterval > 0)
    {
      sleepTime = 1000;
      if (timeInterval < sleepTime)
      {
        sleepTime = timeInterval;
      }
      timeInterval -= sleepTime;

      try
      {
        Thread.sleep(sleepTime);
        simTime += sleepTime;
        lblTimer.setText(formatTime(lastEventTime) +
          " - " + String.valueOf(lastEventTime/1000));
        boolean isStopped;
        synchronized(theLock)
        {
          isStopped = !runSimulation;
        }
        if(isStopped)
        {
          //Now loop and wait until isRelease is true
          boolean loop = true;
          do
          {
            Thread.sleep(1000);
            synchronized(theLock)
            {
              loop = !isReleased;
            }
          }while(loop);
          //isRelease is true, lets find what next (stop or continue)

          if(abortRun)
          {
            //abort simulation
            break stopThread;
          }
          //continue running the simulation
        }
      }
      catch(Exception e){}
    }

    processEvent(event);

  }//end while queue is not empty

  //Queue is empty or abort button was pressed

  if(isReady)
  {
    simStop();
  }
  else
  {
    //wait until pib is ready
    display("\nSAAM is not ready! Waiting... \n");
    boolean loop = true;
    long count = 0;
    do
    {
      lblTimer.setText(formatTime(count));
      //Check SAAM is
      try
      {
        createFlowRequest_IS(1,2,10,10,10);
        isReady = true;
```

```
        display("\nSAAM is ready! Waiting time: " + formatTime(count), true);
        tArea.setBackground(new Color(198,237,176));
      }
      catch(Exception e){}
      if(isReady)
      {
        loop = false;
        button[SIM_LOAD].setEnabled(true);
      }
      else
      {
        try
        {
          Thread.sleep(1000);
          count += 1000;
        }
        catch(Exception e){}
      }
    }while(loop);
  }
}

/**
 * Generates normal variates in accordance with the Rejection Method.
 * @param mu the mean value of the normal distribution.
 * @param sigma the sigma value of the normal distribution.
 * @return the value of the normal variated generated.
 */
private double normalVariate(double mu, double sigma)
{
  //Calculate a normal variate with the rejection method
  double u1, u2, u3, x;

  do{
    u1 = randomGen.nextDouble();
    u2 = randomGen.nextDouble();
    x = -Math.log(u1);
  }
  while(u2 > Math.exp( -(x-1)*(x-1)/2.0d ));
  u3 = randomGen.nextDouble();
  if(u3 > 0.5d)
  {
    return mu + sigma * x;
  }
  else
  {
    return mu - sigma * x;
  }
}

/**
 * Loads the queue with the specified flow request events.
 * @param simDuration the duration of the simulation in seconds.
 * @param svcLevel the service level of the flow request.
 * @param interval the mean value of time between flow requests (seconds).
 * @param durationMean the mean value for the flow duration (seconds).
 * @param durationSigma the sigma of the duration distribution.
 * @param bandwidth the value of the bandwidh of each flow (bps).
 * @return the total number of flows generated.
 */
private int loadFlowRequests(long simDuration, byte svcLevel,
  int interval, int durationMean, int durationSigma, int bandwidth)
{
  int nrFlows = 0;
  double randNormal;
  double interArrivalTime;
  double lambda;            //for the interarrival rate of flow requests
  long flowDuration = 0;    //for the flow duration (ms)
  long simTime = 0;         //for the simulation time (ms)
  //Normalize units
  simDuration = simDuration * 1000; //duration of the simulation in miliseconds
  //Calculate arrival rate of new flow requests
  lambda = 1.0d / ((double)(interval));  //requests per second
  while(simTime <= simDuration)
  {
    //Increment counter
    nrFlows++;

    //Interarrival time of next flow request (exponetial distribution)
    //x = -1/lambda * ln(u) where 0 <= u < 1
    interArrivalTime = (-1.0d/lambda * Math.log(randomGen.nextDouble())); //sec

    //Advance simulation time by the interarrival time
    simTime += (long)(interArrivalTime * 1000);

    //Generate new flow only if begin of new flow does not exceed simDuration
    if(simTime <= simDuration)
    {
      //Duration in miliseconds (normal distribution)
      do{
        flowDuration =
          (int)(1000 * normalVariate((double)durationMean,
            (double)durationSigma));
      }while(flowDuration < 1000);

      //Create and add the new flow request event
      if(svcLevel == BasePIB.INT_SERV)
      {
        queue.add(new SimulationEvent(
          TYPE_FLOW_REQUEST,
          simTime,
          simTime + flowDuration,
          createFlowRequest_IS(1,2,0,0,bandwidth)));
      }
      else if(svcLevel == BasePIB.DIFF_SERV)
      {
        queue.add(new SimulationEvent(
          TYPE_FLOW_REQUEST,
          simTime,
          simTime + flowDuration,
          createFlowRequest_DS(1,2,0,0,bandwidth)));
      }
    }//end if
  } //end while
  return nrFlows;
}

/**
 * Loads the simulation queue with the required LSA events.
 * @param simDuration the duration of the simulation in seconds.
 * @param lsaCycle the LSA interva in miliseconds.
 * @return int the number of LSA events generated.
 */
private int loadLSA(int simDuration, int lsaCycle)
{
  //Advance dataline for the data log output
  dataLine += ",,,,"; //columns 5-8

  //Calculate the number of LSA cycles
  int cycles = (int)(simDuration*1000/lsaCycle) + 1;

  //Add LSA events
  for(int i = 1; i <= cycles; i++)
  {
    queue.add(new SimulationEvent(TYPE_LSA, lsaCycle * i, new Integer(i)));
  }
  return cycles;
}

/**
 * GUI support function. Retrieves the integer number of the text field.
 * @param txt the text field.
 * @return the number retrieved.
 */
private int getValue(JTextField txt)
{
  return Integer.parseInt(txt.getText());
}

/**
 * Loads a new simulation with current GUI values.
 * @return void.
 */
private void simLoad()
{
  display("\nLoading a new simulation...");

  //Reset simulation
  pib.processFlowRequest_test(null);  //Reset PIB
  display("LSA transmitted to reset PIB QoS data");

  reqCount[IS] = 0;
  reqCount[DS] = 0;
  req = new boolean[2][movAvgWindowSize];  //For the moving average
  for (int i = 0; i < INT; i++)
  {
    data[DS][i] = 0;
    data[IS][i] = 0;
  }
  for (int i = 0; i < DBL; i++)
  {
    rrData[DS][i] = 0;
    rrData[IS][i] = 0;
  }
  refreshSimOutput();
  queue.clear();
  eventCounter = 0;

  htInterfaces.clear();
  refreshSimOutput();

  //Simulation data
  double d = Double.parseDouble(txtField[SIM_ALPHA].getText());
  borrowingThreshold =
    (double)Integer.parseInt(txtField[SIM_THRESHOLD].getText())/100.0d;
  pib.setBorrowingThreshold(borrowingThreshold);
  int simDuration   = Integer.parseInt(txtField[SIM_DURATION].getText());//sec
  movAvgWindowSize  = Integer.parseInt(txtField[SIM_MOV_AVG_WINDOW].getText());
  lsaCycle          = Integer.parseInt(txtField[SIM_LSA_CYCLE].getText());//ms
  seed              = Long.parseLong(txtField[SIM_SEED].getText());
  alphaFactor       = Double.parseDouble(txtField[SIM_ALPHA].getText());
  timeFactor        = Double.parseDouble(txtField[SIM_TIME_FACTOR].getText());

  randomGen = new Random(seed);


  //IntServ flow charaterization
  flowReqIntervalIS = Integer.parseInt(txtField[IS_ARRIVALS].getText());//sec
  durationMeanIS = Integer.parseInt(txtField[IS_MU].getText());   //sec
  durationSigmaIS = Integer.parseInt(txtField[IS_SIGMA].getText());     //sec
  flowBwIS = 1000 * Integer.parseInt(txtField[IS_BW].getText());        //bps

  //DiffServ flow charaterization
  flowReqIntervalDS = Integer.parseInt(txtField[DS_ARRIVALS].getText());//sec
  durationMeanDS = Integer.parseInt(txtField[DS_MU].getText());   //sec
  durationSigmaDS = Integer.parseInt(txtField[DS_SIGMA].getText());     //sec
  flowBwDS = 1000 * Integer.parseInt(txtField[DS_BW].getText());        //bps

  //Load IS flow request events
  int nFlowReqs = loadFlowRequests(simDuration, BasePIB.INT_SERV,
    flowReqIntervalIS, durationMeanIS, durationSigmaIS, flowBwIS);
  display("Queue loaded with " + nFlowReqs + " IS flow requests");
  //Load DS flow request events
  nFlowReqs = loadFlowRequests(simDuration, BasePIB.DIFF_SERV,
    flowReqIntervalDS, durationMeanDS, durationSigmaDS, flowBwDS);
  display("Queue loaded with " + nFlowReqs + " DS flow requests");

  //Load LSA events
  int nLSAs = loadLSA(simDuration, lsaCycle);
  display("Queue loaded with " + nLSAs + " LSA events");

  //Display simulation projections
  projActiveFlowsIS = Math.round(durationMeanIS / flowReqIntervalIS);
  projActiveFlowsDS = Math.round(durationMeanDS / flowReqIntervalDS);

  display("\nProjected network load demand (no rejections):\n");
  display("\tSL\tActive flows  \tBandwidth (kbps)\n");
  display("\tIS\t" + projActiveFlowsIS + "\t\t" +
    (Math.round(durationMeanIS / flowReqIntervalIS) * flowBwIS / 1000));
  display("\tDS\t" + projActiveFlowsDS + "\t\t" +
    (Math.round(durationMeanDS / flowReqIntervalDS) * flowBwDS / 1000));
  display("\nSimulation is ready to start");

}

/**
 * Refreshes GUI.
 * @return void.
 */
private void refreshSimOutput()
{
  //Refresh the rejection rate
  if(data[IS][REQ] > 0)
  {
    rrData[IS][RJR] = (double)data[IS][REJ]/(double)data[IS][REQ];
  }
  if(data[DS][REQ] > 0)
  {
    rrData[DS][RJR] = (double)data[DS][REJ]/(double)data[DS][REQ];
  }

  //Refresh gui with current data
  for(int i = 0; i < INT; i++)
  {
    dataLabel[IS][i].setText(String.valueOf(data[IS][i]));
    dataLabel[DS][i].setText(String.valueOf(data[DS][i]));
  }

    dataLabel[IS][ACT].setText(dataLabel[IS][ACT].getText() +
      " ["+ projActiveFlowsIS + "]");
    dataLabel[DS][ACT].setText(dataLabel[DS][ACT].getText() +
      " ["+ projActiveFlowsDS + "]");

  for(int i = 0; i < DBL; i++)
  {
    dataLabel[IS][INT + i].setText( df.format(rrData[IS][i]));
    dataLabel[DS][INT + i].setText( df.format(rrData[DS][i]));
  }
}

/**
 * Generates a single IS Flow Request message.
 * @param srcNodeID the source node ID
 * @param destNodeID the destination node ID
 * @param delay the flow delay bound
 * @param lossRate the loss rate bound
 * @param bandwidth the bandwidth bound
 * @return the IS flow request message
 */
private FlowRequest createFlowRequest_IS(
  int srcNodeID,
  int destNodeID,
  int delay,
  int lossRate,
  int bandwidth)
{
  return new FlowRequest(
    (IPv6Address)pib.htNodeIDtoRouterID.get(new Integer(srcNodeID)),
    (IPv6Address)pib.htNodeIDtoRouterID.get(new Integer(destNodeID)),
    System.currentTimeMillis(),
    (short)delay, (short)lossRate, bandwidth);
}

/**
 * Generates a single DS Flow Request messages.
 * @param srcNodeID the source node ID
 * @param destNodeID the destination node ID
 * @param delay the flow delay bound
 * @param lossRate the loss rate bound
 * @param bandwidth the bandwidth bound
 * @return the DS flow request message
 */
private FlowRequest createFlowRequest_DS(
  int srcNodeID,
  int destNodeID,
  int delay,
  int lossRate,
  int bandwidth)
{
  int userID = 6;

  return new FlowRequest(
    (IPv6Address)pib.htNodeIDtoRouterID.get(new Integer(srcNodeID)),
    (IPv6Address)pib.htNodeIDtoRouterID.get(new Integer(destNodeID)),
    System.currentTimeMillis(),
    userID, (short)delay, (short)lossRate, bandwidth);
}

/**
 * Stops a running simulation.
 * @return void.
 */
private void simStop()
{
  display("Simulation run terminated!!");
  fileClose();
  simState = -1;
  button[SIM_LOAD].setEnabled(true);
  button[SIM_CONTROL].setEnabled(false);
}

/**
 * Starts a simulation.
 * @return void.
 */
private void simStart()
{
  display("\nStarting a new simulation");
  if(fileOpen())
  {
    //Write to file data of this simulation run
    dataOut.println("lsa," + lsaCycle);                        //ms
    dataOut.println("timeFactor," + timeFactor);
    dataOut.println("EMA - weight," + alphaFactor);
    dataOut.println("seed," + seed);                           //ms
    dataOut.println("movAvgWindowSize," + movAvgWindowSize);  //flow reqs
    dataOut.println("borrowingThreshold," + borrowingThreshold);
    dataOut.println("is_arrivals," + flowReqIntervalIS);      //sec
    dataOut.println("is_meanDuration," + durationMeanIS);     //sec
    dataOut.println("is_sigmaDuration," + durationSigmaIS);   //sec
    dataOut.println("is_bw," + flowBwIS);                      //bps
    dataOut.println("ds_arrivals," + flowReqIntervalDS);      //sec
    dataOut.println("ds_meandDuration," + durationMeanDS);     //sec
    dataOut.println("ds_sigmaDuration," + durationSigmaDS);   //sec
    dataOut.println("ds_bw," + flowBwDS);                      //bps
    dataOut.println("borrowing," + allowBorrowing);
    dataOut.println("projActiveFlowsIS," + projActiveFlowsIS);
    dataOut.println("projActiveFlowsDS," + projActiveFlowsDS);

    int colCount = 0;
    for(colCount = 1; colCount < 19; colCount++)
    {
      dataOut.print(colCount + ",");
    }
    dataOut.println(colCount);

    //Start timer
    timer = new Thread(this);
    timer.start();
  }
  else
  {
    display("\n");
    display(" *********************************************************");
    display(" *  W A R N I N G : Could not open file for data output     *");
    display(" *********************************************************");
  }
}

/**
 * This class is used to define simulation events.
 * @author Paulo Silva (August 2001)
 */
private class SimulationEvent
{
  private byte eventType;
  private long startTime;
  private long stopTime;
  private Object eventObject;
  private String [] typeLabel = {"Flow request", "Flow termination", "LSA"};

  /**
   * Constructs a generic simulation event.
   * @param eventType the type of event.
   * @param the event start time.
   * @param the event stop time.
   * @param anyObject the event object to be associated with this event.
   */
  public SimulationEvent(byte eventType, long startTime, long stopTime, Object anyObject)
  {
    this.eventType = eventType;
    this.startTime = startTime;
    this.stopTime = stopTime;
    this.eventObject = anyObject;
  }

  /**
   * Constructs a flow request event.
   * @param eventType the type of event.
   * @param the event start time.
   * @param anyObject the event object to be associated with this event.
   */
  public SimulationEvent(byte eventType, long startTime, Object anyObject)
  {
    this.eventType = eventType;
    this.startTime = startTime;
    this.stopTime = startTime;
    this.eventObject = anyObject;
  }

  /**
   * Retrieves the event type.
   * @return the type of event
   */
  public byte getType()
  {
    return eventType;
  }

  /**
   * Retrieves the start time of the event.
   * @return the start time of the event.
   */
  public long getStartTime()
  {
    return startTime;
  }

  /**
   * Retrieves the stop time of the event.
   * @param the stop time of the event.
   */
  public long getStopTime()
  {
    return stopTime;
  }

  /**
   * Retrieves the event object.
   * @param the event object
   */
  public Object getEventObject()
  {
    return eventObject;
  }

  /**
   * String representation of this event.
   * @return a string representation of this event.
   */
  public String toString()
  {
    String out = "  " + formatTime(startTime) + " - " + typeLabel[eventType-1];

    if (eventType == TYPE_FLOW_REQUEST)
    {
      byte srvLevel = ((FlowRequest)eventObject).getServiceLevel();
      String svc = (srvLevel == 1) ? "IS" : "DS";
      int duration = (int)((stopTime - startTime)/1000);
      out += " [" + svc + "] duration = " + duration + " secs";
    }
    return out;
  }
}// end of class SimulationEvent

/**
 * This class defines a flow termination data object. This object is used to
 * associated single flow data with a single interface. A flow termination
 * event is made of one or more of these objects.
 * @author Paulo Silva (August 2001)
 */
private class FlowTerminationData
{
  private String address;
  private byte serviceLevel;
  private int bandwidth;

  /**
   * Constructor.
   * @param address the IPv6 address of the interface.
   * @param serviceLevel the service level.
   * @param bandwidth the bandwidth assigned to the terminated flow.
   */
  public FlowTerminationData(String address, byte serviceLevel, int bandwidth)
  {
    this.address = address;
    this.serviceLevel = serviceLevel;
    this.bandwidth = bandwidth;
  }

  /**
   * Retrieves the address.
   * @return the address.
   */
  public String getAddress()
  {
    return address;
  }

  /**
   * Retrieves the service level.
   * @return the service level.
   */
  public byte getServiceLevel()
  {
    return serviceLevel;
  }

  /**
   * Retrieves the bandwidth.
   * @return the bandwidth.
   */
  public int getBandwidth()
  {
    return bandwidth;
  }

} // end of class FlowTerminationData

//--GreaterThen---------------------------------------------------------------
/**
 * This class implements the method required by the priority queue to
 * give its ordering behaviot.
 * @author Paulo Silva (August 2001)
 */
private class GreaterThan implements BinaryPredicate
{
  public boolean execute(Object obj1, Object obj2)
  {
    boolean result = true;
    long first = ((SimulationEvent)obj1).getStartTime();
    long second = ((SimulationEvent)obj2).getStartTime();

    if (first < second)
      {
        result = false;
      }
    return result;
  }
}// end of class BynaryPredicate

/**
 * This class defines an Interface object to be used only within the tester.
 * @author Paulo Silva (August 2001)
 */
private class Interface
{
  private int[] slAllocatedBW;
  private String routerID;
  private int bandwidth;

  /**
   * Constructor.
   * @param routerID the routerID.
   * @param interfaceBandwidth the total bandwidth of the interface.
   */
  public Interface(String routerID, int interfaceBandwidth)
```

```
{
  slAllocatedBW =  new int[pib.NUM_OF_SERVICE_LEVELS - 1];

  for (byte sl = 0; sl < pib.NUM_OF_SERVICE_LEVELS - 1; sl++)
  {
    slAllocatedBW[sl] = 0;
  }
  setRouterID(routerID);
  setBandwidth(interfaceBandwidth);
}

/**
 * Increases the allocated bandwidth of a service level by the given
 * amount.
 * @param serviceLevel the servicel level.
 * @param bandwidth the bandwidth allocation increase.
 * @return void.
 */
public void increaseAllocatedBW(byte serviceLevel, int bandwidth)
{
  slAllocatedBW[serviceLevel] += bandwidth;
}

/**
 * Sets a new bandwidth allocation value.
 * @param serviceLevel the servicel level.
 * @param bandwidth the bandwidth allocation increase.
 * @return void.
 */
public void setAllocation(byte serviceLevel, int bandwidth)
{
  slAllocatedBW[serviceLevel] = bandwidth;
}

/**
 * Sets a new value for the router ID.
 * @param routerID the IPv6 based value for the router ID.
 * @return void.
 */
public void setRouterID(String routerID)
{
  this.routerID = routerID;
}

/**
 * Retrieves the router ID value.
 * @return the router ID.
 */
public String getRouterID()
{
  return routerID;
}

/**
 * Sets the new value for the total bandwidth of the interface.
 * @param newBandwidth the new value of the bandwidth.
 * @return void.
 */
public void setBandwidth(int newBandwidth)
{
  bandwidth = newBandwidth;
}

/**
 * Retrieves the interface total bandwidth.
 * @return the interface bandwidth.
 */
public int getBandwidth()
{
  return bandwidth;
}
```

```
    /**
     * Resets the allocated bandwith.
     * @return void.
     */
    public void reset()
    {
      for(int i = 0; i < 3; i++)
      {
        slAllocatedBW[i] = 0;
      }
    }

    /**
     * Decreases the allocated bandwidth.
     * @param serviceLevel the service level.
     * @param bandwidth the amount of bandwidth decrease.
     */
    public void decreaseAllocatedBW(byte serviceLevel, int bandwidth)
    {
      slAllocatedBW[serviceLevel] -= bandwidth;
    }

    public int getAllocatedBW(byte serviceLevel)
    {
      return slAllocatedBW[serviceLevel];
    }
  }//end of Interface class

}//end of BasePIBTester class
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D TEST AND EVALUATION DATA

Te following table summarizes the data obtained from 40 simulation runs.

		Inter-Service Borrowing		Difference	Change
		Disabled	Enabled		

Simulation Run A

IntServ	Flow Requests	723	723	0	-36 %
	Flow Rejections	37	23	-13.5	
	Flow Rejections (%)	5.0	3.2	-1.8	
	Active Flows (avg)	91.3	93.4	2.1	
	Aggregated Throughput	547 847	560 531	12 684	
DiffServ	Flow Requests	79	79	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	20.8	20.8	0	
	Aggregated Throughput	145 758	145 833	75	

Simulation Run B

IntServ	Flow Requests	712	723	11	-35 %
	Flow Rejections	54	35	-18.5	
	Flow Rejections (%)	7.4	4.8	-2.6	
	Active Flows (avg)	92.9	96.1	3.2	
	Aggregated Throughput	557 417	576 664	19 247	
DiffServ	Flow Requests	76	79	2.5	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	21.1	21.1	0.1	
	Aggregated Throughput	147 374	147 800	426	

		Inter-Service Borrowing		Difference	Change
		Disabled	Enabled		

Simulation Run C

IntServ	Flow Requests	704	712	8	
	Flow Rejections	62	46	-16	
	Flow Rejections (%)	8.6	6.3	-2.3	-27%
	Active Flows (avg)	93.9	96.8	2.9	
	Aggregated Throughput	563 421	580 787	17 366	
DiffServ	Flow Requests	78	76	-1.3	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	20.7	21.1	0.4	
	Aggregated Throughput	145 021	147 743	2 722	

Simulation Run D

IntServ	Flow Requests	719	719	0	
	Flow Rejections	80	55	-25	
	Flow Rejections (%)	11.1	7.6	-3.5	-31%
	Active Flows (avg)	94.7	99.4	4.7	
	Aggregated Throughput	568 411	596 332	27 921	
DiffServ	Flow Requests	76	76	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	19.9	19.9	0	
	Aggregated Throughput	139 572	139 588	16	

		Inter-Service Borrowing		Difference	Change
		Disabled	Enabled		

Simulation Run E

IntServ	Flow Requests	719	719	0	-28%
	Flow Rejections	98	71	-27.0	
	Flow Rejections (%)	13.6	9.9	-3.8	
	Active Flows (avg)	95.6	100.1	4.5	
	Aggregated Throughput	573 737	600 503	26 766	
DiffServ	Flow Requests	76	76	0	
	Flow Rejections	0	0	0	
	Flow Rejections (%)	0	0	0	
	Active Flows (avg)	19.9	19.9	0	
	Aggregated Throughput	139 572	139 583	11	

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Shelton, General Henry H. – "Joint Vision 2020" – Approved by the Chairman of the Joint Chiefs of Staff JV2020, http://www.dtic.mil/jv2020/, June 2000.

[2] Gibson, John H. and Dao-Cheng, Kuo, "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", Computer Science Department, Naval Postgraduate School, Monterey, September 2000.

[3] Quek, Henry C., "QoS management with adaptive routing for next generation Internet", Computer Science Department, Naval Postgraduate School, Monterey, March 2000.

[4] Braden, R., Clark, D., and Shenker, S., "Integrated Services in the Internet architecture: an overview", RFC 1633, June 1994.

[5] Floyd, Sally, "Link-sharing and resource management models for packet networks, September 13, 1993.

[6] Devore, Jay L., "Probability and Statistics for Engineering and the Sciences", Fourth edition, Duxbury 1998

[7] Vrable, Dean J. and Yarger, John W., "The SAAM architecture: enabling integrated services", Computer Science Department, Naval Postgraduate School, Monterey, September 1999.

[8] Akkoc, Hasan, "A pro-active routing protocol for configuration of signaling channels in Server and Agent based Active network Management", Computer Science Department, Naval Postgraduate School, Monterey, June 2000.

[9] Kati, Efraim, "Fault-tolerant approach for deploying Server and Agent based Active network Management (SAAM) server in Windows NT environment to provide uninterrupted services in routers in case of server failures(s)", Computer Science Department, Naval Postgraduate School, Monterey, March 2000.

[10] Szcepankiewicz, Peter & Velazquez, Luis, "Authentication in SAAM routers", Computer Science Department, Naval Postgraduate School, Monterey, June 2000.

[11] Jain, Raj, "The Art of Computer Systems Performance Analysis", John Wiley & Sons, Inc., 1991.

[12] Cao, Zhiruo, Wang, Zheng, Zegura, Ellen, "Rainbow Queueing: Fair Bandwidth Sharing Without Per-Flow State".

[13] Kurose, James F. and Ross, Keith W, "Computer Networking", Addison Wesley, 1999

[14] Stoica, Ion and Zhang, Hui, "Providing Guaranteed Services Without Per Flow Management", May 1999.

[15] Stoica, Ion, Shenker, Scott., Zhang, Hui, “Core-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks”, Sept 1998.

[16] Xie, Geoffrey G. and Lam, Simon L., “An Efficient Adaptive Search Algorithm for Scheduling Real-Time Traffic”, in Proceeding of International Conference of Network Protocols, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA 22060-6218

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

3. Chairman, Code CS
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   cseagle@cs.nps.navy.mil

4. Direcção do Serviço de Formação
   Marinha - Portugal
   dsf@mail.marinha.pt

5. Portuguese Naval Attaché
   Washington D.C.
   ponavnlr@mindspring.com

6. Dr. Mari W. Maeda
   Program Manager – DARPA/ITO
   Arlington, Virginia
   mmaeda@darpa.mil

7. Dr. Geoffrey Xie
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   xie@cs.nps.navy.mil

8. Dr. Bert Lundy
   Computer Science Department, Code CS
   Naval Postgraduate School
   Monterey, California
   blundy@cs.nps.navy.mil

9. Mr. Cary Colwell
   Naval Postgraduate School
   Monterey, California
   colwell@cs.nps.navy.mil

10. LCDR Mónica de Oliveira
    Escola Naval
    pmonica@mail.telepac.pt

11. LCDR Angel Sanjose
    Spanish Navy
    Monterey, California
    aesanjose@hotmail.com

12. LCDR Leonardo da Silva Mattos
    Brazilian Navy
    leonardo_mattos@hotmail.com

13. LT António S. Monteiro
    Naval Postgraduate School
    Monterey, California
    asmontei@nps.navy.mil

14. LT António S. Martinho
    Naval Postgraduate School
    Monterey, California
    martinho@pacbell.net

15. LT Eduardo Bolas
    Naval Postgraduate School
    Monterey, California
    ejbolas@nps.navy.mil

16. LCDR Paulo R. Silva
    Portuguese Navy
    pjrsilva@yahoo.com