MASTER'S THESIS

Architecture, Design, Simulation and Performance Evaluation for Implementing ALAX -- The ATM LAN Access Switch Integrating the IEEE 1355 Serial Bus

by G. Charleston
Advisor: A. Makowski

CSHCN M.S. 97-5
(ISR M.S. 97-10)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

ABSTRACT

Title of Thesis: ARCHITECTURE, DESIGN, SIMULATION AND PERFORMANCE EVALUATION FOR IMPLEMENTING ALAX- THE ATM LAN ACCESS SWITCH INTEGRATING THE IEEE 1355 SERIAL BUS

Degree candidate: Giles Colbert Charleston

Degree and year: Master of Science, 1997

Thesis directed by: Dr. Armand M. Makowski
Department of Electrical Engineering and
Institute for Systems Research

IEEE 1355 is a serial bus standard for Heterogeneous Inter Connect (HIC) developed for "enabling high-performance, scalable, modular and parallel systems to be built with low system integration cost". However to date, few systems have been built around this standard specification. In this thesis, we propose ALAX- an internetworking switching device based on IEEE 1355.

The aim of the thesis is two-fold. First, we discuss and summarize research works leading to the architecture, design and simulation development for ALAX; we synthesize and analyze relevant data collected from the simulation experiments of the 4-port model of ALAX (i.e., 4-by-4 with 4 input and output queues)- these activities were conducted during the 2-year length of the project. Secondly, we expand the original 4-by-4 size of the ALAX simulation model into 8-, 12- and 16-port models and, present

and interpret the outcomes. Thus, overall we establish a performance assessment of the ALAX switch, and also identify several critical design measurements to support the ALAX prototype implementation.

We review progresses made in Local Area Networks (LANs) where traditional software-enabled bridges or routers are being replaced in many instances by hardware-enabled switches to enhance network performance. Within that context, the ATM (Asynchronous Transfer Mode) technology emerges as an alternative for the next-generation of high-speed LANs. Hence, ALAX incarnates our effective approach to build an ATM-LAN interface using a suitable switching platform. ALAX provides currently the capability to conveniently interconnect legacy Ethernet and ATM-based networks. Its distributed architecture features a multi-processor environment of T9000 transputers with parallel processing capability, a 32-by-32 way nonblocking crossbar fabric (C104 chipset) partitioned into Transport (i.e., Data) and Control planes, and many other modules interlaced with IEEE 1355-based connectors. It also employs existing and emerging protocols such as LANE (LAN Emulation), IEEE 802.3 and SNMP (Simple Network Management Protocol). We provide the component breakdown of the ALAX simulation model based on the Optimized Network Engineering Tools (OPNET)[1]. The critical parameters for the study are acceptable processor speeds and queuing sizes of shared memory buffer at each switch port. The performance metric used is the end-to-end packet delay. Finally, we end the thesis with conclusive recommendations pertaining to the performance and design measurement, and a brief summary of other possible areas for further research study.

[1]Simulation tool developed by MIL3.

ARCHITECTURE, DESIGN, SIMULATION AND PERFORMANCE EVALUATION FOR IMPLEMENTING ALAX- THE ATM LAN ACCESS SWITCH INTEGRATING THE IEEE 1355 SERIAL BUS

by

Giles Colbert Charleston

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Master of Science
1997

Advisory Committee:

Dr. Armand M. Makowski, Chairman/Advisor
Dr. Prakash Narayan
Dr. Charles B. Silio

PREFACE

This thesis is intended specifically as a "bible" for the ALAX research project, which focused on the development of an ATM LAN access switch based on the IEEE 1355 serial bus standard for Heterogeneous Interconnect (HIC). Its purpose is to 1) provide the basic background material 2) lay the groundwork that led to the networking architecture and design embedded in the ALAX switch and 3) compile the different facets of research studies that spawn off from this initiative and are briefly discussed in some other technical reports [2]. The thesis will hence serve as a working reference for researchers or any interested parties dealing with the design and simulation of an internetworking device or aspiring to seek novel application ventures for the IEEE 1355 standard pushed by Inmos SGS-Thomson.

The research project in question lasted 2 years at the University of Maryland at College Park. It involved several key activities notably, participation in various standardization seminars, technology workshops, technical discussions, design development, building many simulation model components and foremost conducting extensive simulation experiments. An effective research team was set up to carry on the different tasks required. So in essence, the contents of this document do not reflect solely my inputs, but primarily major contributions from this team, followed then by my complementary research investigations in relation to these preceding works. Thus credits should be given for :

[2] These documents are identified as references 22,23,24,25,26, 35 and 36.

1. the ALAX architecture and design-
   (a) to Dr. J. Kim of ETRI and
   (b) to Mr. Man-Geun Ryu of Modacom
2. the ALAX simulation modeling and experimentation activities-
   (a) to Mr. Tom Christofili (specifically for creating parts of the T9000 component),
   (b) to Mr. Sandeep Rao (specifically for creating parts of the T9000 component and most of the traffic model generators),
   (c) to Mr. Rajarshi Gupta (specifically for creating parts of the T9000 component and some of the traffic model generators),
   (d) to Dr. J. Kim of ETRI (specifically for creating parts of the C104 and the T9000 components), and
   (e) to the team in general for building the entire simulation model and running the required simulations.
3. the oversight and leadership of this research interest-
   (a) to Drs. A. Makowski and P. Narayan

During these 2 years, I have collaborated in the achievements of the project's goals, first through my early involvement on simulation development for creating the C104 component of the ALAX switch, secondly in conducting some of the experiments and then by participating in the workshops and discussions which took place. I also developed and created a web page for the project (http://www.isr.umd.edu/Labs/LAST/indexlast.html). And I prepared a poster for the Research Project Display sponsored by the National Science Foundation.

Henceforth, the first three chapters of this thesis provide the introductory concepts and also the background material related to the LAN and the ATM networking worlds, as found throughout the literature. Then, Chapter 4[3] discusses the architecture and design of the ALAX switch. In the fifth chapter[4], I describe the components of the ALAX simulation model based on OPNET. Chapter 6[5], reports, synthesizes and analyzes the large set of data collected for the 4-by-4 model of ALAX (i.e., with 4 input and output queues) during the 2 years of this project. Further in Chapter 7[6], I provide more data and interpretations for a second set of data involving other configurations of ALAX. These configurations include 8-port (8-by-8), 12-port (12-by-12) and 16-port (16-by-16) models developed from expanding the original 4-port model. Finally in Chapter 8, I conclude the thesis with some recommended parameters for the implementation of ALAX. I also look into some other possibilities for enhancing the ALAX architecture, as they surfaced during the course of the project, and later on when I carried on the expansion and simulation tasks from the original 4-by-4 ALAX model.

[3]This chapter describes mainly contributions from Dr. J. Kim and Mr. Man-Geun Ryu.

[4]This chapter summarizes works of Mrs. Giles Charleston, Tom Christofili, Rajarshi Gupta and Sandeep Rao, and also Dr. J. Kim.

[5]The simulations reported here were ran by Mrs. Giles Charleston, Tom Christofili, Rajarshi Gupta and Sandeep Rao.

[6]The simulations reported here were ran by Giles Charleston.

DEDICATION

To my Mother Junie- she has been the living symbol of courage, strength and dedication in my life, and an inspiration throughout the years of hardship.

To Tante Suzel- She is the second mother that I have, a source of wisdom and my protector during my early years in this world.

To my beloved Nathalie- she has always been here to give me hope, support and encouragement. When I needed an ear to listen to me and someone to talk to about the difficulties I was experiencing, she always made the time. She is my soul mate and, the positive spirit that brings happiness to me and makes me realize that tomorrow will always be better.

To my Brothers and Cousins, Elides, Boulou, Youri and Wesley- You have been in my mind all along the way guys, and I dedicate this piece of work to you also.

Love and Affection from the bottom of my heart.

ACKNOWLEDGEMENTS

I would like to thank the Supreme Being for implanting seeds of wisdom, dedication, patience and courage in me throughout my years in school and especially during these Graduate Studies.

I am glad to express my indebtness to the ALAX/LAST research team including Drs. Makowski, Narayan and Kim, Tom Christofili, Sandeep Rao and Rajarshi Gupta; I really enjoyed working with all of you. Without your direct collaborative efforts and initiatives on this project this work would not have been possible at all. My sincerest thanks go to Dr. Kim for keeping me up-to-date on the status of the ALAX prototype, and for the numerous informative discussions. I would like to extend my gratitude to Dr. Makowski for his time, support and patience during the iterative writing process for the thesis, and for all his invaluable suggestions.

I wish to acknowledge Dr. Russell and Mattie Riley. Both of you gave me great support and stimulation to proceed with a thesis. I owe it all to you. The road may have been long but I believe it was worth it. Thank you for all the advice. I must also mention my friends and classmates, Rohit Mahajan and Adil Rajput. They have made the research environment at the University of Maryland an exciting one to remember. Special thanks go to the entire ISR staff for their support administratively and especially with the computing facilities.

Finally, I would like to thank Drs. Makowski, Narayan and Silio for being on my thesis committee, and for their time and recommendations.

July 15, 1997 - Giles C. Charleston

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ARCHITECTURE, DESIGN, SIMULATION
AND PERFORMANCE EVALUATION
FOR IMPLEMENTING ALAX-
THE ATM LAN ACCESS SWITCH INTEGRATING
THE IEEE 1355 SERIAL BUS

Giles Colbert Charleston

February 15, 1998

This comment page is not part of the dissertation.

Typeset by LaTeX using the `dissertation` style by Giles C. Charleston,
Institute for Systems Research (ISR)- University of Maryland at College Park.

Chapter 1

Background and Summary

1.1 Introduction

"A Local Area Network (LAN) as an isolated entity by itself has limited potential and usefulness"[39]. For instance, the number of stations which can be attached on the physical infrastructure of the LAN, is generally restricted due to technical or performance constraints. Likewise, the geographical span of a LAN typically covers about 1 Km. And furthermore, the resources of a single LAN are often inadequate to meet all users' needs.

Early though, during the evolution course of LAN technology, scientists and engineers addressed these problems. Henceforth, the concept of LAN internetworking surfaced. LAN internetworking expands the coverage area of a local area network. It offers to any computer user the ability to share resources beyond that available on the user's own LAN, but also to have access to devices, services and users on other networks.

The internetworking terminology alone describes the generic connection or interconnection of local and remote communication networks required to create an extended network. With the availability of many different environments (i.e., standards such as Token ring, Ethernet, etc.), LAN internetworking thus provides the capability to con-

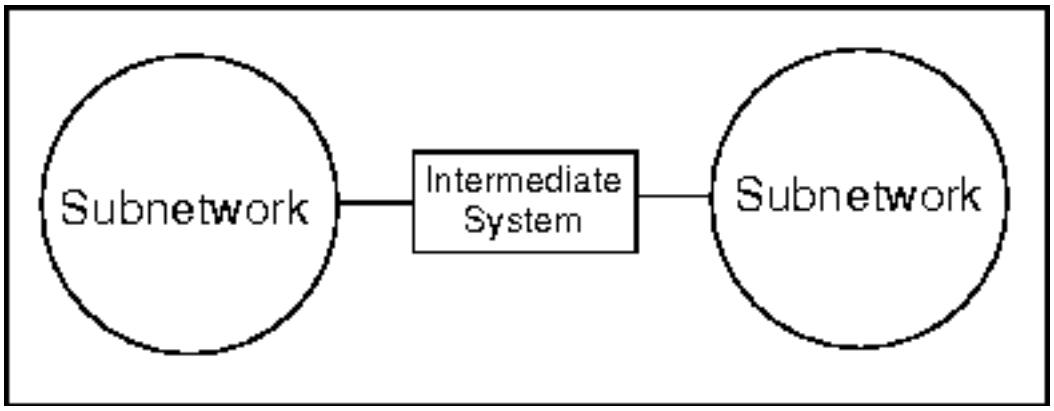

Figure 1.1: Conceptual overview of internetworking

veniently interconnect various networks, so that any two stations on any constituent networks may communicate independently of their underlying network protocols and architectures. In the end, from a user's point of view, the resulting interconnected set of networks appears simply as a larger network. The entire configuration is then referred to as an internet, while each of the constituent networks is called a subnetwork [6,19,39,40].

During the past decade, LAN internetworking has been effectively and efficiently achieved wherever appropriate, primarily through *intermediate systems* or devices known as bridges and routers (see Glossary). However today, with increasing populations of users, the use of new computing paradigms such as the client-server paradigm, the requirements imposed by emerging high-bandwidth multimedia applications and the accessibility of computers providing greater processing power at the desktop, existing routers and bridges have clearly been identified at the roots of the major bottlenecks across LANs [27]. In other words, the network traffic congestion occurs precisely inside routers and bridges. As these intermediate systems were not specifically designed to handle the arising excess of traffic, they are rapidly reaching obsolescence.

In order to cope with these difficulties, the current trend in internetworking design

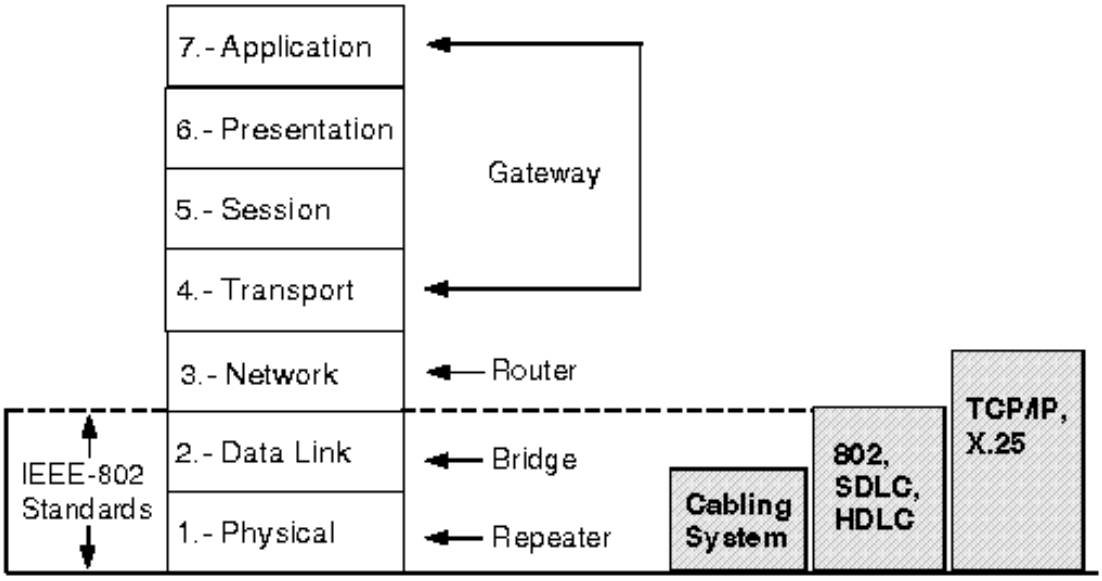

Figure 1.2: Relationships of repeaters, bridges and routers to the OSI reference model

involves integrating a switch, based on advances in VLSI technology, to steer frames at wire speed between stations or networks. This is not a total replacement of bridges and routers, but rather using switches wherever necessary to alleviate slow responses and delays [8,20,27]. Nonetheless, there is still a need for routers to efficiently interconnect to the upper layers. Depending on the network configuration, a switch can allocate dedicated bandwidth to the user's desktop station. Thus, with switches the network performance can increase tremendously, while the bandwidth available to any station can be guaranteed more effectively.

The functionalities of a switch are very similar to a bridge. But, a switch exceeds the performance of a bridge, because the switching and forwarding of packets are done in hardware (i.e., silicon fabric) rather than software [8,20]. Two major types of switching-based internetworking technologies are presently in use, namely LAN and ATM switching [20]. The former reflects a short-term solution approach extending the use of existing shared-medium network hardware and software, while the latter positions itself as a long-term objective consistent with emerging high-speed network

architectures, specifically the Asynchronous Transfer Mode (ATM) technology.

The focus of this thesis is on the combination of ATM and LAN switching, which is referred to as ATM LAN switching. We shall be concerned with developing a standard-compliant approach to integrate both technologies onto one common switch platform to efficiently interconnect ATM and LAN networks.

1.2 Motivation

ATM LAN switching comes into play in order to set up a clear and simple migration path towards the use of ATM technology across LANs. This is necessary for several reasons, foremost among them are strategic network planning and also performance needs.

ATM has been touted as the transport technology for the future generation of high speed communication networks. It is a fixed-size cell switching technique that can sustain the gigabit ranges and can support any kind of traffic (data, video and voice) (Chaps. 2 and 3). The attractive characteristics of ATM include the simplicity of the technology, the improvement in packet transfer delay that it provides and most of all, the large variety of applications that it can integrate across both LAN and WAN seamlessly.

But before the advent of ATM, other technologies have been and are still being used in LANs. Large LAN hardware and software are either Ethernet or Token ring-based, and these very popular conventional technologies are expected to remain in use for the foreseeable future. Consequently, if the ATM deployment is to become a reality, prior hardware and software investments made in these traditional networks must be protected, while the beneficial properties of this new technology are fully exploited. Thus, ATM must coexist with legacy LANs for some time before the widespread use of fully ATM-based networks can become a reality.

Hence, an ATM LAN switching platform should be of beneficial use in current networks experiencing for instance, slow server responses. This can be achieved through intermediate systems called ATM LAN access switches or edge devices.

1.3 Goals

With respect to the context exposed before, this thesis addresses the architectural and performance issues relevant to the modeling of an ATM LAN Access Switch baptized ALAX. Overall, ALAX is part of a research project undertaken at the University of Maryland, to develop the internetworking interface between legacy LANs and ATM. The ALAX project comprises two major steps: an extensive simulation phase and the construction of an experimental prototype; the former allows for an in-depth study of the switch in order to derive qualitative design parameters, while the latter leads to the realization of the switch.

ALAX is based on the IEEE 1355 Serial Bus Standard for Heterogeneous Interconnect (HIC) (Chap. 4). The use of this recently approved standard as the communication paths interconnecting interface modules across the ALAX switch is successfully achieved via the SGS Thomson C104 Packet Routing chipset. With the C104, ALAX adheres to a cost-effective system integration requirement, modularity, simplicity and a parallel architecture. The ATM LAN internetworking aspects of the switch are achieved through the integral use of the LAN Emulation (LANE) protocol suite as specified by the ATM Forum, which allows an ATM network to emulate LAN services. The LAN type being considered is Ethernet.

ALAX is the result of a joint research venture titled, " A Standardization and Research Project on An ATM/B-ISDN Switching Fabric System". The partners involved in this enterprise included the Protocol Engineering Center (PEC) of ETRI (Electronics and Telecommunications Research Institute), a Korean research organi-

zation, Modacom, a Korean electronics company, and the Laboratory for Advanced Switching Technology (LAST), an affiliated research laboratory to the Institute for Systems Research (ISR) at the University of Maryland at College Park. As part of this collaboration, LAST has provided the concepts, modeling, analysis and simulation tools, while the actual hardware implementation is currently being pursued by ETRI and Modacom. The questions under consideration include:

1. the architecture of the switch (both from the protocol and hardware standpoints);
2. the organization of the switch (several configurations);
3. the number of components, e.g., transputers, port interfaces;
4. the optimal size of memory (i.e., buffer occupancy) for processing and managing translation of packets;
5. the performance issues, such as packet transmission delay, time for packet translation; and
6. the future applications that ALAX may support

During the past 2 years, an ALAX simulation model has been created and is now fully operational for experimenting. Several of the characteristics enumerated above have been determined for the basic 4-by-4 original model (i.e., with 4 physical ports where there are 4 input and 4 output queues at the C104). We have been exploring further some other aspects with the ALAX Simulation Model, including: implementations of 8-by-8, 12-by-12 and 16-by-16 models of the ALAX switch.

1.4 Plan of Thesis

The work is divided as follows. In chapter 2 and 3, we review the literature on LANs and ATM. Chapter 2 briefly takes a look at the evolution of LANs from

early shared- to the current switched-access. This leads then into a concise overview of the ATM technology and, further a summary of the internetworking strategies to interconnect ATM-based networks and LANs. Emphasis is given to the works of the ATM Forum and the IETF, namely LANE and Native (or Classical IP). Chapter 3 identifies several architectures for ATM switching in the context of LANs, and surveys some of the actual implementations, and their related properties and characteristics. ALAX is conceptually outlined in great details in Chapter 4. From inception to design, the purpose of the ALAX switch is defined while its architecture is laid out along with schematic diagrams to illustrate the physical and logical attributes of this switch. Chapter 5 describes ALAX within the simulation modeling framework. It discusses simulation tools and packages suitable for modeling. From a tradeoff analysis, OPNET is selected as our simulation environment for ALAX. Then, the OPNET simulation model for ALAX is described and explained. The system modules are identified with their respective function. In Chapter 6, the results of the simulation experiments for the 4-by-4 ALAX model are reported for qualitative and statistical evaluations, and some conclusions. Chapter 7 pursues with further simulations in the cases of the 8-by-8, 12-by-12 and 16-by-16 ALAX model configurations. Results are presented and analyzed. Finally, Chapter 8 of the thesis synthesizes the work done, and discusses ideas to extend the functionalities of the ALAX switching entity such as ATM with Token ring, MPEG and ISDN.

Chapter 2

Evolution of the LAN Infrastructure- A Technological Assessment

First, we present the different technical and technological innovations that have influenced the progress of LANs. One major trend is the move from early shared-medium access towards switched-access. Switches are looked upon as the underlying foundations for future LANs. This prospect is put into perspective when envisioning the emergence of high-speed networks, such as ATM, the appearance of computers with much faster CPUs, and the growing usage and popularity of bandwidth-intensive or delay-sensitive applications. Finally, we discuss the ATM networking technology, the intent behind it, the benefits that it offers and most importantly the strategies to deploy ATM into current networks, specifically legacy LANs.

2.1 Four Generations in LAN Networking

The LAN market has been one of the fastest growing areas of networking for the past ten years. Shared-medium access LANs make up the predominant local enterprise networks around the world. In the United States alone, there were 3.1 million LANs in 1993, up from 0.5 million in 1991. And this trend is likely to continue until the end of the century [8]. Undoubtedly, the key aspect of LANs that has made this possible, is

the fact that the underlying LAN technologies have consistently changed in adherence with the demands of the time. We can clearly distinguish four generations in LAN deployments.

The *first generation* is dominated by small shared-media LANs such as early coaxial-bus topology Ethernet. These early LANs supported less than 100 users whom were for the most part, located in close proximity to one another. These LANs came into existence primarily to meet the communication needs among various distributed computers appearing on the desktop. One typical topology of these LANs was centralized around mini-computers [8]. Then, came Ethernet with a bus-topology architecture based on carrier sensing access protocol with collision detection (CSMA/CD) (see Glossary). Most of the drawbacks encountered in these LANs were ad hoc direct-wiring, and the inexistence of appropriate cabling standards and specifications to allow practical troubleshootings of the network.

Some of these issues were later resolved in the *second generation* of LANs where systematic cabling systems were introduced, pioneered by IBM and AT&T. With these new systems, manufacturers did not need to guess or choose a set of specifications for cable type, length and attenuation crosstalk, thus preventing earlier poor performances, financial losses, recall of products and customer dissatisfaction. Then, the EIA (Electronics Industry Association) established the cabling standards. Star-wired token ring with speeds of 4 and 16 Mbps, and Ethernet on unshielded twisted pair (UTP) cabling with speed of 10 Mbps symbolize this second generation of LANs. During the appearance of this second generation of LANs, routers and bridges were introduced as the alternative intermediate systems to interconnect LANs.

At the *third generation* of LANs, with the systematic cabling systems, previously mentioned, the access point of LANs moves into one central place, the wiring closet. For token ring, this access point is designated as a concentrator and for Ethernet based on UTP, a repeater. In a configuration where there is a combination of concentrators,

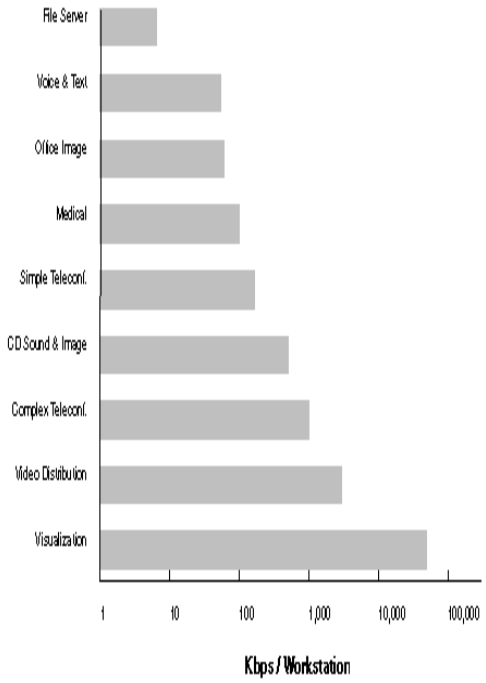

Figure 2.1: Bandwidth requirements for current applications

repeaters and possibly other functions (such as bridging) in a single chassis, this is referred to as a hub. Each card in a hub is called a module, and for each function performed by the hub there is a module. Intelligence is added in the form of microprocessors programmed for specific management and control functions.

Another important technological development which is having a considerable impact on the evolution of LANs, is switching. Because networks have been stretched to their limits in recent years, the response times over the LANs have dramatically increased. One novel method to decrease the heavy utilization of a LAN segment requires reducing the number of stations on the network segment, then further subdividing a segment into other subnetwork segments and finally interconnecting the subsequent segments through switches. This solution approach can provide a significant increase in available bandwidth to the individual stations. When a switch is used across a LAN to forward frames, this is referred to as *LAN switching*. LAN switching is commonly

deployed nowadays, and is expected to maintain the expansion and evolution of LAN technology through the next decade. Since it preserves continued growth of the existing conventional LAN technologies while enabling a new breed of high-bandwidth applications like distributed databases, document imaging and video teleconferencing, to run efficiently (Fig. 2.1). It is also very flexible, because it can interface easily in the context of the future generation of LANs with high-speed networks in the backbone (FDDI, Fast Ethernet, 100VG-AnyLAN and ATM).

Throughout the emerging *fourth generation* which is being shaped today, LANs will continue to change vis-à-vis the new technologies being introduced in applications and network infrastructures. Even though there seems to be a lot of standardization and development work going on for high-speed LANs based on shared-medium access (e.g., Gigabit Ethernet), it is worth noting that these technologies do not resolve the problems of slow response times and long delays, because as the load on these high-speed LANs increases the performance levels will be very similar to the conventional LANs. Therefore, the future of LANs is likely to be based on switching technology, where appropriate with LAN switching or ATM switching implementation.

2.2 The ATM Factor

To get a concise view of the ATM networking technology, two key points will be made. Primarily, one must clearly understand what is ATM and why it was developed. This will identify what ATM can offer with respect to LAN networking especially. And finally, the approaches or methods to create internetworking paths between the LAN and ATM networks must be studied and analyzed.

2.2.1 Rationale and history

ATM was born out of the research initiatives to meet the requirements for the

Broadband Integrated Services Network (B-ISDN). Earlier communication networks were generally conceived and designed for one type of service. This explains in large part why networks in use today, bear the characteristics of specialized networks. In other words, they can only provide a definite service, and their network transport infrastructure is usually dedicated to serve the needs of that specific service. Seldom, are there cases where other kind of traffic can be easily accommodated on these networks. For example, the wide area telephone network can transport data and voice, while no voice can actually be routed on the cable television network, nor can television signals be carried on the telephone system using conventional technologies.

Furthermore, the transport mechanism of these networks is either circuit or packet switching, which have their limitations in terms of efficient use of bandwidth. Circuit switching statically reserves the required bandwidth prior to establishing a connection, whereas packet switching acquires and releases it as it is needed throughout the connection. To understand the implication of these schemes, one should realize that the telephone network is basically a circuit switching system, while the LANs operate with packet switching methods. With circuit switching any unused bandwidth on an allocated circuit is wasted (e.g., moments of silence during a telephone conversation). In contrast, with packet switching this unused bandwidth may be utilized by other packets from unrelated sources going to unrelated destinations because circuits are not dedicated.

In sum, the existing communication networks reflect service dependency, inflexibility and inefficiency in terms of optimal usage of available resources. And, it is to resolve most of the anomalies discussed above, that in the late 1970's the major telecommunications carriers grouped in a common body of the ITU, envisioned and proposed the concept of B-ISDN which extends the notions of the long awaited ISDN standard. Broadband here refers to services and networks that can handle interactive multimedia traffic.

The driving force behind this integrated communication architecture was to bring voice and video in real-time to the end-users, in the form of services like video telephony. Later on, data was also included as another requirement and so a multitude of services could be delivered on a single network, e.g., interactive multimedia with bi-directional, real-time video, voice and data traffic. Consequently, this conceptual network had to incorporate large bandwidth and long delays. B-ISDN was pushed by the telephone carriers to replace the old analog phone system worldwide with a new digital infrastructure featuring the characteristics indicated above. Overall, the objective was to satisfy the needs and desire of modern communications by devising a flexible, service independent and efficient network skeleton.

But the major obstacles to B-ISDN were, among many, the fact that this high-speed network could not be implemented on top of the conventional architectures and protocols defined for lower-speed networks, and by the same token, the inexistence of a transmission technology to allow inter-operation between already existing network standards. Thus, to make B-ISDN a reality, a mechanism was needed to move some functions into lower layers and into hardware implementations.

In 1980, at AT&T's Bell Laboratories and France Telecom's Research Center, a big step was accomplished towards some actual realization, and the ATM technology was born. ATM gave at low-layer (hardware), switching, routing and multiplexing of virtual circuits and point-to-point transmission facilities.

Subsequently, the physical layer of B-ISDN was standardized to be SONET (622 Mbps) while the network protocol layer consisted of ATM. Noteworthy, though, B-ISDN implied ATM, but ATM did not imply B-ISDN. What this meant in actuality, was that given its advantages, ATM could be used outside of B-ISDN, running on non-SONET-based networks with such low speeds as 100 Mbps and 25 Mbps, and even lower (see Fig. 2.3). Thus, ATM was seen as the glue that would seamlessly unite both WANs and LANs.

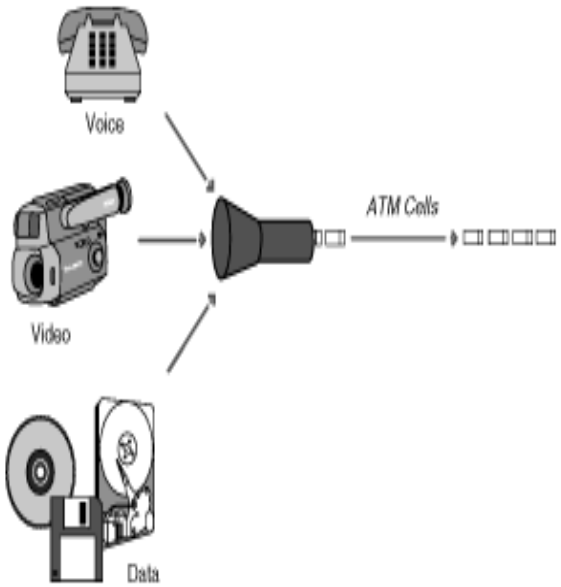

Figure 2.2: ATM Traffic Classes

2.2.2 Description

ATM is a connection-oriented protocol. It combines advantages from both packet and circuit switching transmission techniques. It multiplexes and switches cells from multiple sources to multiple destinations. It uses cells of a fixed size of 53 bytes, with 48 bytes of payload and 5 bytes of overhead. The "word asynchronous is being used since an asynchronous operation between a sender clock and and a receiver clock is possible. The difference between both clocks can easily be solved by inserting/removing empty/unassigned packets in the information stream, i.e. packets which do not contain useful information"[9,pp.61].

ATM cells are carried through logical connections known as virtual circuits. A virtual circuit looks like a "real" connection to users, but is really made up of a set of logical links with only local significance. In the ATM cell header, virtual circuits

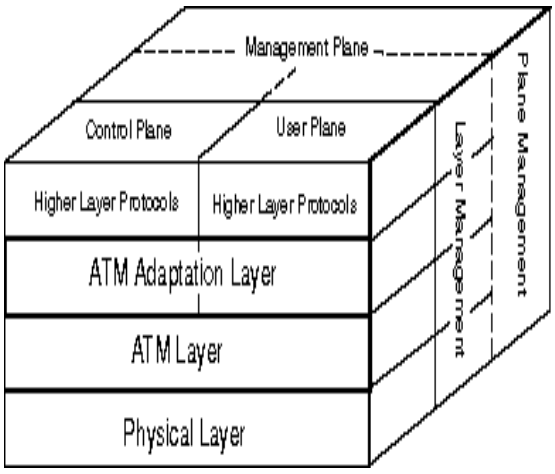

Figure 2.3: B-ISDN protocol reference model

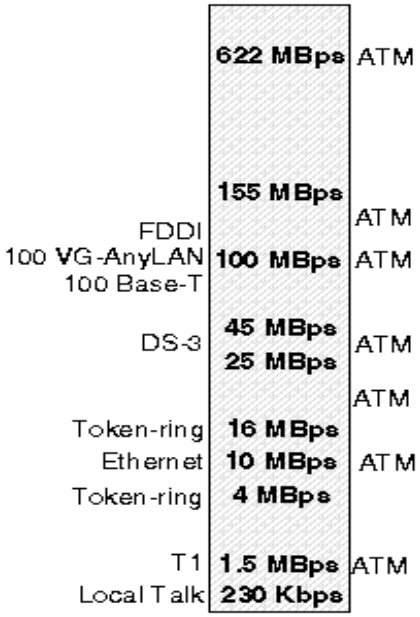

Figure 2.4: Protocols and speeds supported by ATM

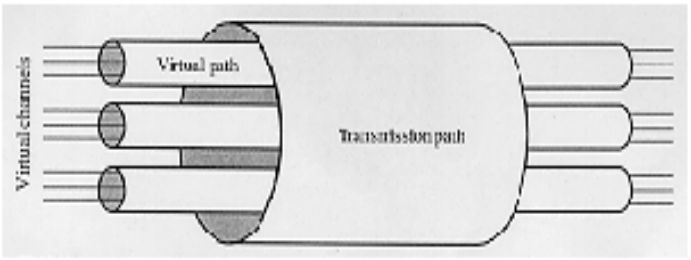

Figure 2.5: Profile of ATM connections

are identified by a pair of fields, the virtual path identifier and the virtual channel identifier, VPI and VCI respectively. VPIs and VCIs are not addresses however. They are temporarily assigned for multiplexing, demultiplexing and switching a cell through each step of its journey through the network. The virtual path implied by this principle, can be visualized as a conduit that can hold many wires, with each wire being a virtual channel (Fig. 2.5). By definition, Virtual channels (VCs) and virtual paths (VPs) are always unidirectional. When allocating VPI and VCI values to a virtual connection, the same values are used in both backward and forward directions. A virtual connection can be said then to consist of two virtual channels (one in each direction). Connections are either Permanent Virtual Circuits (PVCs) or Switched Virtual Circuits (SVCs). SVCs use dynamic routing and load balancing, whereas PVCs are strictly static [6,9,39,40].

In ATM, each connection has a Quality of Service (QoS) which is associated with it (Table 2.1), and negotiated during the setup of the connection [5,9]. ATM will drop cells if necessary to meet its connections' QoS requirements, and the higher layers must recover from dropped cells. Elaborate traffic management schemes determine which cells must be dropped in order to maintain QoS to which the network has committed for all its connections. Congestion control schemes keep traffic flow smooth to prevent the emergence of conditions in which cells would be dropped.

In order to convert every type of traffic into ATM cell formats, the ATM Adapta-

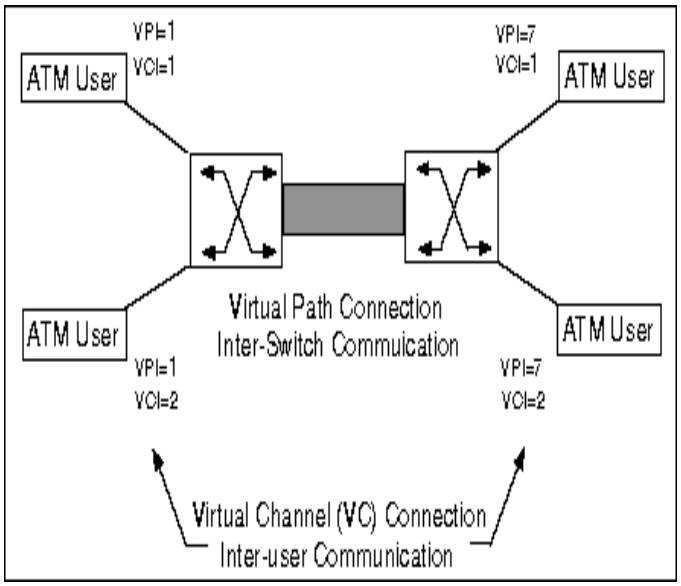

Figure 2.6: Conceptual operation of ATM virtual paths and channels

QoS Class	Traffic Patterns	Service Class	Application
0	Unspecified	UBR, ABR	best effort
1	specified	CBR	voice, circuit emulation
2	specified	VBR	video, audio
3	specified	Conn. Data	data transfer

Table 2.1: Quality of service classes in ATM

tion Layer (AAL) is used. A number of different AALs are available for segmentation and reassembly (SAR) functions, depending on the specific needs of a connection. AAL1 is for Constant Bit Rate (CBR) services (e.g., voice), AAL2 is for Variable Bit Rate (VBR) service with required timing between source and destination (e.g., audio, video), AAL3/4 is for VBR connectionless services and AAL5 is a simpler version of AAL3/4, with less error checking and protocol overhead (appropriate for data communications).

2.2.3 Emergence

ATM was conceived primarily as a communication transport across the Wide

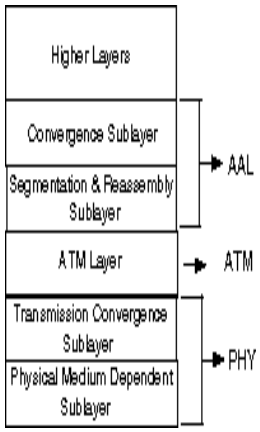

Figure 2.7: The ATM protocol stack

Advantages	**Disadvantages**
Fixed-size cells allow efficient switching in hardware	High overhead for data transfer
Multiplexes multirate multimedia traffic	Connection-oriented model complicates connectionless service
Dynamic bandwidth management and allocation	Immature technology (e.g.,congestion problems)
No shared media: statistical multiplexing	Multiple types of traffic may not aggregate well
QoS class support, guaranteed service	
Speed scaleable	
Distance scaleable	
Automatic Configuration: Topology Discovery, Failure Recovery, Dynamic re-routing,Load Balancing	

Table 2.2: Advantages and disadvantages of using ATM

Area Network (WAN) to run over SONET carriers with OC-3 (155Mbps) as the most common transmission rate. However, it was determined applicable also in other physical-layer signaling, such as TAXI in the LAN and DS-3 in the WAN [5]. A variety of standards have been developed since then for different speeds (e.g., 25 Mbps). The signaling between a host station and the ATM network is known as a User-to-Network Interface (UNI), while the signaling between two networks is called Network-to-Network Interface (NNI) [2,5].

The ATM Forum was created in order to facilitate the increasing presence of ATM applications and products in the networking field. This body includes over 500 organizations and companies representing all sectors of the communications and computer industries, as well as a number of government agencies, research organizations and user groups. This consortium oversees the drafting of the standards for the development and deployment of ATM-related products and services, and the maintenance of interoperability across the different vendors.

Even though this thesis is focusing solely on an ATM LAN access device, other ATM products deserve being acknowledged as well. The following are some of the most common ATM products which we can mention:

1. ATM switches
2. ATM host adapters
3. Routers with ATM interfaces
4. Hubs with ATM interfaces
5. ATM CSU/DSU

2.3 The ATM LAN Internetworking Paradigms

Even though ATM was specifically designed for telecommunication transport, it

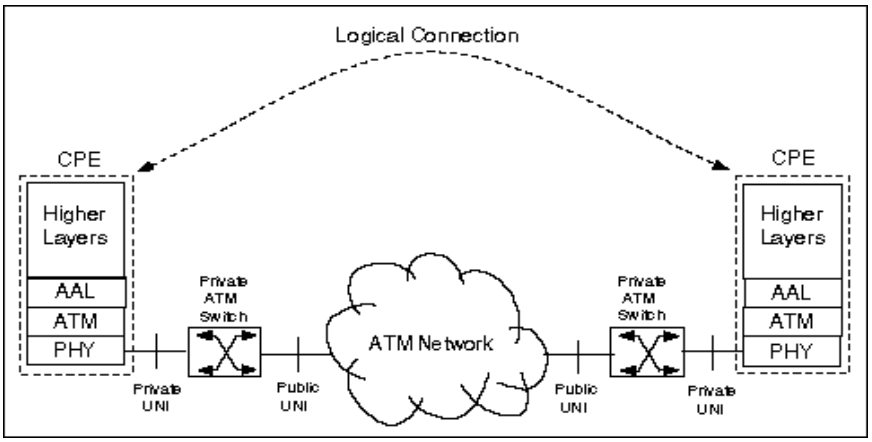

Figure 2.8: ATM Network Architecture

has been widely accepted as the ultimate solution, at least in the long-term, for current LANs. However, many organizations are not ready to cast aside their significant installed base of traditional LANs to make room for an end-to-end ATM solution. Nonetheless they would gladly enjoy taking advantage of the performance increase that it can incorporate into their existing networks.

Standard bodies, computer vendors and service providers, who favor the ATM technology, have been working actively to alleviate these constraints. A common approach is to have ATM networks in the backbone internetworking with the LANs. This has also introduced a new concept called "virtual" LANs [2,8,29] whereby ATM end-stations can be logically subdivided into logical LANs for security and administrative reasons. With these ATM LAN architectures, networks can migrate smoothly from low speed to high-speed connectivity, and grow from few users to many users without congestion and disruption. This can help keeping the current legacy LANs operating at greater throughput, while giving network managers the convenience of slowly moving towards ATM networks. For example, an ATM LAN edge device like ALAX, with Ethernet ports offers the advantage that each Ethernet port can use Ethernet's full 10 Mbps bandwidth potential, while exploiting ATM's non-shared media.

2.3.1 LAN Emulation and Classical IP over ATM

Two completely different standards have been defined for integrating legacy LANs with ATM networks. The ATM Forum has published its LAN Emulation version in January 1995, but a full year earlier in January 1994, the Internet Engineering Task Force (IETF) proposed its own method of running IP traffic over ATM networks [43,44]. This IETF specification is known as "Classical IP and ARP over ATM" and labeled RFC 1577. It is intended to make IP run over ATM as efficiently as possible. Along with RFC 1577, IETF has also developed RFC 1483, which defines the encapsulation of IP datagrams directly in AAL5. The ATM Forum released its long awaited LAN Emulation specification, hereafter referred to as LANE 1.0 . Overall, both approaches enable the use of existing LAN applications over an ATM network without modification. The key difference between the two approaches is that Classical IP over ATM is focused on IP networks only, while LANE is not limited to a single protocol. LANE supports both Ethernet and Token Ring networks, as well as the IP protocol and network operating systems such as Netware[1] and Windows NT[2][44].

LANE is the migration technology that lets applications requiring features unique to legacy protocols run on ATM. It is essentially a MAC layer specification that allows the ATM fabric to emulate logical Ethernet or Token Ring segment. It transparently interconnects legacy LANs, performing protocol conversion, address resolution and requiring multicast/broadcast delivery. LANE is an important feature in the LAN-to-ATM migration phase for two major reasons:

1. It allows an ATM network to be used as a LAN backbone for hubs, bridges, Ethernet switches, and the bridging feature in routers.

2. It allows end-stations connected to legacy LANs to communicate through an

[1]Software product developed by Novell.

[2]Software product developed by Microsoft.

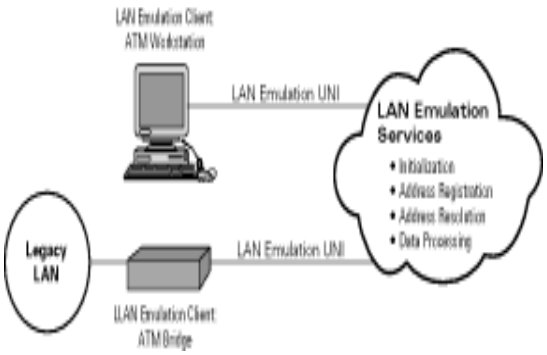

Figure 2.9: LAN Emulation architecture

ATM LAN hub/bridge/switch without requiring traffic to pass through a more complex device such as a router.

LAN Emulation does not replace routers or routing, but rather provides a method of transparently interconnecting legacy LANs. Since almost all LAN protocols depend on broadcast or multicast packet delivery, an ATM LAN must provide the same service. LAN Emulation uses the concept of "virtual" or emulated LANs (ELANs) to allow devices connected to ATM to find and communicate with one another. It coordinates the set up of SVCs on demand and multiplexes LAN traffic on existing SVCs with an ATM end point. LAN addresses must be resolved to a destination address and an ATM address for the terminating switch, because ATM uses a 20-byte addressing scheme, while Ethernet and Token ring both use a 48-bit MAC address to identify stations. Furthermore, since ATM is connection-oriented, packets go only to the stations to which they are addressed. Ethernet on the other hand, is connectionless with packets going to all stations on the network but being acknowledged only by the station to which they are addressed. LANE typically requires four entities in order to run: LAN Emulation Clients (LECs), a LAN Emulation Server (LES), a LAN Emulation Configuration Server (LECS) and the Broadcast and Unknown Server (BUS) [43,44].

LANE follows the client-server model, in which multiple clients connect to the LANE service (LES) components. LANE clients (LECs) are typically implemented

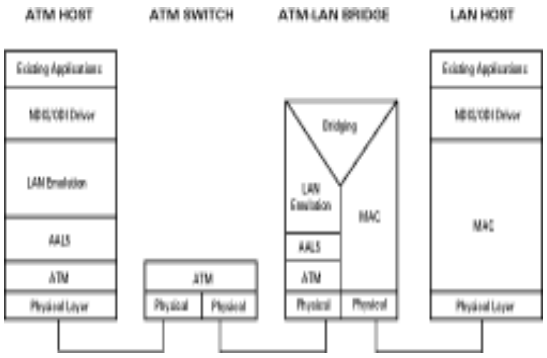

Figure 2.10: LAN Emulation Protocol Stack

as software on adapter cards, routers or ATM LAN edge devices/switches. The LEC maps the MAC addresses of all known LAN stations to known ATM virtual circuits. If a LAN PDU's (Protocol Data Unit) destination is local, that is, within the LANE's immediate domain, LANE forwards the packet locally. Otherwise, if the destination is a subnetwork across the ATM network, the client LANE interface converses with the LANE server to request an SVC, perform address resolution, then sends the packet over ATM to the destination address.

A LAN Emulation Server is at the heart of the ELAN and basically controls communication between stations on the ELAN. Every ELAN must have its own LES. LES builds and manages an address table that matches the MAC address of each device on the network with the ATM address. LES keeps a connection to every LEC in the ELAN.

The LECS resolves MAC addresses and ELAN numbers to help a LEC find its proper LAN Emulation Server. If there are several ELANs, there must be a LECS. Only one LECS is needed for any ATM network, no matter how many ELANs are contained in that network. LECS identifies the type of each ELAN and the whereabouts of its LES.

A BUS is in charge of receiving broadcasts and multicasts, and forwarding them to the appropriate stations. This is necessary since ATM being point-to-point, does

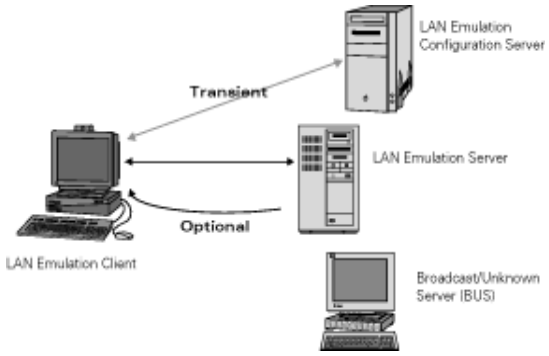

Figure 2.11: Control connections in LANE operation

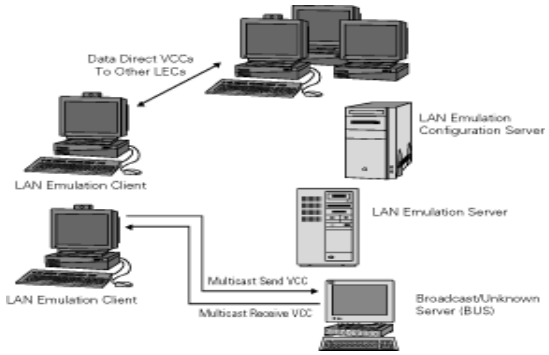

Figure 2.12: Data connections in LANE operation

not know how to handle broadcasts and multicasts. BUS also helps the LES send messages to stations unknown to the LES.

The LAN service components can be distributed among multiple switches or routers across the network to support multiple emulated LANs, or centralized in a central controller.

LANE can run on top of two standards: User-Network Interface (UNI) 3.0 or UNI 3.1 (UNI 3.1 is the updated version of 3.0). The UNI specifications basically define the way in which switched virtual circuits (SVCs) are established. Devices register their ATM addresses with a switch using UNI specifications. This allows the switch to correlate the device address with a port number. LEC, LECS and LES must register with UNI before they can find each other and participate in LANE.

RFC 1577 or Classical IP and ARP Over ATM, defines the ATM address resolution protocol to map IP addresses to ATM addresses within a logical IP subnetwork. A router must be used for communications outside of a subnetwork. For networks running only IP, as in the case of UNIX-based stations and IP routers, Classical IP is arguably more efficient than the ATM Forum's LANE. In a mixed network, with IP on Ethernet and ATM for example, routers with ATM interfaces can be employed to translate between protocols, although LANE allows the use of cheaper Ethernet-to-ATM switches in place of the more expensive routers. Another advantage of LANE is that the specification supports multicast, while RFC 1577 currently does not. However both methods need routers when communicating between subnetworks or LAN segments.

2.3.2 Multi Protocol Over ATM

MPOA (Multi Protocol Over ATM) is the next major internetworking protocol that is being considered by the ATM Forum. It is up for completion in early 1997. In many respects, MPOA takes off where LANE finished. The goal is not for MPOA to replace LANE, but for LANE to evolve into MPOA. MPOA can be considered an evolution of LANE because MPOA operates at both layer 2 for bridging and layer 3 for routing, while LANE operates by bridging at layer 2. Since MPOA is a far more ambitious and complex proposal than LANE, it is expected to overcome some of LANE's limitations. For instance, MPOA provides direct connectivity anywhere in the ATM cloud, while LANE relies on routers to connect subnetworks. LANE was intended to hide ATM's QoS from the legacy application, while MPOA will partially expose ATM QoS.

The work on MPOA involves close assimilation with LANE. For example, MPOA will utilize LECS, which allow hosts to automatically configure. Since it was developed specifically for LANE, the ATM Forum has made some changes into the LECS

specification in order to make the definition more general so it can be adapted to MPOA. MPOA also involves the contributing efforts of the IETF, since it will pursue work done by the IETF's Routing Over Large Clouds group, in particular the NHRP (Next Hop Resolution Protocol) and Multicast Address Resolution (MARS), a means of doing IP multicast address resolution over ATM.

The guiding concept behind MPOA is the separation of switching and routing through the use of a route server. Switching based on dedicated silicon is fast and cheap. The route server is software running on an inexpensive workstation. When a switch receives the first packet, it sends an address query to the route server, which returns the necessary routing information to a cache in the edge device. Subsequent packets are forwarded along without any further processing. A router in comparison examines every packet it receives, and is thus slower despite being more expensive.

2.4 Summary Note

The LAN world has considerably changed from what it was ten years ago. And most of these changes were forced by the inefficiencies and flaws experienced within the current legacy LAN architectures combined with the novel improvements introduced by advances in technology. These LANs were for the most part shared-medium based.

As ATM is projected to be the data transport infrastructure for future LANs, there is an immediate need to interface ATM with these existing legacy LANs in order to protect the prior investments in shared-medium LAN technology. Thus, using a switch to interconnect these networks in question seems the appropriate solution. In the next chapter we look at some of the switching architectures for ATM in the context of LANs.

Chapter 3

ATM Switching for the Network Infrastructure

We survey several architecture designs involved when considering ATM switches. We outline the key factors that interplay in the optimal organization of these switches. Finally, we briefly address the performance characteristics of an ATM switch.

In the description of ATM switching, attention will be paid in this chapter only to the "transport" part of the switch, also called the transport network, and not to the "control" part of the switch, responsible for handling the signaling functions [9].

The *transport network* is defined as all physical means which are responsible for the correct transportation of the information from an ATM sending device to an ATM receiving device, within the QoS specifications of ATM. This transport network mainly performs functions located in the user plane of the protocol reference model in Fig. 2.3 . The *control part* of the switch is that which controls the transport network. It, for instance, decides which input device to connect to which output one. This decision is based upon incoming signaling information or set by an operator on a semi-permanent basis. The control network mainly performs functions located in the control plane of the protocol reference model in Fig. 2.3 [9,30].

3.1 Switch Categories

ATM switches operate between ATM networks themselves, and also across the

other popular types of networks used in the corporate, education and research environments. They can be classified into the following: Carrier, Enterprise/WAN and Campus/LAN [2,5,43,44].

Carrier switches provide services similar to a telephone company, in which the customer has no equipment on site. The switch can be made fully redundant (processor and fabric). The internal switching capacity is very high (up to 10 Gbps). Physically, the switches are kept in a central office within rack-stack configurations. They are large in size and they support WAN and SONET interfaces such as T1/T3 or DS1/DS3, and the OC-family, generally over single mode fiber and copper.

Enterprise networks are wide-area and metropolitan-area oriented, offer some redundancy and support interfaces to WAN signaling such as T1/E1/DS1, T3/E3/DS3, TAXI with single mode fiber, coaxial and copper. An Enterprise switch is often used as a carrier edge node, the point of access to carrier network.

Campus/Local ATM switches are LAN oriented, have less switching capacity, processing power and ports (4 or more). Furthermore they are physically smaller and much less expensive than carrier switches. Campus switches interconnect to LANs such as Ethernet, Token ring and FDDI. They can also provide for example DS3, TAXI and SONET interfaces over multimode fiber, coaxial, copper and UTP. Features such as LAN Emulation and Virtual LANs are included. In sum, there are three basic types of ATM switches:

1. Carrier (expensive, redundant high switching capacity)
2. Enterprise (WAN/MAN, some redundancy, single mode fiber)
3. Campus/Local (cheaper, legacy LAN interfaces, multimode fiber).

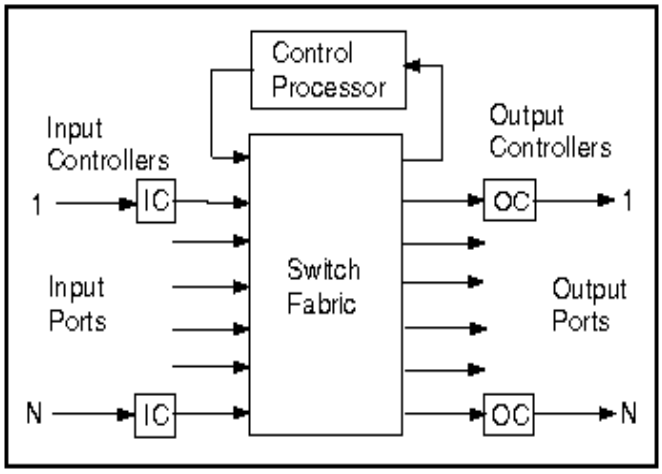

Figure 3.1: General structure of an ATM switch

3.2 Anatomy of an ATM switch

An ATM switch is essentially a connection-oriented device that functions at the ATM layer, to send and route ATM cells between sources to destinations. Unlike most packet switches, ATM switches are not store-and-forward switches, thereby reducing critical delays in the node. However, many ATM switches still contain input and/or output buffers for traffic management purposes.

The core of an ATM switch is the switching fabric (silicon). The general structure of an ATM switch is illustrated in Fig. 3.1 . All of the per cell processing functions are performed in hardware by the input controllers (ICs), the switch fabric and the output controllers (OCs). The control processor is required only for higher-level functions such as connection establishment and release, bandwidth allocation, maintenance and management. The control processor may communicate with the input and output controllers either by a direct communication path or via cells across the switch fabric.

All cells have their VCI translated by the input controllers before being submitted to the switch fabric. This operation is performed by table lookup on the incoming VCI in a connection table. The connection table may also contain a routing field to

specify the output port of the switch through which the virtual connection is routed. Other information may be included in the table on a per connection basis such as the priority, class of service and traffic type of the connection.

In an ATM switch, cell arrivals are unscheduled so that a number of cells from different input ports may simultaneously request the same output port. This event is referred to as output contention (or conflict). A single output port can transmit only one cell at a time. Thus, one cell must be accepted for transmission and any other simultaneously requesting that port must either be buffered or discarded. The location of these buffers has a major impact on the switch's overall performance, and greatly affects the complexity of the switch fabric. If buffers are not located at every point where contention occurs, a contention resolution technique is required to detect contention and to direct cells that can not immediately be handled, into the buffers.

The topology of the switch fabric, the location of the cell buffers, and the contention resolution technique are the most significant aspects of an ATM switch design.

3.3 Switching Fabric

The switch's fabric structure is an important consideration in any ATM switch hardware design. It affects the cost, performance, capacity, growth, capability and complexity of the switch design. Many alternative structures of switch fabric have been recommended in the literature. A simple classification of switch fabric design that includes most of the proposed approaches is shown in Figure 3.2 .

Most switching fabrics are composed of patterned collections of identical basic switching components called switching elements or switching building blocks, interconnected in a specific topology. Thus, a switch fabric is defined when its topology is determined and when the basic switching building blocks are defined. A switching element is frequently implemented within an integrated circuit while a switch fabric

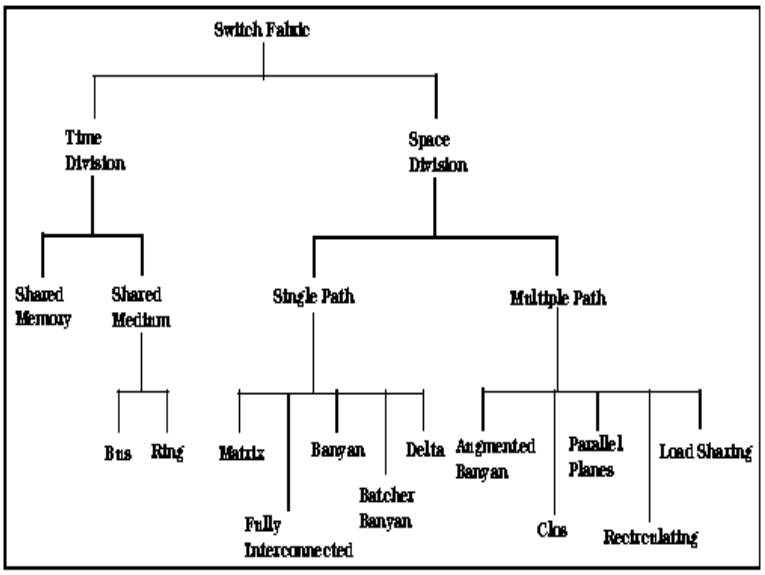

Figure 3.2: Classification of switch fabrics

requires one or more circuit cards.

3.4 Survey of Switch Fabric Architectures

The fundamental models for switch fabric architectures are: Time Division and Space Division architecture. Most designs implemented in ATM switches derive from one of these models or a combination thereof [5,30].

3.4.1 Time Division

Time division switching involves the partitioning of lower-speed data streams into pieces that share a higher-speed data stream with other data pieces. The individual pieces or time slots are manipulated by a control logic to route data from input to output [39].

A switch fabric based upon time division, allows all cells to flow across a single communication highway shared in common by all input and output ports (network

interfaces). This communication highway may be either a shared medium such as a ring or a bus, or a shared memory. The throughput of this single shared highway defines the capacity of the entire switch fabric, and therefore sets an upper bound on the capacity for any particular implementation beyond which it can not grow.

For the construction of small switches with a capacity up to a few Gbps, time division can be a flexible technique. Access to the bandwidth can be arbitrated dynamically among the switch ports on a demand basis. This supports the efficient multiplexing of interfaces with widely differing access rates. The advantages of a time division switching architecture are that it is simple and inexpensive for a small to a medium number of ports, and contentionless given that the aggregate port capacity does not exceed the switch's capacity. Its chief disadvantage is that it does not scale well for adding ports. Because more ports will quickly exceed the capacity of the entire switch, introducing port contention and exceeding buffers, thus reducing performance.

Shared memory designs have been reported operating with a total capacity of up to approximately 5 Gbps. Prelude [11] from the French CNET was one of the first ATM research projects. A number of manufacturers have developed a shared memory switching element as a single chip or a small chipset including Alcatel , the European RACE collaborative research program, Hitachi and Toshiba [1,11,30]. These devices have been built to serve in switch modules within a larger switch design.

Shared medium designs with a total capacity of up to about 10 Gbps have also been implemented. Several manufacturers have developed a shared medium switching element as a single chip or a small chipset e.g. Alcatel, the Atmospheric ring switch from the RACE research program and the ATOM switch from NEC [1,30].

3.4.2 Space Division

Space division switching was originally developed for the analog environment

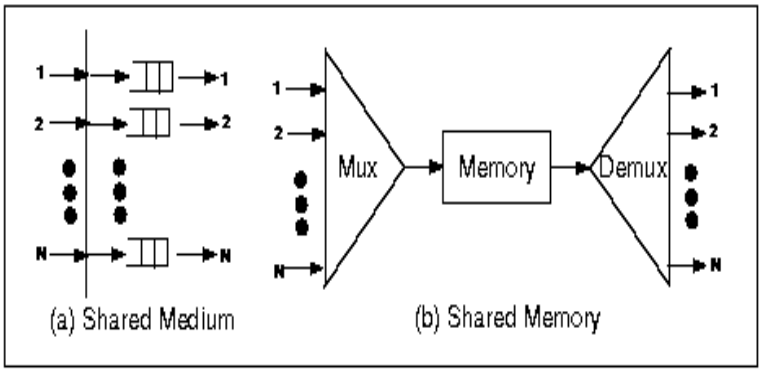

Figure 3.3: Switching fabric architectures based on Time division switching

and has been carried over into the digital realm. The fundamental principles are the same however. As its name implies, a space division switching fabric is one in which the communication paths that are set up are physically separate from one another (i.e. divided in space) [5,39].

Hence, contrary to time division where a single communication highway is shared by all input and output ports, in space division a plurality of paths is provided between the input and output ports. These paths operate concurrently so that many cells may be transmitted across the switch fabric at the same time. Thus assuming that each path has the same bandwidth, the total capacity of the switch fabric is the product of each path's bandwidth and the number of paths that on average can transmit a cell concurrently. The upper bound on the total switch capacity is therefore theoretically unlimited [30]. In practice, nevertheless, it is restricted by physical implementation constraints such as device pin-out, connector restrictions and synchronization considerations which altogether limit the size of the switch fabric.

The plurality of paths in the switch fabric requires a routing function in order to select a path to the appropriate output port for each cell. Either a self-routing or a label-routing technique may be employed.

In the *self-routing* approach the switch fabric is constructed from a self-routing

interconnection network of switching elements. Each input controller on the switch prefixes a routing tag in the header of every incoming cell using the same table lookup mechanism it uses for VCI translation. The properties of a self-routing interconnection network permit each switching element in the switch fabric to make a very fast routing decision simply by inspecting the routing tag. Each cell will arrive at the required destination regardless of the switch port at which it enters. The majority of switch designs based upon a space division switch fabric employ a self-routing scheme.

In *label routing*, the VCI (label) is used to index routing tables within the switch elements of the switch fabric. The label-routing approach permits a simple implementation of multicast operation, but requires a large number of routing and translation tables to be maintained within the switch fabric. In some switch designs, self-routing for point-to-point traffic has been combined with label-routing for multicast traffic [30].

Because a space division switching architecture allows multiple cells flowing through concurrently, the switch fabric is subject to the possibility of conflict between cells requesting the same path or output port. This conflict is resolved by a contention resolution scheme, or by introducing cell buffers at every potential point of conflict. Blocking refers to contention for a link occurring inside the switch fabric. Contention for the output ports of the switch fabric may still occur despite the switch fabric itself being nonblocking.

In general, there are several types of interconnection networks for a space division fabric. These may divided into two basic classes: single- and multiple-path networks. A single-path network has a unique path through the interconnection network between any given input and output pair. A multiple-path network has a number of different paths available between any input and output. Both classes of space division fabric types can be implemented within a single- or multi-stage switching configuration.

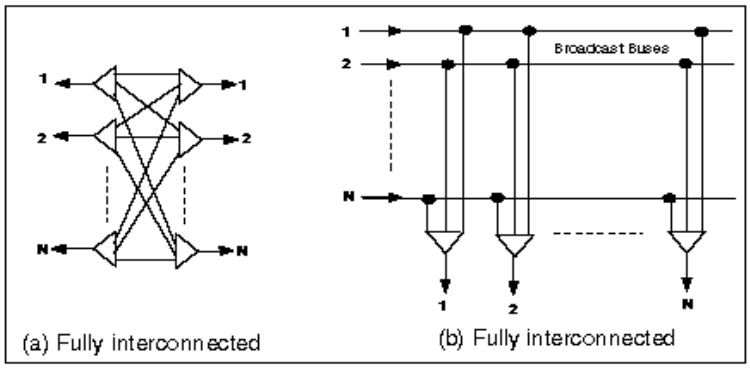

Figure 3.4: Single-path switching architectures based upon Space division (1st illustration)

Single-Path Networks

The single-path interconnection networks most often proposed for use in ATM switch designs are crossbar, banyan, delta and batcher-banyan [3,9,26,30,31,41,45].

The term "crossbar" derives from a particular design of single-path, nonblocking fabric developed for analog telephony. It used the topology of Fig. 3.5(a) in which each active element, or crosspoint, was a single electrical contact. Currently, the term is used to describe any single path nonblocking network that has a complexity that grows as a function of N^2 (where N is the number of input and output ports). From this point of view, the topologies of Figs. 3.4, 3.5 and 3.6 are all crossbar designs, differing only in the use of the input and output ports. The fully interconnected networks of Fig. 3.4 are also known as networks with N^2 disjoint paths [4,5,30].

Since crossbar designs bear a complexity in paths or crosspoints that grows quadratically with the number of ports, they do not scale well to large sizes. They are however, very useful for the construction of nonblocking, self-routing, switching elements and switches of modest sizes. In the literature several switches that use this crossbar architecture have been reported. By interleaving the distribution and concentration stages of Fig. 3.4(a), the Christmas Tree switch uses less than N^2 paths [45]. The Knockout switch is one example of the structure of Fig. 3.4(b) [30]. A matrix

structure with cell buffers in each of the crosspoints is used in the switch designs from Fujitsu, and in the RACE program [26,30].

The banyan network, as originally defined, covered a large class of interconnection networks that had only a single path between any input and output. Currently the term "banyan" is applied to a family of self-routing networks constructed from 2x2 switching elements with a single path between any input and output pair [12]. The banyan network has a complexity of paths and switching elements of order N log N. This makes it more suitable than the crossbar structure (of order N^2) for the construction of large switch fabrics. Unfortunately, the banyan is a blocking architecture and its performance degrades rapidly as the size of the network increases. Noteworthy, the degree of blocking is related to the specific combination of destination requests present in the incident traffic. The performance may be improved if switching elements larger than 2x2 are employed. This leads to the class of delta networks.

Delta networks are self-routing multi-stage interconnection networks with a single path between any input and output. Although the performance of a delta network can be significantly better than that of banyan networks, it is still a blocking network. Its performance degrades as it increases in size, and it is sensitive to the incident traffic pattern [30].

A banyan design will offer nonblocking performance if the incoming cells are sorted in order based upon their output port requests. The batcher network can sort an arbitrary set of cells into order based on their routing tags and grouped the active cells together. Thus the combination of a batcher and a banyan network will offer nonblocking performance if some means is provided to prevent multiple cells from simultaneously requesting the same destination [12,26].

Although the individual 2x2 sorting elements of the batcher-banyan network are simple to implement, a large batcher-banyan network is not easy to partition into integrated circuits and maintaining synchronization across the entire structure becomes

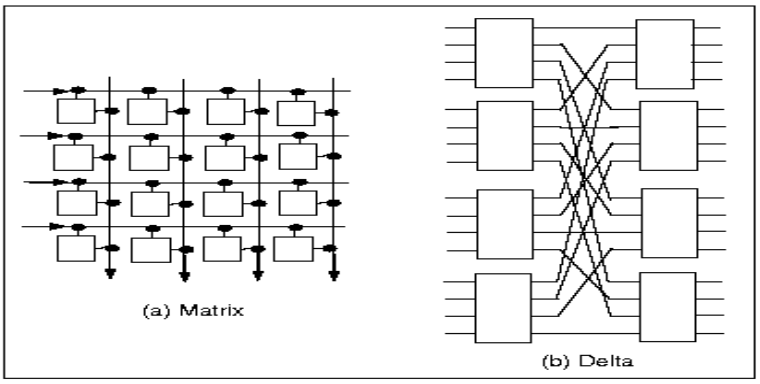

Figure 3.5: Single-path switching architectures based upon Space division (2nd illustration)

increasingly difficult with size. Additionally, the growth of a batcher network is of order N(log N^2), so many more switching stages are required in batcher networks than in banyan networks. The batcher-banyan network is useful for the construction of nonblocking switch fabrics of much greater size than can be achieved with a crossbar design [30].

The batcher-banyan interconnection network was first proposed in the Starlite switch design from AT&T [30]. The construction of a 256x256 Batcher-banyan network (requiring 36 sorting stages in the Batcher network and eight switching stages in the banyan network) has been reported by Bellcore [12,26,30]. It is being implemented in a set of five chips and forms the switch fabric of the Sunshine switch. Most of the interest in the batcher-banyan switch fabric has come from the research community rather than from equipment manufacturers.

Multiple-Path Networks

Multiple-path networks are used to overcome the limitations and improve the performance of a single-path network, or to construct large switches from switch modules. Some of the multiple-path interconnection networks suggested for switch

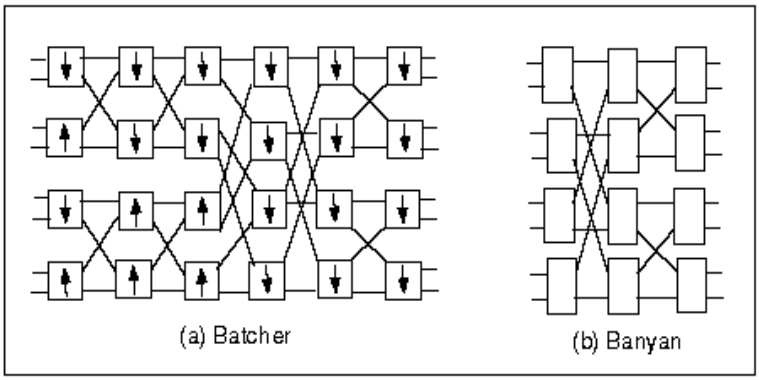

Figure 3.6: Single-path switching architectures based upon Space division (3rd illustration)

designs are: augmented banyan networks, switch planes in parallel, Clos networks, recirculating networks and load sharing networks [30].

In the augmented banyan network, multiple paths are introduced into a banyan or delta network by adding multiple stages of switching elements. The extra stages can be used to distribute the traffic evenly across the banyan (or delta) network, to remove the sensitivity of the network to the incident traffic pattern, and to improve the network's performance.

Switch designs that propose this approach include: Turner's switch [4] and also the MARS switch [31,41]. Extra switching elements and interconnection links may also be added between the stages of a banyan or delta network to provide redundant paths for fault tolerance and to improve the performance.

Another method for introducing multiple paths into an interconnection network is to connect multiple switch planes in parallel. This technique offers both increased reliability and improved performance, since the loss of a complete switch plane will reduce the network's capacity but not the connectivity [3,30].

Fig. 3.8(a) shows the two-sided Clos network. A Clos network built with non-blocking switch modules will have m paths between each input-output pair, and will be strictly nonblocking if $m > 2n - 1$. Fig. 3.8(b) illustrates the Clos network in

the folded form, in which case, all of the interconnection links are bidirectional and cells pass through the first stage of the folded network in both directions. So, the first stage performs the functions of both the first and last stages of the two-sided structure. The Clos structure has been proposed in a number of very large switch designs. The two-sided structure is used in the large switch proposals from Bellcore, the Generalized Knockout switch from AT&T and the NEC ATOM switch. Examples of switch designs using a folded structure can be found in [1] from Alcatel and in [10] from the RACE program. The ATOM switch and the Alcatel projects involve the implementation of large switch prototypes based on a Clos structure of interconnected shared medium or shared memory switching elements.

Another approach for providing multiple paths through an interconnection network of switching elements is to recirculate cells that have failed to reach their destinations. Some recirculating designs require that the size of the interconnection network be increased to accommodate the ports required for recirculation [30]. Other designs allow cells to exit from the network's internal stages [26,41].

Finally, a multiple-path network may be achieved by adding extra paths to interconnect switching elements in the same stage of a banyan (or delta) network while preserving the self-routing property [12,41].

3.5 Queuing Disciplines

A switch's buffering mechanism plays an important role as does a switch's throughput or the type of switching fabric employed. Buffers are needed to hold cells in the event the switch fabric can not process a particular cell to its requested destination. Holding cells in a buffer can increase delay, but not buffering cells (i.e., dropping them) can decrease the throughput. The smaller the requirements for cell de-

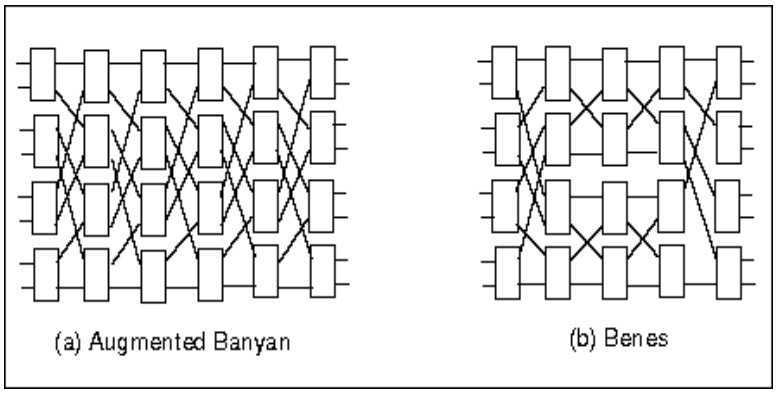

Figure 3.7: Multiple-path switching architectures based on Space division (1st illustration)

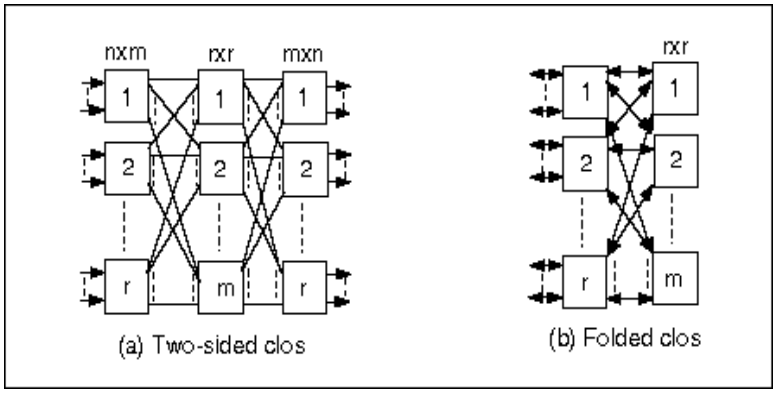

Figure 3.8: Multiple-path switching architectures based upon Space division (2nd illustration)

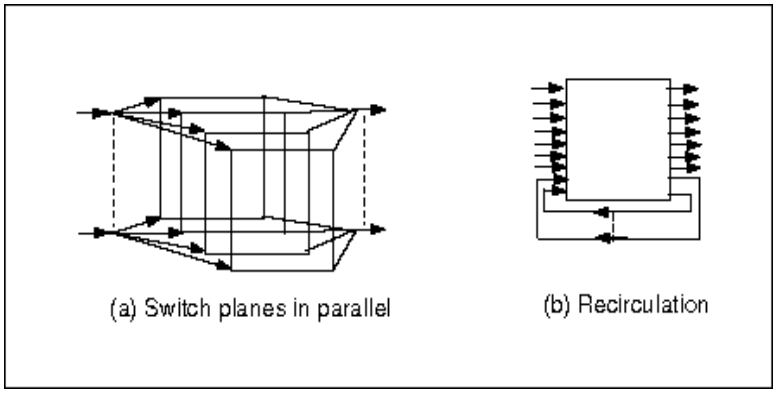

Figure 3.9: Multiple-path switching architectures based upon Space division (3rd illustration)

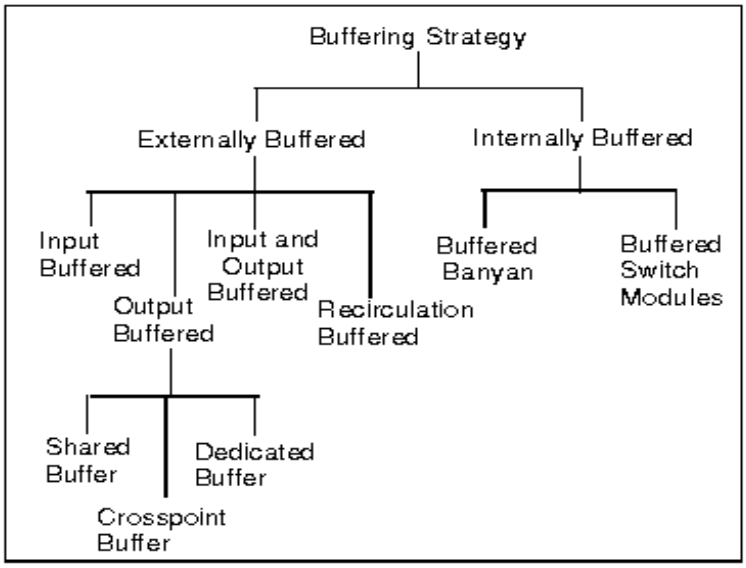

Figure 3.10: Classification of buffering strategies

lay variation (jitter) and cell delay, the smaller the buffers should be. The smaller the requirements for cell loss the larger the buffers should be. Buffers provide a convenient place to apply traffic management schemes to examine and select cells for tagging or dropping. They can be arranged in multiple priority queues according to traffic type. They can also be attributed per port, per group of ports, or one large buffer allocated per connection.

Mainly two different buffering strategies are possible. These are determined by the physical location of the buffers, namely within the switch fabric (internally buffered) or outside of it (externally buffered). *Internal buffering* is usually employed between switch elements of a matrix architecture, while *external buffering* occurs either before or after cells pass through the switch fabric, and thus can exist at input or output ports (i.e., *input* and *output queuing*).

3.5.1 Internal Buffering

In this approach, with cell queues located within the switch fabric, all cells belonging to the same virtual connection must travel the same path across the switch fabric, if sequence errors are to be avoided. This implies that if a multiple-path switch fabric is used, a path across the switch fabric must be selected for each virtual connection at call setup. To do this, the switch must keep a record of the estimated traffic load on each link within the switch fabric and base its call-acceptance decision on the estimated load across each possible path. For a large switch, this will complicate the call establishment process significantly. The alternative is to allow each cell to take any path through the switch fabric and re-sequence the cells on exit from the switch [5].

A switch formed from a single shared memory switch module (Fig. 3.3(a)) may be considered as internally buffered with respect to its physical construction although it offers the performance of an output buffered switch. The shared memory design permits a single buffer to be shared by many input and output ports. This sharing of buffers substantially reduces the number of cell buffers required to support a given performance. This is a tremendous advantage if the cell buffers are implemented within a custom integrated circuit (i.e., there is limited space available for buffers). However, when congestion occurs, the sharing of the buffers between many ports can make it more difficult to locate the source of the congestion by monitoring queue occupancy.

One class of internally buffered designs locates the buffers on the input side of every switching element. These designs generally adopt a banyan (or possible delta) network for the switch fabric and are often referred to as buffered banyan designs [10,31]. In this class of design the switching element is simple to construct, but the performance suffers from blocking within the switch fabric. Internally buffered designs with higher performance are obtained if a multiple-path, multi-stage switch fabric is employed with shared memory or output buffered switch modules [31]. Many large

switch designs are based on this idea.

3.5.2 External Buffering

Externally buffering allows the cell queues to be located close to the switch ports that they serve. Each switch port may monitor its cell queues and perform load monitoring to support congestion control. Furthermore, each queue may be separated easily into multiple classes of service, and each port controller may implement a dynamic scheduling policy based on queue occupancy to best serve the delay and loss requirements of each traffic class. The absence of cell queues within the switch fabric eases the support of multiple levels of priority across the switch fabric to support the different classes of traffic.

In this context, if a multiple path switch fabric is employed, a random selection between the alternative paths may be implemented to distribute the traffic evenly across the switch fabric. Hence, the switch need not keep a record of the estimated traffic load on every internal link within the switch fabric. This considerably simplifies the call acceptance process. Cells on the same virtual connection will not suffer sequence errors, even if they take different paths across the switch fabric, as there are no buffers within the switch fabric.

With the external buffering strategy, buffers can be organized as input, output, shared input, shared output or shared input-output queues.

Input Buffering (Queuing)

In an input buffered scenario, the bandwidth between each input port and the switch fabric, and between the switch fabric and each output port, need only be slightly greater than that of the port itself. This permits the input queues to be located

separately from the switch fabric, simplifies the implementation of the switch fabric, and avoids the necessity for buffers operating at some multiple of the port speed.

Input buffering or queuing is easiest to implement, but it can severely degrade performance due to Head-of-Line (HOL) blocking, where a blocked cell at the Head-of-Queue (HOQ) prevents cells with unblocked paths from proceeding. Nevertheless, the HOL problem can be alleviated by mixing with a carefully chosen output queuing strategy, or with other operations such as windowing, and so does appear on some switches.

Input buffers are more likely to appear on larger switches, particularly ones which have direct interfaces to other protocols. A large nonblocking switch with FIFO input buffers, saturated with uniform random traffic, has a throughput of about 58 percent compared to that of the ideal output buffered switch [5,10,31]. The performance may be improved by a technique known as input queue bypass if access is permitted to other cells in the input queues besides the cell at the HOQ. This technique nonetheless complicates the implementation of the input buffers and requires a more complex contention resolution scheme.

Output Buffering (Queuing)

Output queuing has been shown to be optimal, with shared output queuing providing the optimal delay-throughput performance. Most switches use output buffers, the majority of which are shared. In an ideal output buffered switch every output port must be able to accept a cell from every input port simultaneously. In anything larger than a small switch module, it is unreasonable to expect the switch fabric and output buffers to have sufficient capacity to achieve ideal output buffered operation. Thus, in an output buffered switch of reasonable size, there is always the possibility that more cells will request access to a particular output port than the

switch fabric or output buffer can support. In this case, the excess cells must be discarded.

It is the switch designer's task to ensure that the cell loss probability is sufficient low for all reasonable patterns of incident traffic and acceptable operating loads. An output buffered switch can be much more complex than an input buffered switch because the switch fabric and output buffers must effectively operate at a much higher speed than that of each switch port to reduce the probability of cell loss.

A single-stage, shared memory design (Fig. 3.3(a)) may be considered as output buffered from the viewpoint of its performance. It has a single buffer shared by all input and output ports. A switch with dedicated output buffers has a separate buffer on each output port (Fig. 3.11). Each output buffer is shared by all input ports wishing to access that output port. Output buffered switch designs based upon a matrix interconnection network (Fig. 3.5(a)) use crosspoint buffers. A separate crosspoint buffer is required for each input-output pair resulting in N^2 crosspoint buffers.

Input and Output Buffering (Queuing)

An input and output buffered switch combines the two approaches of input buffering and output buffering. Cell loss within the switch fabric of an output buffered switch due to transient traffic pattern is undesirable. Therefore, instead of discarding cells that can not be handled during the current time slot, they are retained in input buffers. The input buffers need not be large to substantially reduce the probability of cell loss for reasonably random traffic, even at very high loads.

Recirculation Buffering (Queuing)

In this approach, output port contention is handled by recirculating those

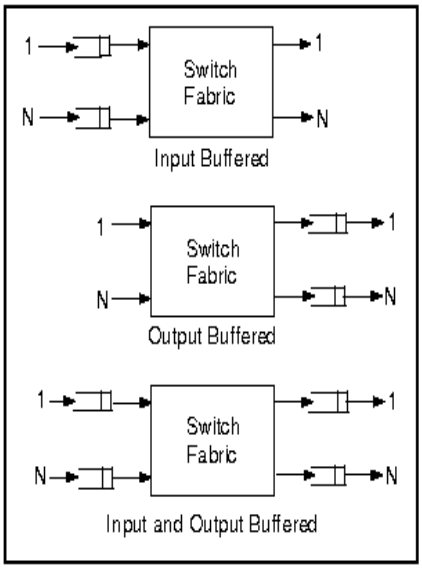

Figure 3.11: Illustration of external buffering strategies with respect to the switch fabric

cells that can not be output during the current time slot back to the input ports via a set of recirculation buffers. It offers the performance of an output buffered switch, but will discard cells in the switch fabric if more cells require recirculation during a particular time slot than the recirculation ports. Recirculation may cause out-of-sequence errors between cells in the same virtual connection unless steps are taken to prevent it.

3.6 Contention Resolution

Given the context of queuing disciplines discussed above, a contention resolution scheme is a relevant design feature to improve the performance of a switch. A simple classification of contention resolution methods is shown in Fig. 3.12 .

In an internally buffered switch, contention is handled by placing buffers at the point of contention, while in an externally buffered switch some practical contention

	Single-Stage Fabric	**Multi-Stage Fabric**	**Broadcast Matrix**	**Time Division Bus**
Fabric Blocking Probability	Zero	Medium	Zero	Zero
Cell Replication Complexity	Fair	High	Low	Low
Scaleability (fabric speed)	Good	Fair	Fair	Poor
Scalability (number of ports)	Poor	Good	Fair	Poor
Buffering in Fabric	Likely	Required	Likely	Unlikely
Input Buffering	No	Possible	Possible	No
Cost to Produce	Low	Medium	High	Low

Table 3.1: Comparison of some switch fabric architectures

resolution approach is needed. Three basic actions can be taken once contention is detected in an externally buffered switch: backpressure, deflection, and loss. Input buffered switch designs typically use backpressure from the point of contention to the input buffers. Pure output buffered designs use a loss mechanism in which cells that can not be handled are discarded at the point of contention. Switch designs based upon recirculation use both deflection and loss. A deflection mechanism will route the cells that lose contention over a path other than the shortest path to the requested destination.

The decisions as to which cells to accept and which cells to reject is made by an arbiter. The arbitration decision may be based on cell priority, a time-stamp associated with the cell, or it may be random. The arbiter may be centralized and implemented externally to the switch fabric, or distributed and implemented internally within each switching component that forms the switch fabric.

Three basic arbitration mechanisms have been proposed: ring reservation, sort and arbitrate, and route and arbitrate. In ring reservation, the input ports are interconnected via a ring which is used to request access to the output ports. For switches

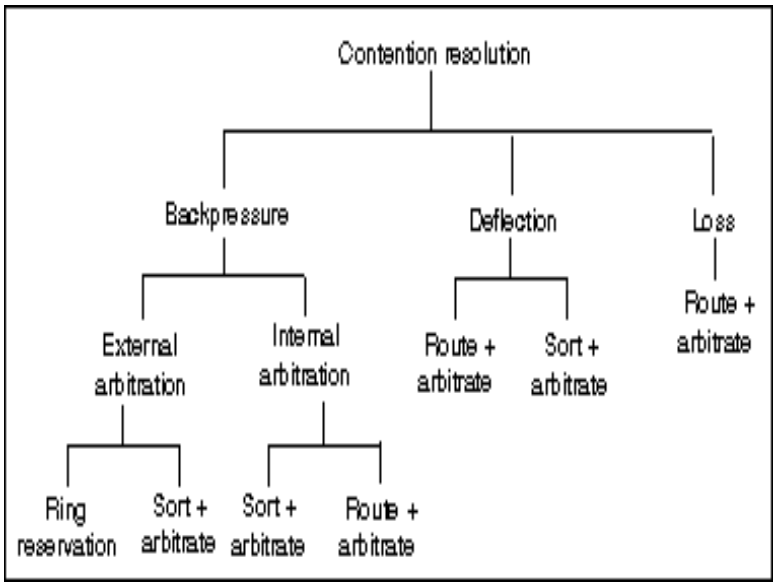

Figure 3.12: Classification of contention resolution schemes

that employ a sorting mechanism in the switch fabric, all cells requesting the same output port will be adjacent to each other after sorting. Thus, an arbitration mechanism may be implemented by comparing the destination requests of each cell to those of its neighbors on exit from the sorting network. In the route and arbitrate approach, cells are routed through the switch fabric and arbiters detect contention at the point of conflict.

spacing1

3.7 Other Issues related to Switch Fabric Architectures

A key factor in switch architecture design is blocking probability. Non-blocking architectures guarantee an absence of internal conflicts. In blocking architectures, paths from input ports to output ports share links between stages. Virtually non-blocking indicates a very small blocking probability in a blocking architecture.

All switches must employ buffer management and traffic management schemes

to compensate for potential cell loss, so a blocking switch will not automatically perform worse than a non-blocking switch based solely on the blocking probability of the architecture. Blocking performance is sensitive to switch architecture and traffic characteristics. A time division-based switching fabric, for example, can be non-blocking provided the aggregate port capacity does not exceed the bus capacity. Space division switch blocking probability depends on the number of paths between switch elements. The more paths, the lower the blocking probability. If there are as many paths between switch elements as there are input ports, then it is non-blocking. Most space-division switches particularly of the campus and enterprise type, are virtually non-blocking.

One important consideration in ATM switch technology is scalability. Scalability refers to increasing the number of ports, increasing the speed of the port interfaces or increasing distances between switches; all with no major architectural or software changes nor significant performance loss. Primarily, users will be concerned with performance and reliability, but issues in scaling and response to traffic characteristics deserve careful attention as well in selecting the optimal switch architecture.

Another design issue in ATM switching, is the maximum number of ATM ports across the ATM switch. Currently, most campus-type switches offer at least 16 ATM ports and the switch providers plan on offering up to 64. However, none have released a switch of that capacity yet. The more ports a switch has, the flatter and more interconnected the network topology can be. More ports also facilitates redundant links for critical connections.

3.8 Performance Characterization

For the performance parameterization of a switch, *switch capacity* and *transit delay* are the characteristics mostly promoted in the literature and the industry. And also since they are similar factors considered in our ALAX switch design, we advocate

them here.

Aggregate switch capacity refers to the maximum number of bits that can pass through the switch fabric per second while aggregate port capacity is the maximum number of bits that all ports on the switch can feed into the switch, with continuous traffic running on all ports at the highest supported data rate. Aggregate switch capacity and aggregate port capacity should be looked at together. If aggregate switch capacity is less that the aggregate port capacity, then the latter becomes the upper bound on maximum throughput. If aggregate switch capacity is greater than aggregate port capacity, this may indicate the possibility of speed scalability of the architecture for faster ports, or it may reveal an architectural limitation that requires increased internal speed.

Transit delay or switch latency is the time it takes for a cell to enter, pass through and exit the switch. Since the speed within VLSI circuits has become so fast and reliable, the dominant delay in the network becomes the switch transit delay. Transit delay is by no means the ultimate measure on throughput, since it does not take congestion into account. However, it establishes a lower bound for total latency over a connection, since the total delay will be at least equal to the transit delay. Measurements taken under different switch load and configurations also have an impact on transit delay measurements as well. Transit delay can be measured as a loop back (in and out of the same port), or from one input port to a different output port. It can also be measured as the delay between individual cells in a stream of cells.

3.9 Summary Note

We have analyzed several architecture designs involved when considering ATM switches. We have outlined the key factors that interplay in the optimal organization of switches. Finally, we have briefly set the grounds on the performance characteristics

of a switch.

With respect to ALAX, the switching architecture of the fabric conforms to the space division model, as described earlier. The switch topology follows a crossbar format which is non-blocking in nature while the switch routing property abides to the self-routing technique. A combination of input and output queuing is favored, along with a backpressure contention resolution scheme.

Chapter 4

The ALAX System: Architecture, Organization and Preliminary Design

This chapter defines and describes the ALAX system. It discusses functional, organizational and operational issues pertaining to the logical and hardware design.

The scope of the material presented extends only from the architecture to the preliminary design of the ALAX switch. These technical aspects of ALAX [1]are also discussed in the documents titled, "Design of an ATM LAN Access Switch Based on the IEEE P1355" and "ATM LAN Access Switch (ALAX): Protocol Architecture (LAN Emulation Version)"[22,23].

4.1 Purpose and definition

The primary interest that pushed the research for the development of ALAX is the IEEE 1355 or P1355 serial bus standard [14]. Using this new standard, and specifically devices built around it, we sought to introduce new approaches and devise novel strategies for an ATM switching fabric. Hence, in the name ALAX, A refers to ATM, L stands for LAN, A implies Access and the X symbolizes the crossbar switch

[1]The system architecture and design of ALAX involved major contributions and efforts from Dr. Kim and Mr. Man-Geun Ryu.

fabric upon which this switch is built. The purpose of ALAX consists primarily of providing an interface between legacy LANs and ATM networks. While investigating the optimal design for this switch, some significant constraints include low system integration cost and compliance with emerging standards.

Being an ATM LAN edge device, ALAX encompasses all the packet relaying and routing functions needed to ensure reliable and efficient end-to-end communication between the respective ATM and LAN network interfaces (ports). It possesses also the capability to forward packets between LANs. However, the only version developed thus far is for interconnecting with the IEEE 802.3 Ethernet LAN types. Packet conversion, fragmentation and reassembly between the ATM and the Ethernet worlds are accomplished using the ATM Forum LANE standard version 1.0. At the heart of the ALAX system resides the SGS-Thomson C104 fast packet routing switch (STC104) [16], a communication device based on the IEEE 1355 Bus Standard. This is essentially the switching fabric that forwards and routes packets across all the ports of the ALAX switch.

From a functional point of view, ALAX maintains a hub architecture to allow interface ports for other applications to be easily integrated in the future by inserting the plug-in modules in question. This is consistent with a modular design approach. The fragmentation and reassembly of packets inside the network interfaces of the switch is performed with T9000 transputers. This leads to a parallel and distributed architecture, and by the same token, a concurrent processing capability within ALAX.

4.2 System Level Architecture and Organization

The internetworking differences between legacy Ethernet and ATM is resolved within ALAX by employing commercial off the shelf (COTS) components. Thus,

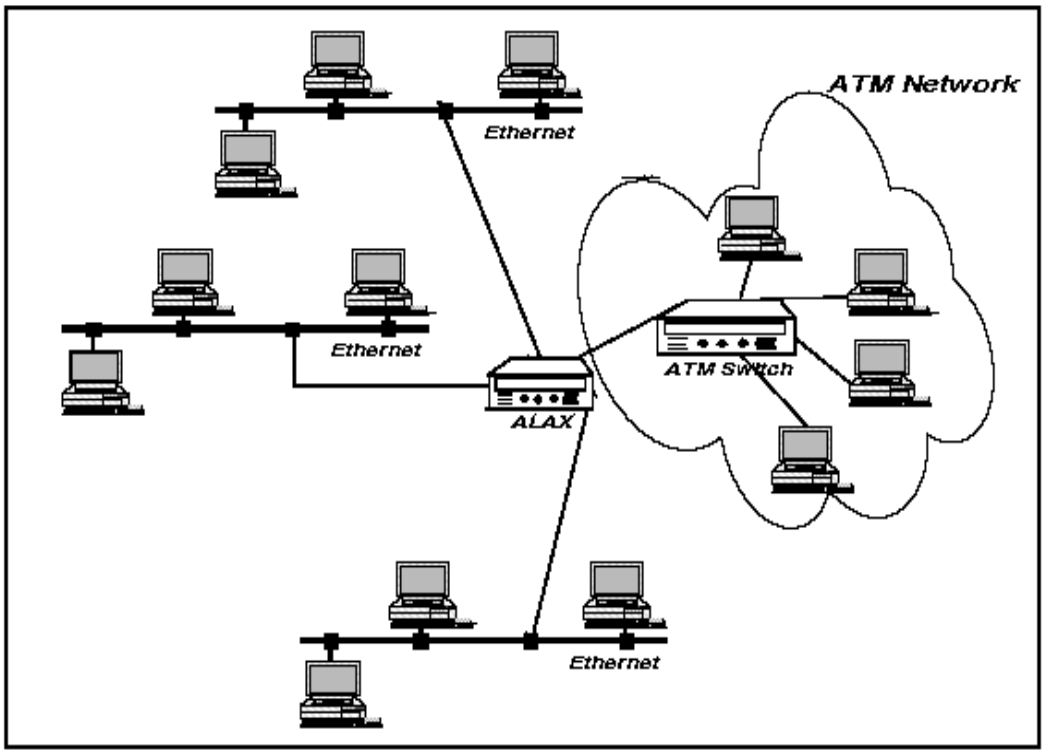

Figure 4.1: Operational context currently envisioned for ALAX

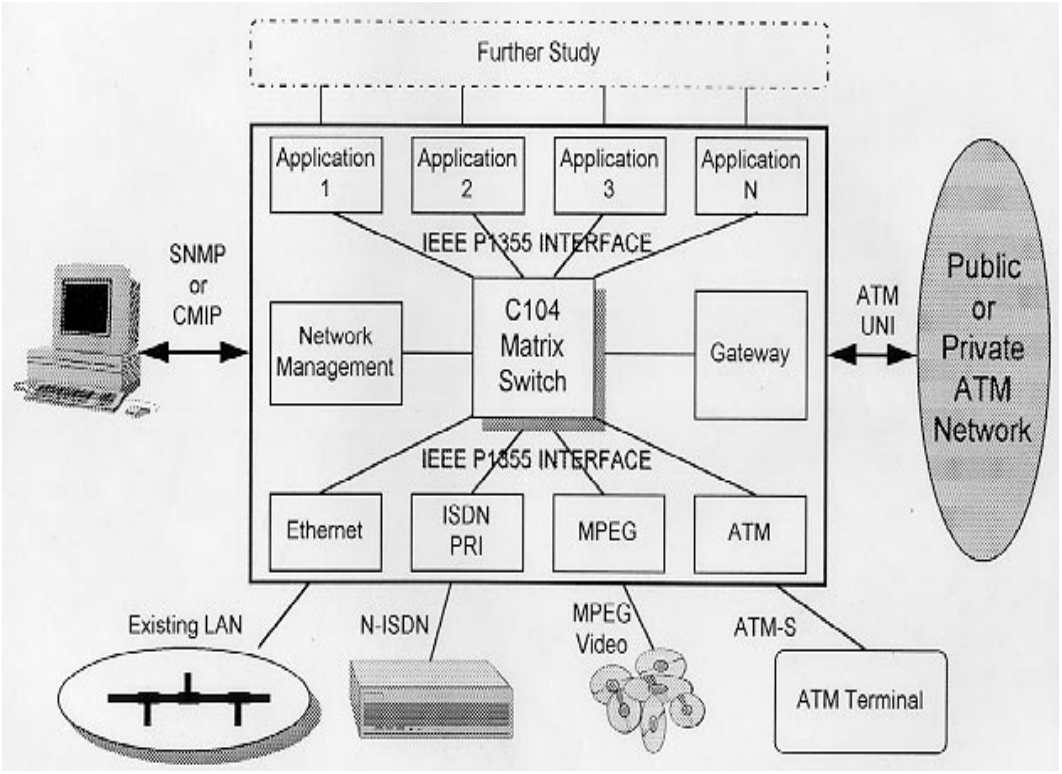

Figure 4.2: Overall system architecture for the ALAX switch with some proposed interfaces (reproduced from [22,23])

ALAX complies with an open system architecture requirement. This edge switching device is designed to operate in conjunction with an IBM PC or compatible platform across the PCI bus. The ALAX switching device alone constitutes only one integral component of the overall system.

There are other subsystems which participate in the operation of this access switch. The complete ALAX system comprises primarily a Graphical User Interface (GUI), an operating system, main control and network management functions, all of which reside on the PC's main CPU, device drivers, the PCI bus and the ALAX switch per se. The purpose of these subsystems is to facilitate the coordination, control and monitoring of the operations of the ALAX switch. The whole system can basically be visualized as a PC with the ALAX switch subsystem being a peripheral for the interconnection of networks.

The main control functions include the system initialization (e.g. start up, booting, etc.), status reporting and reconfiguration. They are required to manage all the interface modules or adapter boards installed within ALAX. The commands issued by these functions and the responses in return will be transmitted via the PCI bus. The system control processor on the ALAX switch is responsible for the interaction between the main CPU of the IBM PC and the interface modules (network ports). So, the PCI bus will be used essentially to provide communication paths between the system control processor of the ALAX switch and the main CPU of the IBM PC. Additionally, the system control processor will perform control of the interface modules (i.e., adapter boards) through its IEEE 1355 communication links.

All these functionalities mentioned above will be integrated within a Graphical User Interface (GUI) display at the desktop for the convenience of the network manager or operator. Thus, the GUI will be the end-point control interface to oversee the operation of ALAX. While developing this GUI, the principles of good interface design, taking into account human factors, prior knowledge and experience of the user

will be observed.

The network management tool for the system will most likely be SNMP, this should also be part of the GUI integration design. Since, the application of ALAX is intended for a PC environment, the network operating system of choice is primarily Windows NT/Windows 95[2]. There will probably be versions of ALAX for each of these operating systems. Finally, NDIS drivers will be distributed across the interface modules.

Figures 4.5 and 4.7 show the networking protocol architecture environment where the current prototype version of ALAX will be implemented. Observe that, the ALAX switch incorporates three main and independent generic subsystems: ATM-IEEE 1355 interface module, LAN-IEEE 1355 interface module and the IEEE 1355 switch fabric. These devices implement the lower layers in LAN and ATM networks, and there are a bridging relay along with LANE to connect these layers [23]. It is worth noting also that there is a packet conversion from IEEE 1355 format into MAC format and vice versa, which occurs at this bridging relay. This is named MAC Mapping Layer (MML) and is crucial to guarantee transparency between the IEEE 1355 and MAC layers. MML was developed in-house by the LAST research team at the University of Maryland. It represents a sub-layer intrinsic to the switch's internal operation.

In summary, The protocol components required for the design of ALAX in the user plane are then LANE, Bridging relay function, IEEE 1355, MML, MAC, LAN physical, AAL5, ATM and ATM physical. Some other protocols necessary within the control plane, but which are not being addressed in this document, are the ATM signaling and Network Management (i.e., SNMP) (see summary in Fig. 4.3).

[2]Software Products by Microsoft

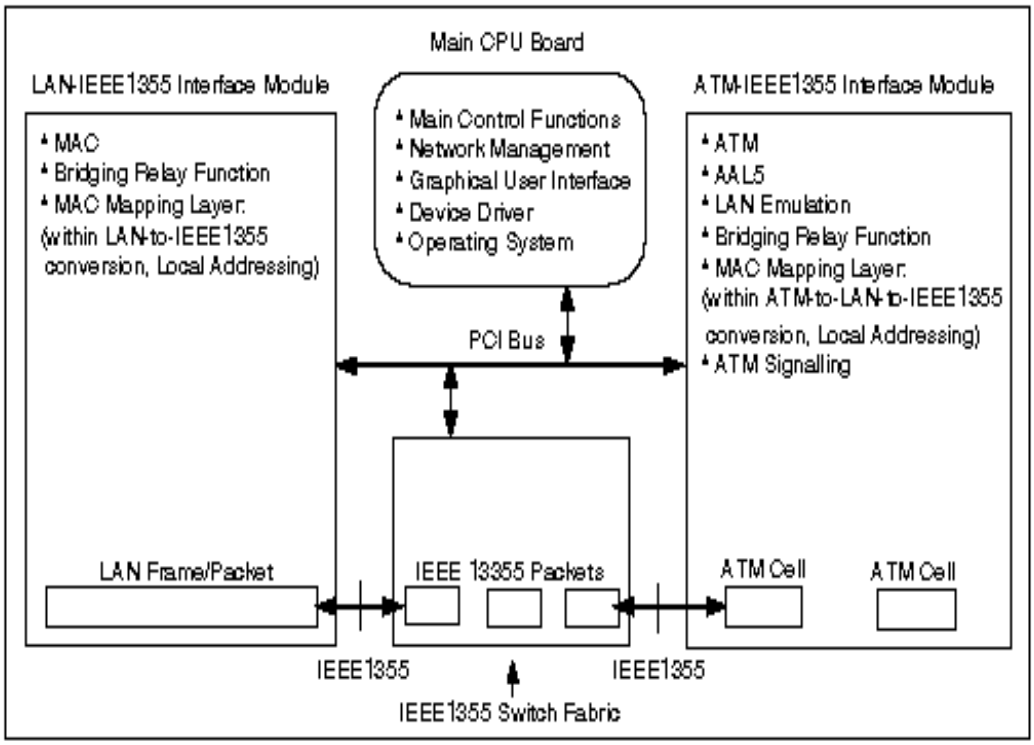

Figure 4.3: ALAX in LAN Emulation environment with the control and transport protocols (reproduced from [22,23])

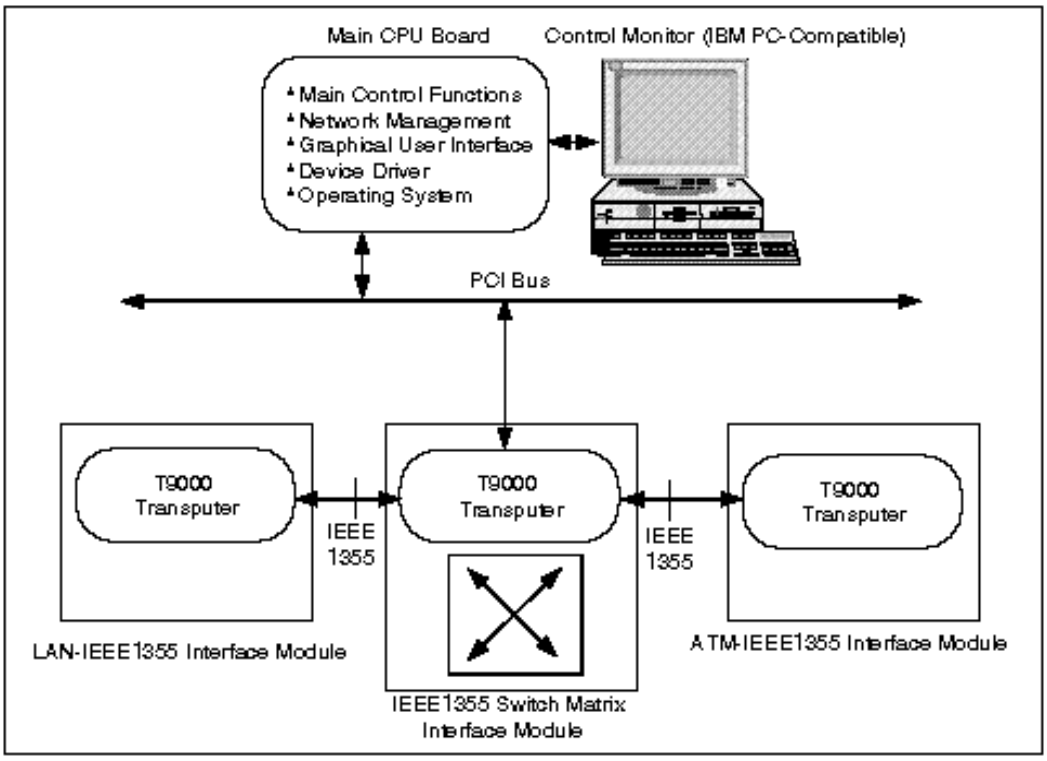

Figure 4.4: Flow of control functions in the ALAX (reproduced from [22,23])

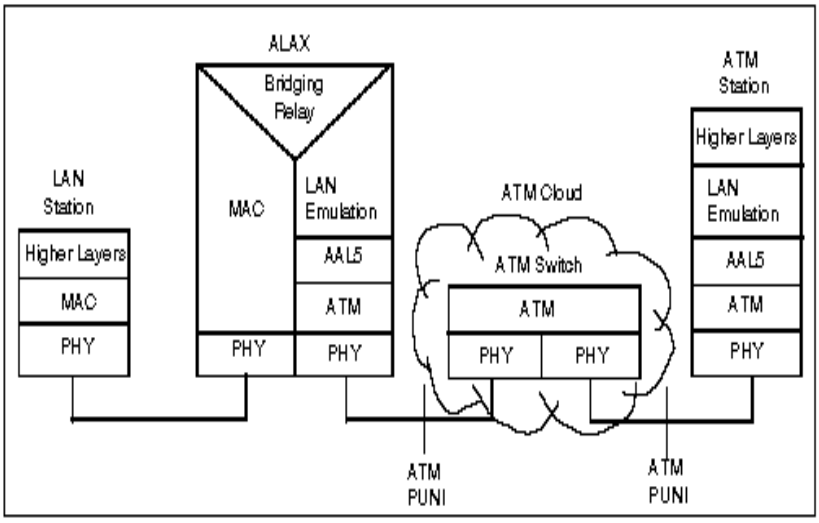

Figure 4.5: Protocol architecture environment where ALAX will be set up

4.3 Key Functionalities inside ALAX

Before delving into the system design layout of ALAX, we first take a brief look at some inherent functionalities that differentiate ALAX from other comparable access switches. Aside from the adoption of a multiprocessor and modular design approach, we can cite the ATM LAN integration solution, the internal packet conversion and the data transport mechanisms. Studying these features, provides the opportunity to clarify some concepts alluded in the previous paragraph, while allowing a smoother assimilation of the system design characteristics.

4.3.1 ATM and LAN Integration

The ALAX platform relies on LANE version 1.0 (see Chap. 2) to assure that ATM can be incorporated transparently into an Ethernet-based network with existing hardware and physical media. LANE as deployed in ALAX is not so distinct since we are complying with the standards. The LEC can be (another T9000 on) any of the Ethernet ports. The other entities, such as LES, LECS and BUS, were not precisely determined at this point of the preliminary system design. But, based on recent updated information[3], we expect these functionalities to be implemented remotely.

4.3.2 Packet Conversion

Packets in ATM and LAN network environments respectively are different in both formats and sizes. Hence, within the ALAX switch one of the main tasks will consist in translating packets from one network port to the other. In fact, these packets will be further converted into the IEEE 1355 format to maintain consistency with the IEEE 1355 switching fabric implemented in the C104 chip. Effectively, what this amounts to is: frames from the LAN side with the IEEE 802.3 Ethernet format

[3]Status updates on the ALAX prototype received from Dr. J. Kim, in Korea.

are converted first into IEEE 1355 format and then into ATM cells, before being sent out to the ATM network side. A similar process occurs in the opposite direction. However, for LAN-to-LAN communications, IEEE 802.3 packets are put into IEEE 1355 format, and then back into IEEE 802.3 format before being forwarded to the appropriate LAN destination.

The packet conversion process inside the ALAX involves several notions such as addressing, routing, fragmentation and reassembly. Fig. 4.6 illustrates the packet conversion operations that take place within the ALAX switching system. One key step necessary in the packet conversion at both types of network interfaces (i.e., ATM and LAN) is the MML (MAC Mapping Layer). On the LAN side, this protocol simply facilitates addressing between the switching fabric and the LAN port, while on the ATM side, the packet which results from the MML process is LAN emulated first (i.e., LANE is performed on the packet) before being mapped into the appropriate ATM cells and vice versa.

The main functions of MML are to *perform local addressing* and *conversion to IEEE 1355 packet format* (Fig. 4.8). Local addressing provides the means to extract the local address from the MAC address. A Local Addressing Look Up Table (LALUT) is maintained. It contains all the local destination port addresses (local with respect to the switch fabric). This local address is then matched one to one with the port number of the C104 switch fabric chip so that the packets can be routed to their requested destination ports. The conversion into IEEE 1355 format is necessary because all the packets that traverse the switch fabric must be in that format. This requires fragmentation of the MAC packets into IEEE 1355 packets at the incoming DS links and re-constitution of the entire packets at the outgoing DS links of the destination ports.

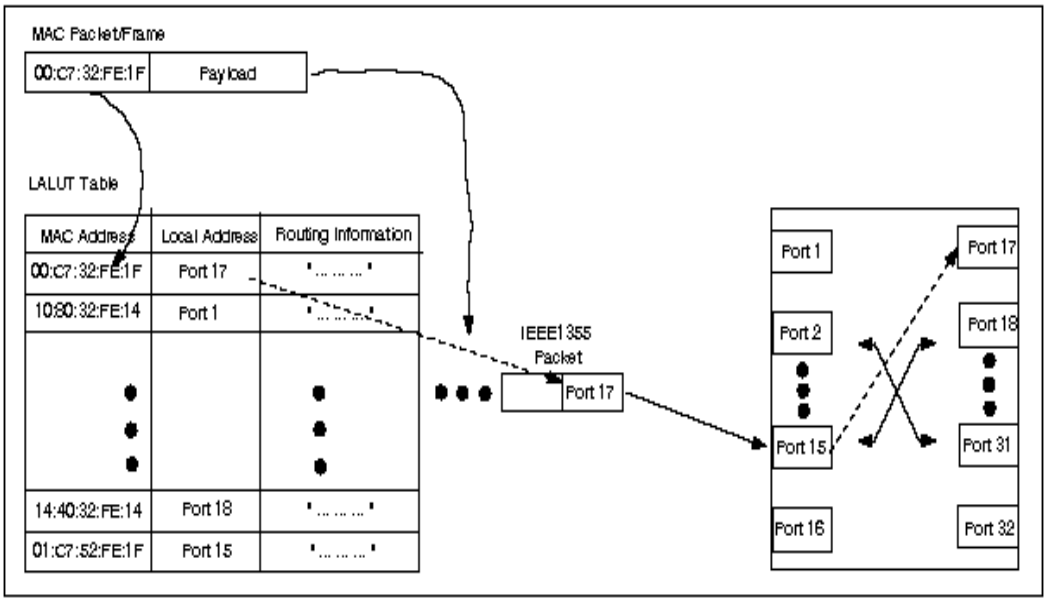

Figure 4.6: Local Addressing and conversion to IEEE 1355 within MML (reproduced from [22,23])

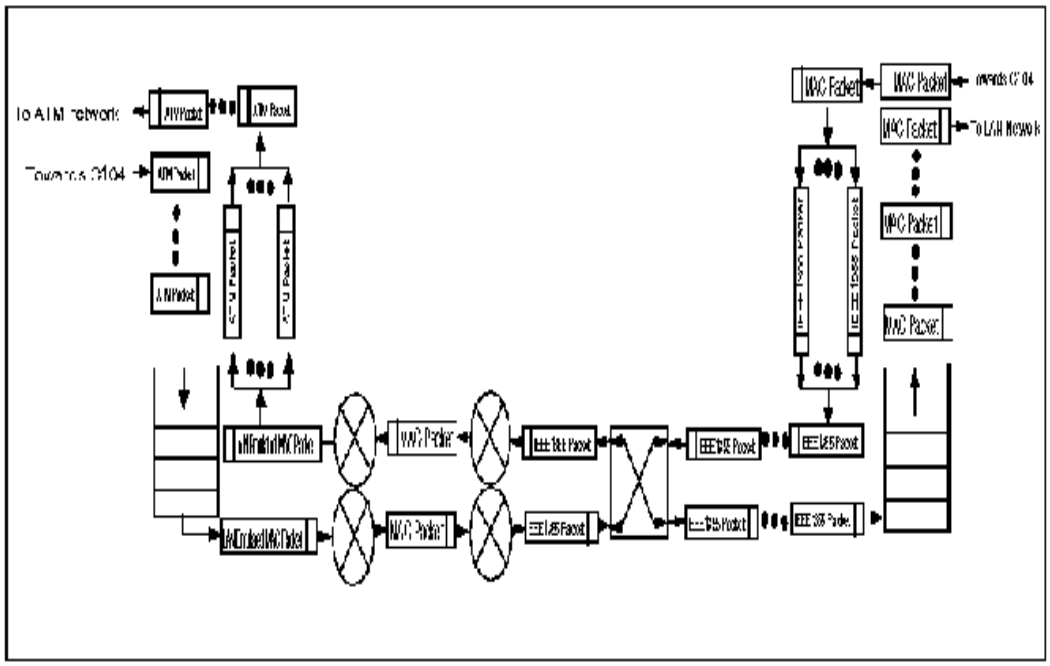

Figure 4.7: Packet translation/conversion within ALAX

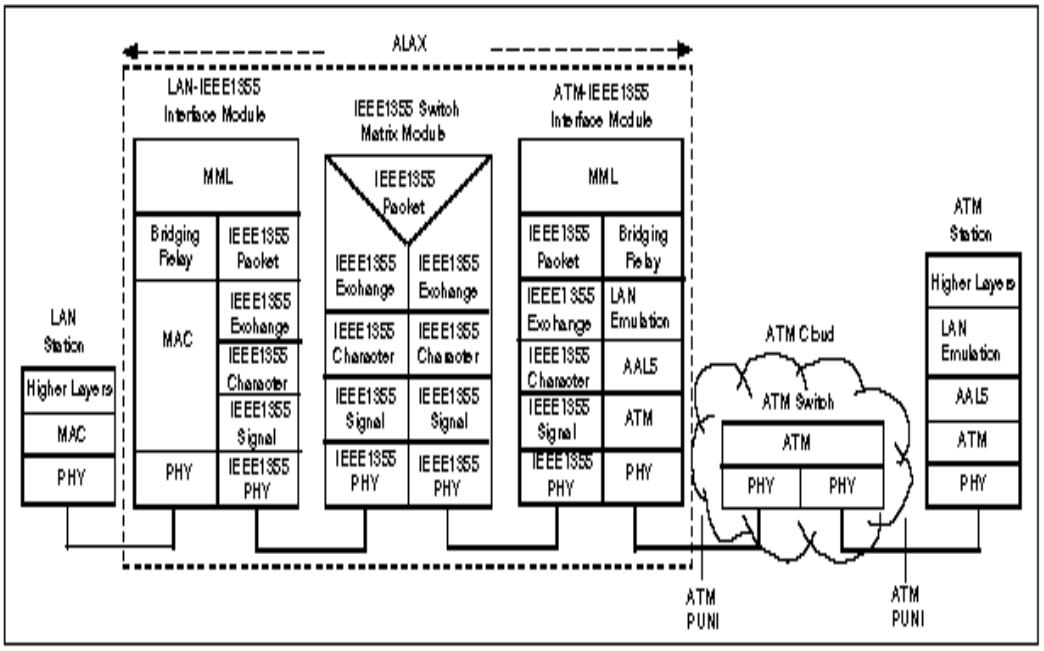

Figure 4.8: A detailed look at the data transport protocols implemented in ALAX

4.3.3 Data Transport with the IEEE 1355 (P1355) Serial Bus

ALAX relies on the fast data bus of the IEEE 1355 to transmit packets or signals between the T9000 transputer and the C104 chipset. IEEE 1355 is a standard that has been developed through the works of the ESPRIT OMI/HIC Project. This standard was proposed "to complement recent technical developments of highly integrated, low power interconnect technology implemented in high volume commodity VLSI processes" [14,16] for chip-to-chip and board-to-board communications and "to exploit the simplifications in encodings and protocols resulting from the use of relatively reliable media over relatively short distances"[14,16].

The motivation for this new serial bus standard finds its roots in the search of modular devices for constructing high-performance systems with parallel or distributed communications. Such a construction generally must be fast and low-latency, otherwise it will be a limiting factor in system performance, and it must be low-cost or

else it will dominate the system cost. It must also scale well in both performance and cost relative to the system size, otherwise highly parallel systems will be limited in performance or too expensive. IEEE 1355 was strategically devised to meet these criteria whereas existing standards could not.

IEEE 1355 is appealing since it enables high-performance, scalable, modular, parallel systems to be built with low system integration cost. In addition, it supports communications system fabric, provides a transparent implementation of high level protocols and can effectively create links between heterogeneous systems.

Two bi-directional link protocols were born out of these IEEE 1355 or P1355 research initiatives: a 100 Mbps Data-Strobe (DS) link and a 1-3 Gbps High Speed (HS) link [14]. ALAX is based on the DS link protocol. This DS link protocol consists of a data and strobe signals (Fig. 4.9). The data line carries the data while the strobe line only changes state when the data remains constant. In this protocol the clock is encoded, enabling autobauding at the receiver and asynchronous links. There are four levels or layers, a Bit level at the lower end of the physical infrastructure, and on top of the Bit level, the Character, Exchange and Packet levels.

The Character level PDU is a group of consecutive bits (characters) used to represent data or control information. The Exchange layer describes the exchange of characters to ensure proper function of a link. In particular flow control characters are used to enable traffic flow from the link sender. This guarantees that the switching fabric is lossless: no characters are lost internally due to buffer overflow. At the Packet layer, a packet defines a sequence of characters with a specific order and format: a header, which contains routing information, a payload containing zero or more data bytes and an end of packet marker. The protocol does not indicate a specific packet size. Messages are sent through the network as a sequence of packets.

A family of communication devices have been developed to support the DS link protocol, these include a parallel DS link converter (C101) and an asynchronous packet

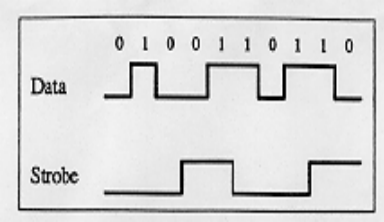

Figure 4.9: The DS Link protocol

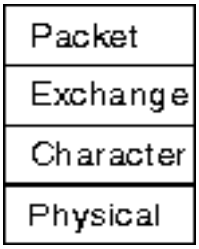

Figure 4.10: The IEEE 1355 protocol stack

routing chip (C104) by Inmos SGS-Thomson. The DS link is also used by the T9000 transputer.

Having identified the activities in ALAX for packet conversion and data transport, we are now well positioned to approach and clearly understand the design structure of ALAX. The next section summarizes the design characteristics of ALAX as found in [22,23].

4.4 System Design

The proposed hardware set design to implement the logical and functional attributes of the ALAX switch is partitioned into three major components as indicated earlier: the ATM-IEEE 1355, the IEEE 1355 switch matrix and the LAN-IEEE 1355 adapter boards. Because the T9000 transputer and the C104 chip play the key functions inside the switch, we will describe them first and then proceed with the other system components.

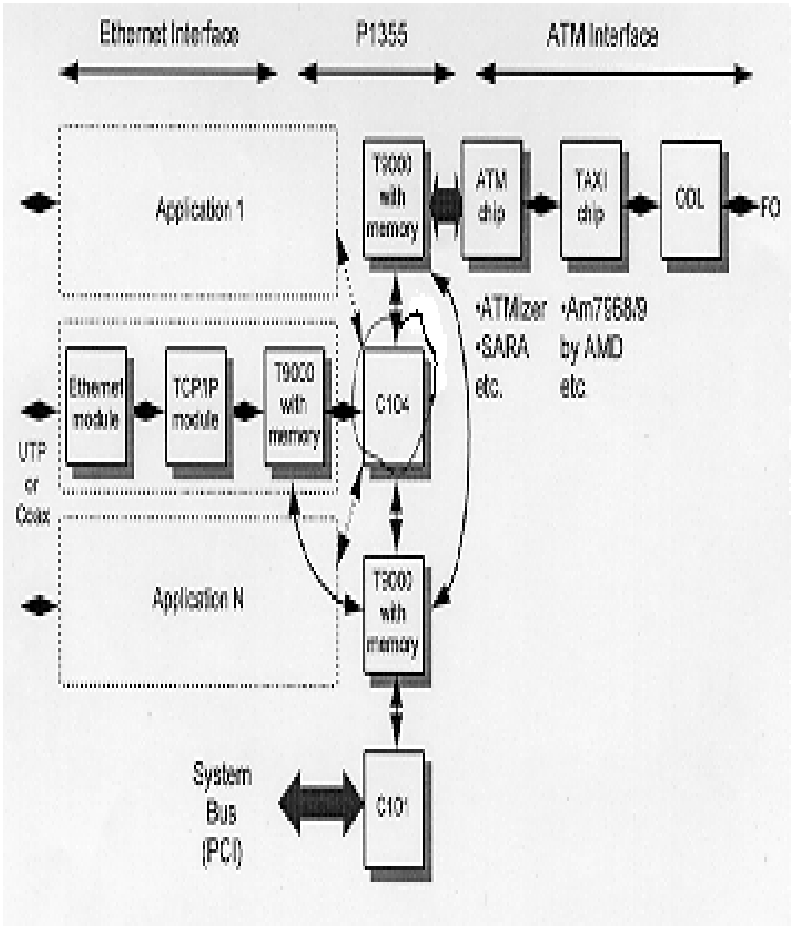

Figure 4.11: Block diagram representation of the entire ALAX system hardware design (reproduced from [22,23])

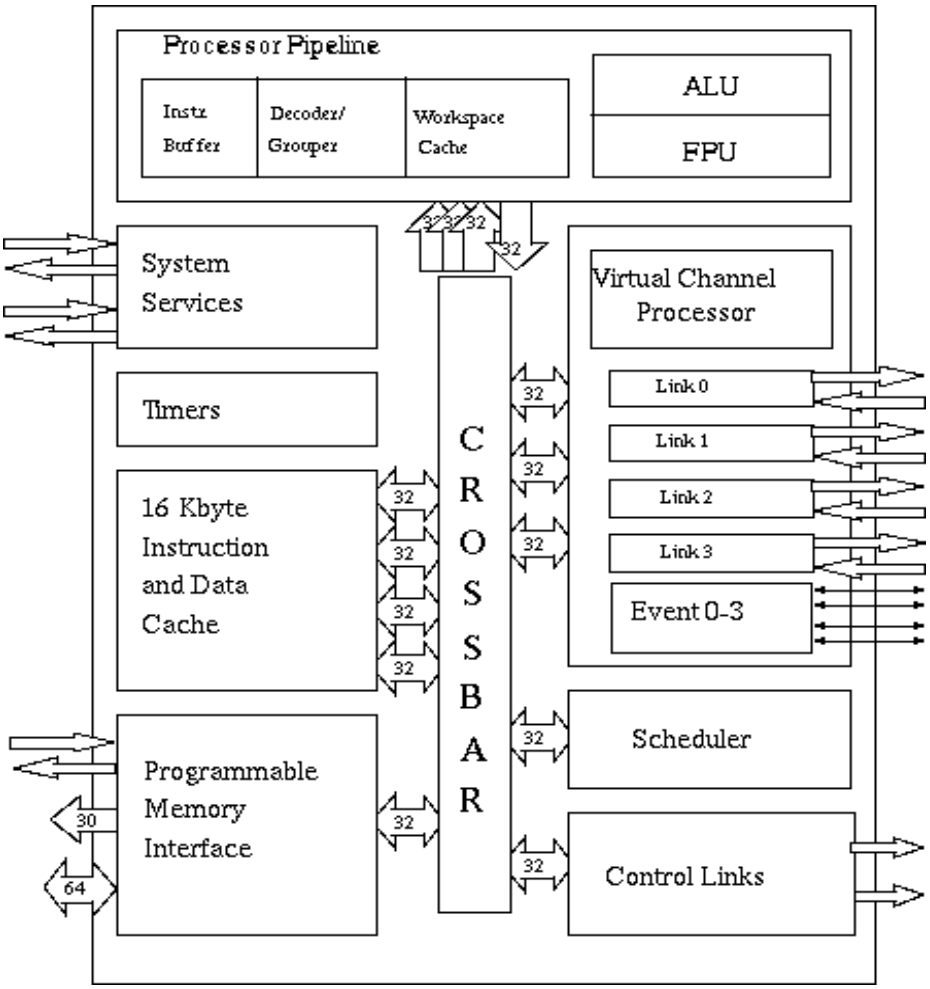

Figure 4.12: T9000 functional block diagram (reproduced from Inmos SGS-Thomson Technical Data Sheet)

4.4.1 The T9000 Transputer

A transputer is a complete microprocessor integrated in a single VLSI chip. Unlike conventional computers which execute programs sequentially, the transputer executes programs concurrently which means that many steps of the program can be run at once. This device can be used on its own, but its forte is in multi-processor applications. It is easy to run many transputers in parallel, and performance increase is directly proportional to the number of processors used. This is not the case with conventional processors. Internally, the transputer uses reduced-instruction-set computer (RISC) architecture. This allows high performance on a small silicon area. The average operating speed is about 10 million instructions per second [28].

The T9000 is the second-generation of transputers from INMOS. It has a 32-bit pipelined processor with a 64-bit FPU and 16 Kbytes of cache. There are four bi-directional serial data links and an autonomous Virtual Channel Processor (VCP) allowing efficient T9000-to-T9000 communications [28].

The T9000 has several improvements over the previous generation of transputers in both performance and functionality. Improved performance has been gained through an increase in clock speed (50 MHz design), the implementation of an on-chip cache and a pipelined super scalar architecture. Improved communication is also provided by the new Data-Strobe (DS) link technology (mentioned earlier). Messages in the T9000 are divided into a sequence of packets of the same format as the C104 chip (see next section). All routing information is contained in the packet header. The T9000 can handle a maximum packet body of 32 bytes. Any device receiving a data packet replies to the sender with an acknowledgment packet. No further packets are transmitted by the sender until the corresponding acknowledgment response has been received by the sender.

Communication between T9000 processes is performed via virtual links. A virtual link is defined as a single logical communication connection between two processes

mapped onto a physical processor link. The VCP of the T9000 is a hardware communications processor which multiplexes the virtual links onto a specified processor link. Packets from separate virtual links are interleaved onto the physical link, allowing separate processes to communicate simultaneously. The virtual link to which a packet destined is identified as a field in the packet header. T9000 processors can be directly connected using their DS links or through a network of C104 packet routing chips, thus enabling the construction of large networks with scalable communication bandwidth between nodes [28].

4.4.2 The C104 Asynchronous Packet Routing Chip

The SGS Thomson C104 (STC104) is a complete, low latency, packet routing switch on a single chip, that connects 32 high bandwidth serial communication links to each other via a 32 by 32 way non-blocking crossbar switch. The links operate concurrently and the transfer of a packet between one pair of links does not affect the data rate or latency for another packet passing between a second pair of links. The packet latency across the C104 has been measured to be $\approx$1 microsecond [16,28]. The nominal link speed of 100 Mbps allows a maximum bi-directional bandwidth across the chip of approximately 300 Mbytes/s [16].

To enable packets to be routed, each packet has a header at the front which contains routing information. The router on the C104 uses the header of each incoming packet to determine the corresponding output link. Anything after the header is treated as the packet body until the packet terminator is received (i.e., EOP (End of Packet) or EOM (End of Message) token).

The C104 uses wormhole routing, in which the routing decision is taken as soon as the routing information, which is contained in the packet header, has been input. Wormhole routing functions essentially in a similar fashion as the self-routing

cell mechanism of some space division switching architectures (see Chap. 3). The algorithm that makes the routing decision in the C104 is called interval labeling; it is deadlock-free, inexpensive and fast [16]. Each link on the routing switch is labeled with an interval of possible header values, and only packets whose header values falls within that interval are output via that link.

Hence, as a token stream is received on the DS link, it is passed to the Packet Processor and interpreted as a sequence of packets. Each packet is either output through one of the 32 links of the crossbar switch or discarded. If the specified link of the header is not busy, the packet is transmitted immediately without being buffered. If the output link is busy, as much data as possible will be buffered before data flow is stalled until the output link becomes available. The C104 uses a token flow control method to regulate the data flow across the links. This flow control technique approximates to a backpressure scheme in case of contention (See Chap. 2). The benefit of the token level flow control lies in the simplification which it implies, and guarantees for higher-level protocols. It prevents data from being lost due to buffer overflow and so removes the need for re-transmission unless errors occur.

This token level flow control is performed in each link module of the C104. It is a mechanism which prevents a sender from over-running the input buffer of a receiving link. Theoretically, each receiving input link contains a buffer for at least 8 tokens. However, 20 tokens of buffering are actually provided for the input link whereas 50 tokens for the output link. So in essence, the C104 combines both input and output buffering design approaches. Whenever the input link has sufficient buffering space available for a further 8 tokens, a flow control token (FCT) is transmitted on the associated link output, and this FCT gives the sender permission to transmit a further 8 tokens. This sender waits until it receives another FCT before transmitting anymore tokens.

We can identify the basic hardware contents of the C104 as follows (see Fig. 4.15)

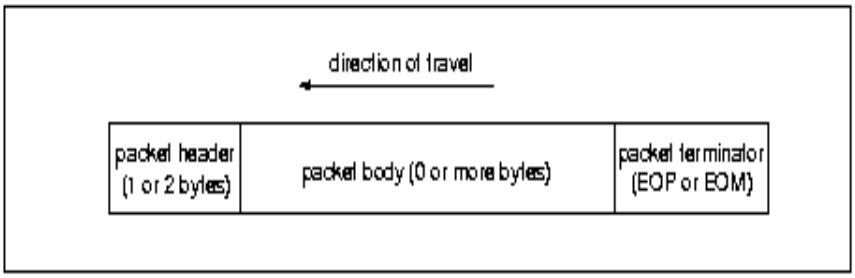

Figure 4.13: Packet structure in the C104 chip (reproduced from Inmos SGS-Thomson Technical Data Sheet)

:

1. IMS C104 Packet Routing Switch Object Module
2. D-S(Data-Strobe) Link Functional Module
   (a) Packet Processor Sub-Module
      i. Interval Selector Sub-Sub-Module
      ii. Random Header Generator Sub-Sub-Module
      iii. Header Buffer Sub-Sub-Module
      iv. Header Stripper Sub-Sub-Module
   (b) Data Link Sub-Module
3. Command Processor Module
4. Output Crossbar Switch Module
5. Control Unit Module
   (a) Control Link0 Sub-Module
   (b) Control Link1 Sub-Module

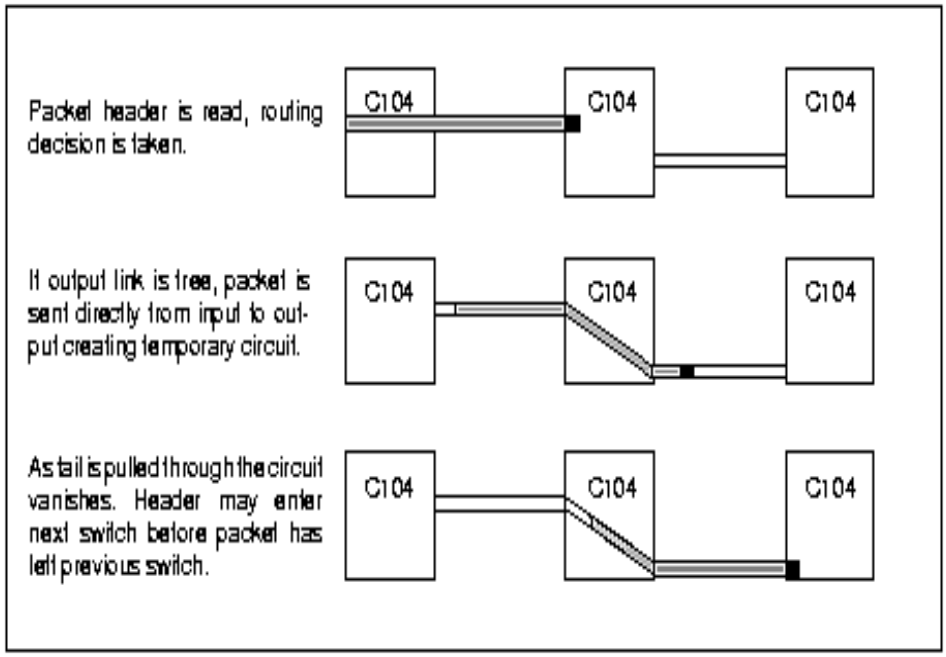

Figure 4.14: Wormhole routing in the C104 switch (reproduced from Inmos SGS-Thomson Technical Data Sheet)

4.4.3 ATM-IEEE 1355 Interface Module (Adapter Board)

This module holds by far the largest level of complexity proportionally to the extensive set of tasks that it is expected to perform within the ALAX switch. One end of this module receives and sends ATM traffic, while the other end receives and sends IEEE 1355 packets. As the schematic indicates (see Fig. 4.16), this module consists of a T9000 transputer (in original preliminary design there was 1 T9000, current design includes two instead; we will see why later), an ATMizer chip, a TAXI chip, the ODL interface and some shared memory buffer. These building blocks will combine their functions to achieve packet conversion and other related services, discussed earlier, for the ATM port of the ALAX. The ODL chip reflects the multi mode fiber connector standard of 100 Mbps. The TAXI chip provides the means of connecting parallel data over serial links, whereas the ATMizer introduces the B-ISDN layer.

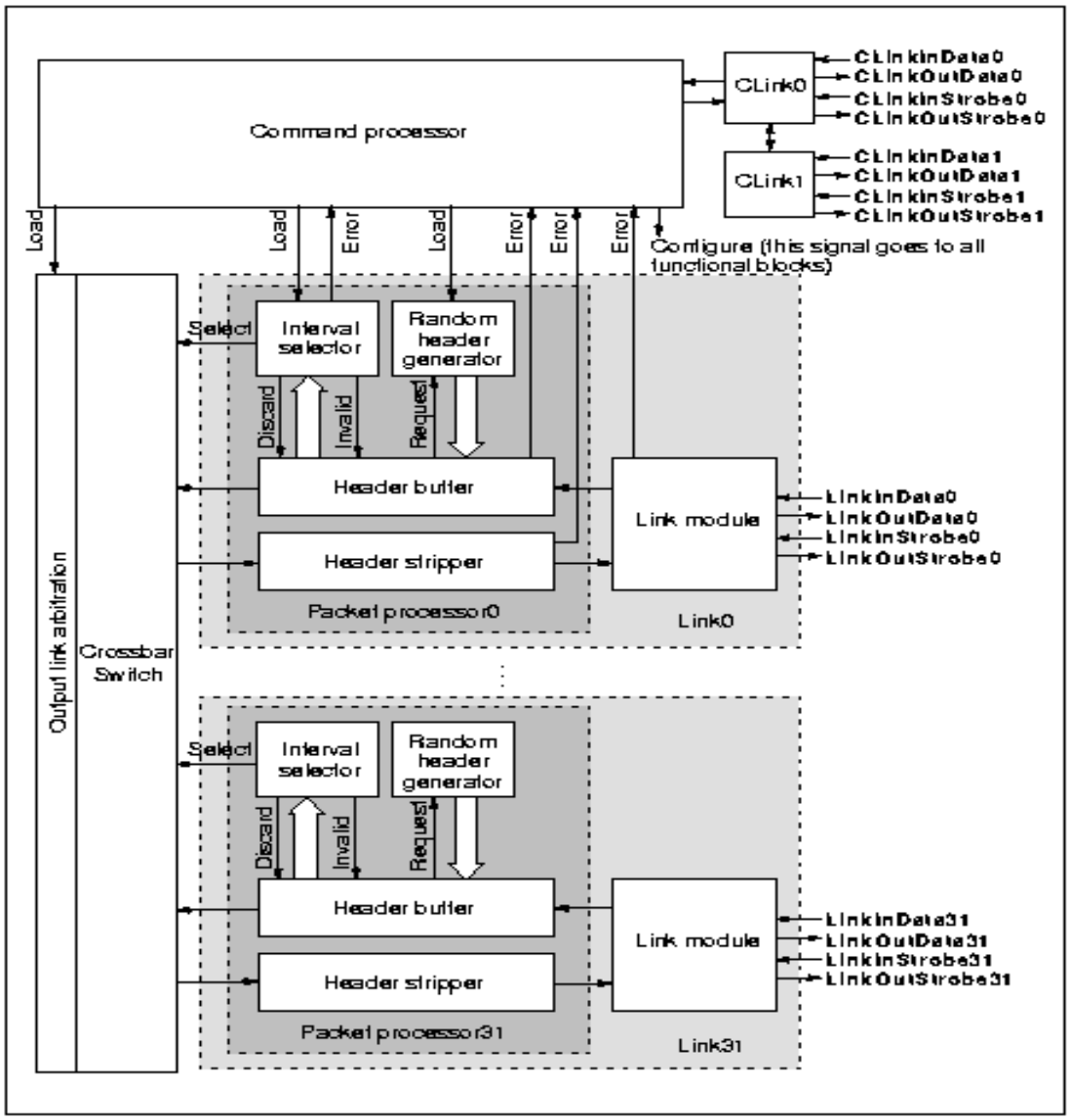

Figure 4.15: C104 functional block diagram (reproduced from Inmos SGS-Thomson Technical Data Sheet)

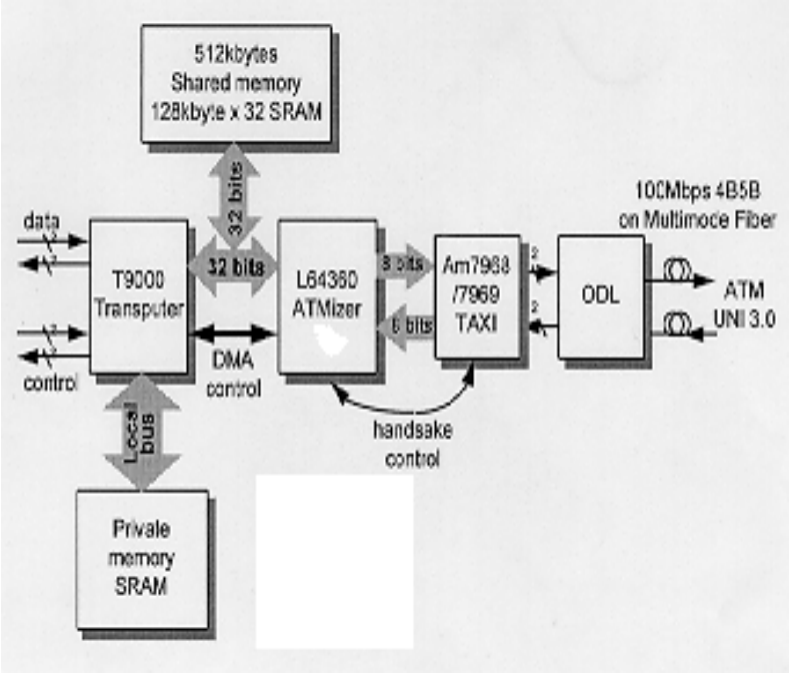

Figure 4.16: Block diagram representation of the ATM-IEEE 1355 interface module (reproduced from [22,23])

4.4.4 IEEE 1355 Switch Matrix Interface Module (Adapter Board)

This module includes the C104, a T9000 transputer, a PCI Bus controller chip and some buffer memory. The T9000 is assigned the task of controlling the C104 and its related DS links. This T9000 represents the system control processor as defined in section 4.2. It is connected to the control links of the C104, and interacts with the main CPU of the IBM PC, required for the entire ALAX system, through the PCI bus controller. It sends information and receives commands destined for the ALAX switch (i.e., configuration, status report, etc.). The memory mentioned here is required only for management purposes. The C104 divides equally its total number of DS links into the data and control paths for the T9000s of each interface module that it is connected to. In other words, ALAX can support up to 16 data and 16 respective control ports across the 32-by-32 way C104 switch. Because each T9000 transputer of an interface module will use two DS links of the C104 chip, one for sending and receiving data

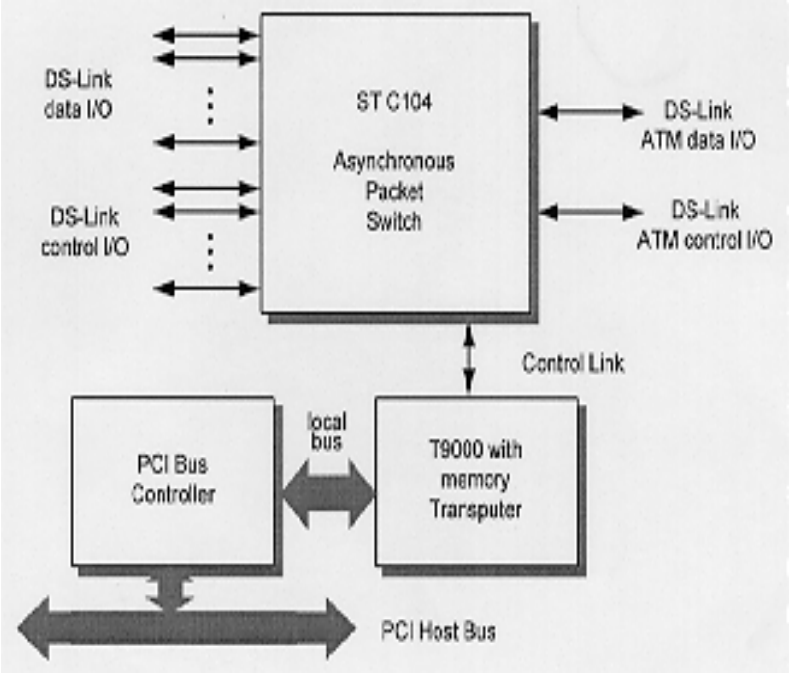

Figure 4.17: Block diagram representation of the IEEE 1355 switch matrix interface module (reproduced from [22,23])

and the other for sending and receiving control information from the system control processor towards a particular interface module or port.

4.4.5 LAN-IEEE 1355 Interface Module (Adapter Board)

The block diagram for this interface module (Fig. 4.17) shows a standard Ethernet interface connector, the LLC chip, the T9000 and some buffer memory. The level of complexity in this board is of a smaller scale compared to the ATM one. The T9000 bears the same functionalities as previously described, except that now the packets are only LAN frames. The functionalities are included also for maintaining a Local Addressing Look Up Table (LALUT).

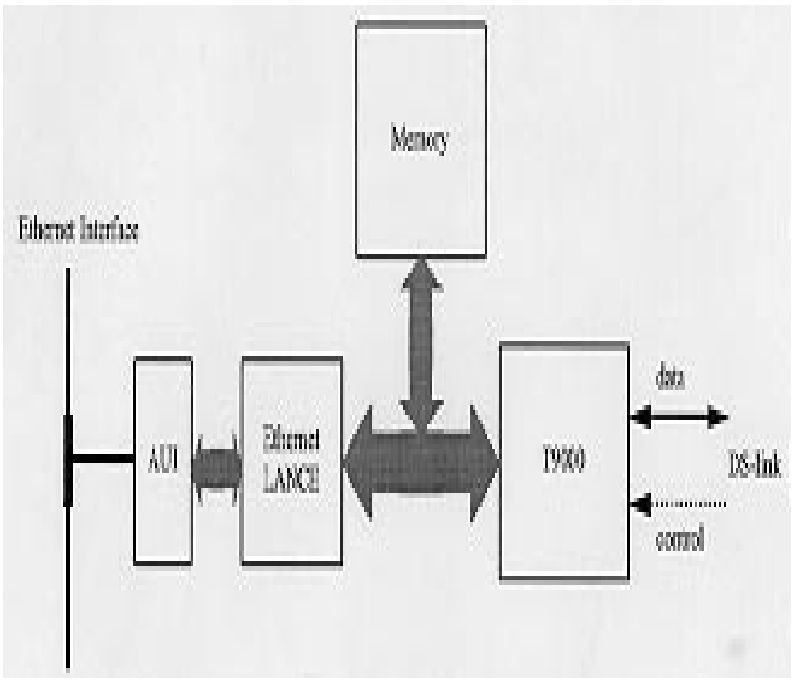

Figure 4.18: Block diagram representation of the LAN-IEEE 1355 interface module (reproduced from [22,23])

4.4.6 Specifications Underlying the System Design

In summary, the system design for ALAX can be characterized as follows:

1. ALAX must be built using COTS components.

2. The system design must be simple and compliant with current and emerging standards, and interoperability practices.

3. ALAX uses one C104 switching fabric.

4. ALAX can support up to 16 data network ports, e.g. 15 Ethernet ports and one single ATM port.

5. Even though, the C104 has 32 DS links only half of them are actually used for transporting data, the other half is used for control signals of the interconnected T9000 processors at the interface modules.

6. T9000 transputers are distributed among the interface modules or network ports of the switch (i.e., each module has its own T9000 processor). Thus each interface module is independent, and this provides the ease of partitioning for building scalable system and adding more modules as the need arises.

7. At the ATM port, the T9000 transputer performs LAN Emulation, ALAX local addressing (i.e., MML) and other higher ATM layer functions such as multi protocol process, Common Part Convergence Sub layer function in AAL and signaling function.

8. At the LAN port, the T9000 transputer performs the MML function and LAN Circuit Emulation (LANCE).

9. The T9000 maximum message or packet size is 32 bytes, so the maximum packet size in the C104 is constrained to the same size.

10. T9000's functions across all the ports require some buffer memory to carry and maintain the packet conversion.

4.5 Summary Note

ALAX has been defined and presented from its early conception to the actual preliminary design. It embodies a simple communication system platform with superior performance to support the needs of speed increase and demand for dedicated bandwidth across an ATM network. In addition, ALAX reflects a cost-effective solution since on the one hand, it utilizes COTS parts and, on the other hand, protects investments made in existing LANs by interfacing primarily with IEEE 802.3 Ethernets, the most dominant LAN type in use. Noticeably, several design characteristics of the ALAX system distinguishes it from other existing or emerging systems: the dual adoption of the IEEE 1355 standard, and the multi-processor approach using the C104

and T9000 hardware components respectively; and last but not least the modularity of the overall switch architecture.

One other major task of the ALAX project besides the physical construction of the switch prototype, requires the determination of design parameters to guide the construction. In the next chapter we specify the design parameters of interest and outline the strategies and approaches adopted to obtain them.

Chapter 5

The ALAX Simulation Model

This part of the thesis focuses on ALAX from the simulation modeling angle. Simulation modeling is one of the most crucial aspects of the ALAX project to which the LAST research team at the University of Maryland has been devoting the bulk of their time.

Inside the modeling section, we define and introduce the purpose of simulation. We establish the rationale and motivation for using simulation in developing the ALAX system, and also the goals to be met by simulating the ALAX switch design. Then, some selected simulation languages and packages utilized for modeling of systems are discussed. And with some engineering judgment calls, we choose the simulation tool appropriately suited to meet the needs of the ALAX system design. Finally, the simulation model of ALAX is thoroughly described and explained, along with all its related components.

5.1 Introduction

An engineer would like to predict the performance of a given switch design architecture without first having to build it. This prediction is what is referred to as *performance evaluation* [15,16,23]. In order to accomplish it the performance analyst has commonly two approaches (Table 5.1). One approach requires making use of *ana-*

lytical or statistical models of switch behavior to determine such essential performance measures as throughput, time delay, utilization, blocking, loss probability and buffer occupancy. The other approach is *simulation*, and more precisely discrete-event simulation [15,17,28].

The analytical approach relies on the tools and techniques of queuing theory. Compared to a discrete-event simulation, analytical models, once developed, are usually solved quickly. And, more importantly analytical models give insight into the operation of switches and networks. Nonetheless, the ALAX switch architecture at hand has already been determined with the chosen switching fabric and the necessary protocols to support the ATM LAN internetworking communication (Chap. 4). Thus, our interest at this point lies not so much in getting insight into the operation of the switching architecture, but instead we seek to produce a numerical evaluation for supplying design parameters in order to build a prototype of the switch.

Henceforth, we take the road to simulation because we are concerned with a preliminary assessment of the feasibility of the ALAX system design, validating and verifying the protocol architecture conceptualized for the switch and, possibly optimizing this developed protocol-based communication system. Simulation is often favored as a viable method due to the risk of high cost that can be incurred in prototyping different scenarios or configurations of a particular system, especially when the system is novel and fairly complex. Consequently, a simulation model for ALAX will provide an easy and cost-effective way to study and predict the performance of the system.

Generally speaking, simulation implies the imitation or representation of the operation of a real-world system over time. It generates numerical measures to draw inferences pertaining to the operating characteristics of the real system that is represented. Simulation prevails as an indispensable problem-solving methodology for the solution of many real-world problems such as to describe and analyze the behavior of a system, ask "what if" questions about the real system, and aid in the design of real

Criterion	Analytical Modeling	Simulation Modeling	Measurement
Stage	Any	Any	Post-prototype
Time Required	Small	Medium	Varies
Primary Resources Required	Analysts	Simulation Tools	Instrumentation
Accuracy	Low	Moderate	Varies
Trade-off Evaluation	Easy	Moderate	Difficult
Cost	Small	Medium	High
"Saleability"	Low	Medium	High

Table 5.1: Performance Evaluation Techniques and Criteria for Selection (reproduced from [17])

systems.

5.2 Overview

5.2.1 Simulation Design Analysis Discussion

Looking back at the ALAX system design (Chap. 4, Sec. 4.4), several components can clearly be identified as congestion points or bottlenecks in the process of transporting data from one end to the other of the ALAX entity. They are primarily the C104 switch and the T9000 transputer. Consequently, from an engineering-economic angle, these components ought to be configured and designed in a cost-effective manner with adequate resources to support the intended network environments. Otherwise, they could dramatically affect the cost per port for ALAX, and by the same token the final cost of the ALAX switch. This is significant, for we must not depart from our earlier projection that ALAX must be a low-cost platform.

The C104 should obviously be considered since it is the point where data traffic converges and decisions are made for forwarding packets to their required destinations. It is one of the stages where a data packet can suffer substantial delays or can literally

be lost. Additionally, The T9000 handles several functions that can cause also overall delays on the ALAX switch: LAN Emulation and possibly, MML. These functions in order to be achieved efficiently and effectively must be supplemented with sufficient buffering. Furthermore, given that the T9000 will be the intermediary for forwarding packets to the C104, again in this case adequate buffering must be provided to protect against packet loss for the worst-case of congestion that may occur at the C104.

5.2.2 Simulation Objectives

Henceforth, the basic goals of the ALAX simulation activities are:

1. to study the performance tradeoffs associated with crucial design decisions (e.g., number of transputers)

2. to provide a means for testing, evaluating and verifying the performance of our proposed ALAX design at the functional level, hence identifying strengths and weaknesses

3. to investigate issues related to potential bottlenecks inside the ALAX switch

4. to determine the *end-to-end packet delays* and the *acceptable (memory) buffer sizes* to perform the ATM LAN internetworking process defined in chapter 4

5. also, another interesting parameter that necessitates careful investigation, is the *optimal processing speed of the T9000* particularly at the ATM port where LAN Emulation along with the MML function will be performed.

Thus, on the one hand, the simulation model for ALAX will provide answers to some specific unknown design parameters, while on the other hand, it will generate a simple model for the protocol architecture of the switch to supplement the system design.

In general, when building any model, it is important to choose the correct level of detail commensurate with the scale of questions that the model is expected to answer.

This is sometimes referred to as the granularity of the model. At this step of the modeling process, caution must be observed while analyzing and making the decisions, as this will greatly influence the degree of success of the simulation. For instance, if not enough details are included, then some important aspects of the system's behavior may be omitted, or the precision or accuracy of some targeted measurements may be flawed. In contrast, if too many details are taken into account, this would require investing a great deal of effort for researching and implementing the details. Consequently, a model which is larger than needed may result and this implies longer times to run each experiment. Eventually, the decisions to be made call upon the experience and the engineering judgment of the analyst.

In our case, since the objectives to be achieved in building a simulation model of ALAX are already established, the scope of the ALAX simulation is to a great extent scaled down and simplified. Because, we know where to emphasize and put weight in our simulation design. This reduces the complexity of details that would be involved if the entire switch and its components were modeled.

5.3 System Modeling Tools

In order to create a simulation model of the ALAX switch, several tools are available. The tools introduced in this section are packages which are widely used and readily available off the shelf. They consist of either simulation languages or network simulators for developing simulation models of communication systems. To end up with an adequate simulation tool, we explore several avenues:

1. the selection of a simulation language
2. the selection of the best network simulation package
3. the tradeoff of developing a simulation model in the simulation language or package selected

The simulation languages investigated here are FORTRAN, SIMSCRIPT II.5 and SIMAN. The network simulation packages considered are COMNET and OPNET. These languages and network simulation packages are all briefly described throughout the following paragraphs. Some other simulation tools are also discussed in [16,23].

5.3.1 Simulation Languages

FORTRAN (FORmula TRANslation) is a structured programming language widely used for simulation because of its processing power. Many mathematical computations can be done quickly through the use of FORTRAN.

In spite of this, many experts in the field would concede that FORTRAN is practically extinct. It is still used, but many other languages have been developed which have been confirmed more efficient than FORTRAN. Newer languages also perform much better than FORTRAN, because of the lines of code necessary to create the same function (e.g., C, C^{++}). FORTRAN has a lot to offer, but functions such as random distribution and status reports have to be coded. Other structured programming languages have more to offer, hence FORTRAN is seldom used currently.

SIMSCRIPT II.5[1], developed by CACI Products company, is a powerful, free-form, English-like discrete-event simulation language that greatly simplifies writing programs for simulation modeling. Programs written in SIMSCRIPT II.5 are easily read and maintained. They are efficient, and generate adequate results.

Unlike other simulation programming languages, SIMSCRIPT II.5 requires no coding in other languages. It has been in existence for over 30 years now, and its wide acceptance and success can be attributed to the design format of the language which consists of *Entities* and *Processes*. This provides a natural conceptual framework that easily relates real objects to the model. A well-designed package of debugging facilities is also included. Simulation status information is provided, and control can

[1]Developed by CACI Products

optionally be transferred to a user program for additional analysis and output. Many modifications, such as the choice of set disciplines and statistics are simply specified in the *Preamble* block of the SIMSCRIPT program. Moreover, SIMSCRIPT can be integrated with the SIMGRAPHICS package to create advanced GUI displays and develop icons for the animation of the simulation process.

In summary, SIMSCRIPT II.5 can effectively be used for several types of simulation which extend from single queue and server to complex communication networks.

SIMAN[2] (SIMulation ANalysis) on the contrary, is a general-purpose procedural language for modeling combined discrete-continuous systems. The SIMAN language allows simulation programs to be entered using either batch input statements or through an interactive mode. CINEMA is the associated animation package which provides for constructing animation objects, linking them to the SIMAN model, and jointly executing the simulation and animation. Arena is a new modeling environment which allows models to be constructed through a GUI. Within the Arena environment, the SIMAN and Cinema code is automatically generated. Models can be run (with the animation) from within the Arena environment. Thus Arena is an environment which is layered on top of SIMAN and that integrates the CINEMA functionality.

SIMAN models are defined by two text file components which are compiled and used by the simulation "engine" to perform the simulation, the *Model Frame* and the *Experimental Frame*. The *Model Frame* embodies the structural elements of the model, its components, the layout, etc. The *Experimental Frame* defines the parameters, the random number seeds, the number of replications, and the data collection components. The intent is to isolate the model data from that characterizing individual experiments. Thus, different experiments can be performed without altering the model.

[2]SIMAN is developed by Systems Modeling Corp.

Tradeoff Analysis

Among the simulation languages discussed above, SIMSCRIPT was the primary choice. This can be attributed partly to the fact that the SIMAN block structures are most appropriately conceptualized and suited for manufacturing and assembly line systems, while on the contrary SIMSCRIPT offers a much greater generality and flexibility for the prospect of representing any type of systems. Nevertheless both languages are limiting since they can not conveniently and efficiently characterize the context of communication systems. We will see then, what some of the selected network modeling tools can offer in contrast to these simulation languages.

5.3.2 Simulation Network Modeling Tools

COMNET III [3] (current version of COMNET) is CACI's next generation of network planning tool integrating LAN and WAN performance prediction in a single, object-oriented environment. COMNET III's hierarchical design fits equally to all networks from single high performance LANs to some complex, enterprise-wide systems, existing or planned.

With COMNET III, the performance of proposals can be verified, "what-if" analysis tested, competing designs evaluated, and the operation of a proposed network can be observed using the built-in graphics and extensive reporting. No programming is required. Predefined hardware and protocol objects make model description fast and easy. The objects can also be customized to some degree in order to suit specific needs.

COMNET III shows an animated picture and dynamic graphs while the simulation is running. Based on observation, parameters can be changed and, links and nodes can be taken down with the effect of such changes immediately apparent. A COMNET III model may contain hierarchically defined subnetworks. The designer can observe the

[3] Developed by CACI Products.

entire subnetwork in animated form. COMNET III models data transport protocols as well as the scheduling of applications on end systems, and the use of processing, storage, and transport resources during application execution.

OPNET[4] (OPtimized Network Engineering Tools) on the other hand developed by MIL3,is a comprehensive and extensive network modeling system capable of simulating large communications networks with detailed protocol modeling and performance analysis. The features of OPNET include among other things, graphical specification of models, a dynamic event-scheduled simulation kernel, integrated data analysis tools and hierarchical object-based modeling.

OPNET deals with the problems of distributed algorithm development through its hierarchical modeling structure to describe and control the network to be analyzed. This includes from top to bottom: the *Network*, *Node* and *Process* levels. The *Network* level consists of network nodes that can be connected by means of unidirectional or bidirectional communications links. Within the *Node* level, there are queue and processor modules which handle packet generation, termination, storage and routing. The distance between them can be specified, and this extends to link delays at the network level. The function and operation of these modules in question can be altered by modifying their respective finite state machine (FSM) within the *Process* level. This provides a lot of flexibility to model any type of communication node.

The FSM provides the means to process and operate the communication modules of the *Node* level. Specifically, the FSM through self generated interrupts, that model event transitions from state to state, can collect packets off the communication stream, read pertinent information from the packets, perform other processing operations on the packets and forward them on outgoing stream links. Inside a state of the FSM, a programming code written in $pseudo-C$ ($Proto-C$) can be included to execute some of the operations mentioned earlier. In addition, the *Analysis*, *Tracing* and *Anima-*

[4]Developed by MIL3

tion capabilities of OPNET help the designer in synthesizing, interpreting the output data, while monitoring the network behavior. It is worth also adding that the OPNET environment comes rich with built-in models of networks, such as X.25, Frame Relay and ATM, suitable for design of current and emerging technologies. These built-in models can be altered to conform to a specific custom design, and they are all based on approved specification standards.

Tradeoff Analysis

Among these two network environment tools evaluated, OPNET was picked. This decision was based upon the fact that, since ALAX is considered complex due to the novelty of some of its protocols, it becomes imperative to utilize a package that will grant as much flexibility and generality as possible, to simulate the protocol operation in anticipation of uncertainties that may arise later. Consequently, if we were to use COMNET for instance, we could have been limited with the scale of modeling framework and libraries available on this package, which can not be modified for instance. But, OPNET has more to offer since programming can be accomplished to the extent appropriate as the need emerges. Thus, it brings a large set of capabilities in our hands to create the ATM LAN access switch, ALAX.

Indeed, OPNET models are very fast and memory-efficient compared to COMNET models at a similar level of detail. While OPNET has been successfully utilized to model networks as large as 17000 individual nodes, anecdotal evidence would indicate that COMNET is ineffective for modeling network larger than a few dozen nodes. High-speed networks for example, produce so many events that simulation with COMNET is prohibitive.

Furthermore, modeling exercises are iterative in nature, requiring successive implementations of more and more detailed models before all of the data important to

the specific network or technology being studied is generated. The OPNET model hierarchy (*Network*, *Node*, and *Process*) allow lower levels to be initially modeled in a coarse manner, and subsequently refined as more data becomes available. In contrast, COMNET models while easier to lay out initially, do not support this type of gradual, modular refinement. There are also standard features of OPNET for which there is little or no support for in COMNET. We will mention only a few of them: mechanisms to support inter-process communications, protocol specification using an FSM and a library of built-in procedures that provides programmable access to interrupts, and the ability to extract information from named fields in data packets.

5.4 The OPNET Simulation Models

5.4.1 Developing the simulation model

The process of building the OPNET simulation of ALAX involved initially breaking down the conceptualized operations and functionalities of the proposed architecture into sample queuing constructs. This methodology was applied for each interface module and further, for each device within the ALAX switch as designed in Figs. 4.16, 4.17 and 4.18. Several iterative analyses and exercises followed then, before settling for the most adequate queuing abstracts leading to the detailed characteristics simulated within the OPNET environment. Subsequently, a great deal of simulation implementations were produced in OPNET, prior to generating the final model that contained the most significant information for pursuing our study.

In the next discussions, we engage in summarizing these findings by outlining the ALAX model properly. We define the extent of the embedded assumptions and related specifications. We present our basic representative queuing system and its respective operating properties. Finally, we establish and describe the OPNET models derived from the preceding analytical studies.

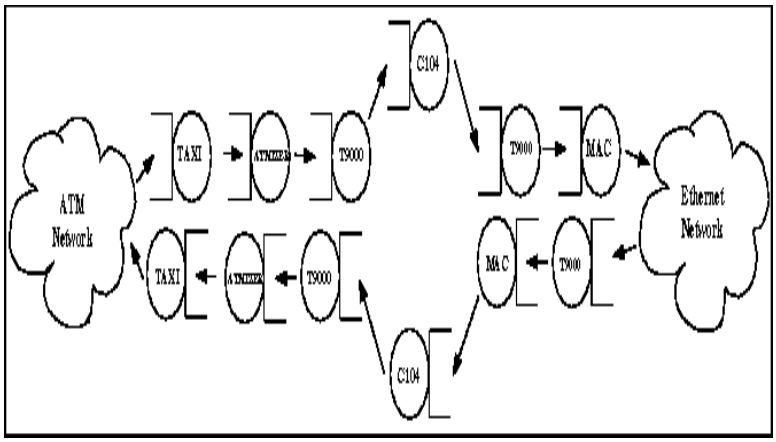

Figure 5.1: Simplified representation of ALAX as a queuing system

Assumptions and Specifications

The first version of the OPNET simulation of ALAX incorporates precisely four network ports. This constitutes the 4-by-4 model, with one ATM-IEEE 1355 and three LAN-IEEE 1355 interface modules. Fig. 5.1 shows a system-level sketch of this basic ALAX model with the components deemed necessary for our investigation. Several considerations and ground rules are made in building this model:

1. The ATM-IEEE 1355 module is abbreviated to account only for three submodules or devices, contrary to the original design schematic of Fig. 4.16. The three submodules are TAXI, ATMizer and T9000 chipsets. In this context, the ODL interface module is omitted due to the anticipated miniscule delays that it would impart overall compared to the other modules of the ATM-IEEE 1355 network adapter.

2. The LAN-IEEE 1355 interface module for Ethernet network is sufficiently represented with just two submodules: T9000 transputer and the MAC chipset.

3. Since the C104 is a key component of this switching system, it is modeled independently as a queue with further details internally. This is also true for the T9000. Meanwhile, the other modules are simply single queues with some delays built-in, and performing packet stripping and collection.

4. The shared memory buffer across the T9000 transputer is divided into subqueues within the T9000 model itself.

5. The T9000 message or packet with its maximum size of 32 bytes is referred here and after, as the IEEE 1355 packet. Noteworthy, this is actually a misnomer, since the IEEE 1355 standard does not directly imply a threshold for the packet size. However, as acknowledged in Sec. 4.4.6, the architecture and system design of ALAX, and by the same token the simulation model, are constrained to this size. And this is due to the fact that T9000's are utilized along the data and control paths.

5.4.2 Queuing System for Derivation of OPNET Model

Generality

The diagrams in Figs. 5.1 and 5.2 reflect the essence of the OPNET simulation model for ALAX. They represent the underlying queuing systems accounting for the assumptions stated above. In contrast to the first one, the second diagram delves into the specific characteristics of both the T9000 and the C104. And, these are merely seen as a network of queues and servers. In order to bring them to the context of simulation, we explore in an in-depth manner the significance of these queuing models.

The Queue and Server Entities

The C104 queuing representation directly follows from the description in the data sheets provided by Inmos SGS-Thomson [28]. It includes primarily queues and servers to replicate the DS link transmission and reception mechanisms (GQ1355, HQ1355, SGQ1355 and SHQ1355). The main server, SC104 along with the input and output queues, RQ_i and XQ_i, participate in accomplishing the switching and forwarding of IEEE 1355 packets as defined in [28] also.

The T9000's respectively of the ATM-IEEE 1355 and the LAN-IEEE 1355 interface modules are distinguishable, for they differ in their functions and processing load. At the ATM-IEEE 1355, the T9000 separates the ATM cloud from the C104 switch. This T9000 includes the following queues:

1. On the top half, AQ0, AQ1 and AQ1355.

2. On the bottom half, BQ1355, BQMAC and CQMAC.

In AQ0, where incoming MAC packets, are held, the HOL packet is served by the server SAQ0 first, and then by SAQ1. AQ1 receives the MAC frames for segmentation and mapping into IEEE 1355 packets. The activities of these first two servers account for the T9000 processing the packets (i.e., LANE, packet segmentation and local addressing). At the end of the processing stage, the packets are stacked up in AQ1355 where they remain until the C104 is free to accept them. When this occurs, the IEEE 1355 packets are transmitted over the D-S links. This transmission over the DS serial links is emulated by the sever SAQ1355. This represents a very small delay, which can literally be ignored, because the DS link bears a high-speed transmission property.

BQ1355 and SBQ1355 play the role of DS link receivers for IEEE 1355 packets. There are several BQ1355 queues for the incoming IEEE 1355 packets from the C104 switch. Each of these queues corresponds to a different input port of the C104. Hence with a 16-port C104 switch, the number of BQ1355 queues can potentially be 15.

For each BQ1355 queue there is also a BQMAC queue that collects the reconstructed MAC packets for the LAN Emulation process to take place. BQMAC and SBQMAC together attend to the HOL MAC packet. As soon as the first N IEEE 1355 packets corresponding to the MAC packet arrive the server SBQMAC can start the LANE tasks. Once the LANE is over, the frames can be accepted by CQMAC, where they can be dispatched by SCQMAC to the ATMizer.

The processing capacity of this T9000 is to be divided primarily between two main tasks: LANE service and local addressing. These tasks are performed on both incoming and outgoing traffic streams.

At the LAN-IEEE 1355, the T9000 links the Ethernet cloud to the C104 switch. In this case, the hardship on the T9000 is lesser compared to the previous one, since the packet translation is somewhat different. The queues inside the T9000 are:

1. On the top part, DQ0, DQ1 and DQ1355.

2. On the bottom part, EQ1355, EQMAC and FQMAC.

DQ0 receives MAC packets from the Ethernet world. DQ1 holds the MAC packets for local addressing and segmentation (MML) into IEEE 1355 packets. DQ1355 receives the IEEE 1355 packets and makes them available for transmission over the IEEE 1355 serial links. In this context, we can easily get away with not having DQ1, however, we keep it here for the sake of symmetry with the other T9000 at the ATM-IEEE 1355 interface. This will lead to an easier generic implementation of the T9000 in OPNET. The queues EQ1355 are similar to the BQ1355 ones above. The same is true for the EQMAC and BQMAC queues. But, at the FQMAC queue no LANE service is done, the MAC packets are simply transmitted over to the MAC chipset. Noticeably, for the LAN-IEEE 1355 interface the T9000 processing capacity is once again split among the top and bottom part, i.e. outgoing and incoming traffic. This is done in a similar fashion as indicated for the ATM-IEEE 1355 interface, however this is likely to be less

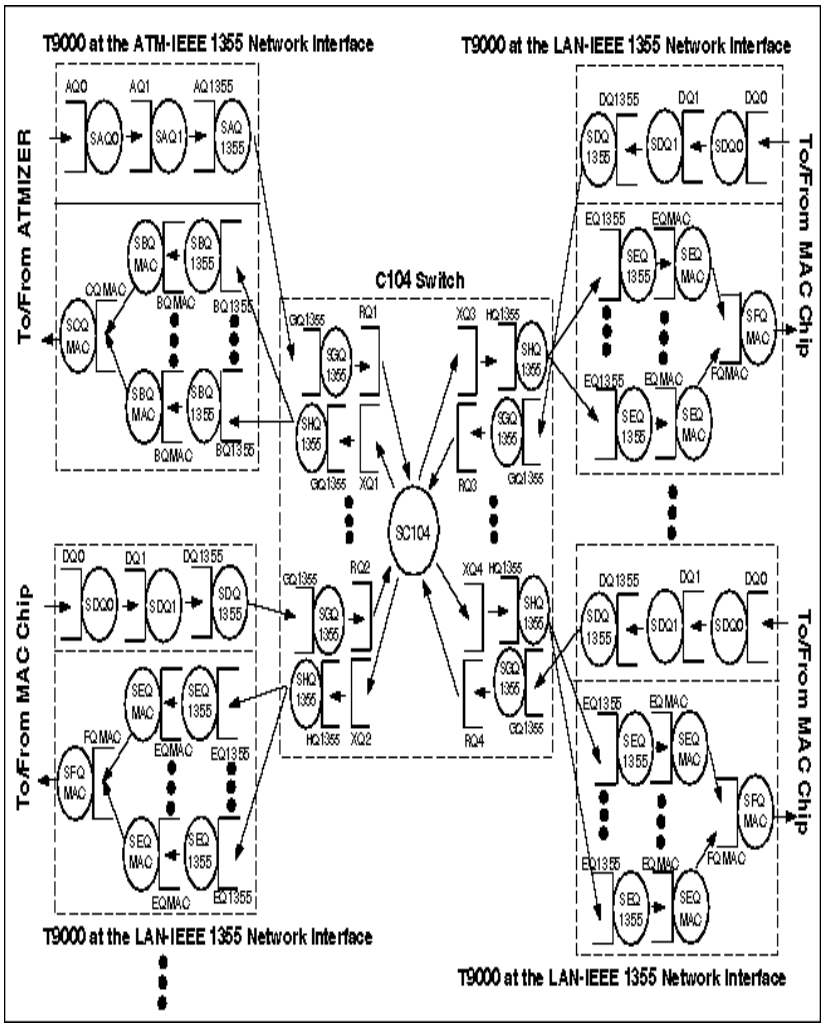

Figure 5.2: Detailed Queuing Model of the T9000 and C104 in ALAX

extensive since no LAN Emulation occurs at the LAN side.

While creating these queuing constructs, several considerations emerged for scheduling the sequence of tasks at the T9000. They were First Come First Served (FCFS), Longest Packet First Served (LPFS) and Round-robin (RR). These scheduling algorithms referred to the scheduling between HOL packets of the various queues attended by the servers inside the T9000, both within the top and bottom half. The Round-robin approach was chosen for its fairness and optimal benefits.

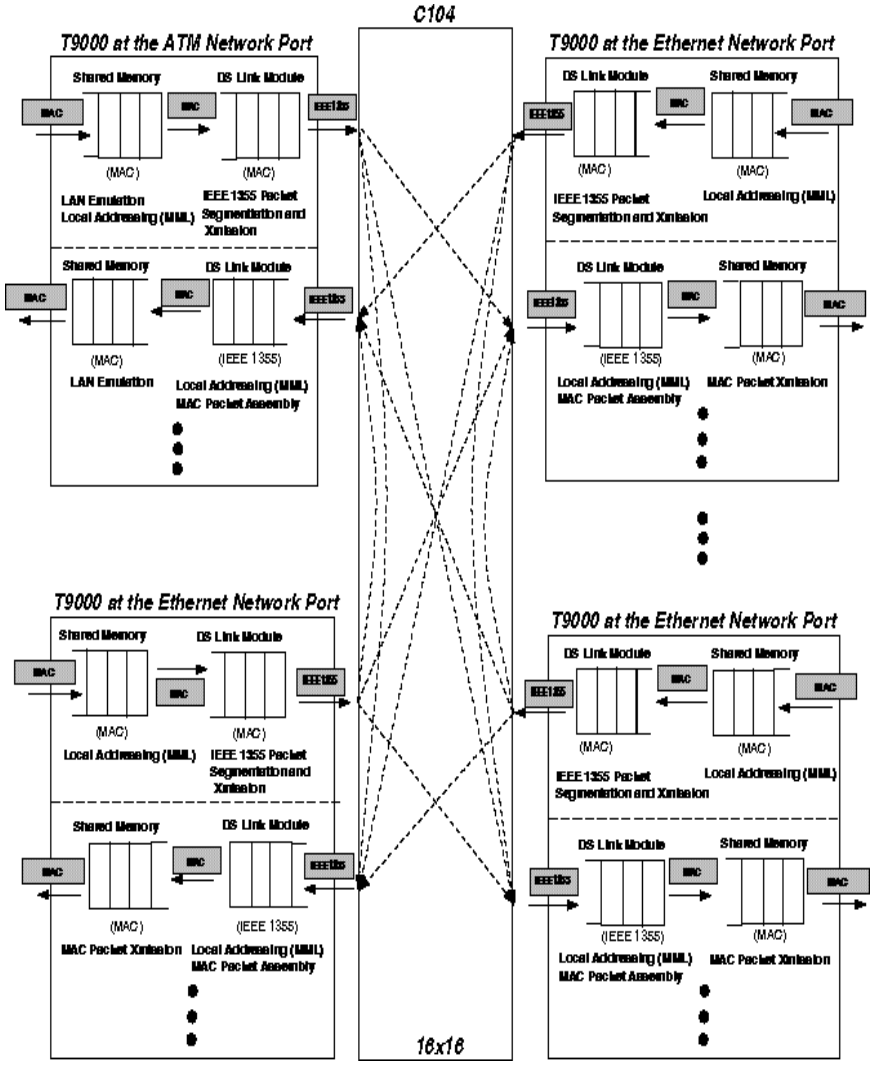

Figure 5.3: Operation of the ALAX switch within OPNET

5.4.3 Description of the T9000 Model in OPNET

In OPNET, the T9000 is first represented as a queue at the *Node* level. Then within the *Process* level, the T900 FSM further divides the queue into many subqueues, as established above, with Proto-C programming code to perform the tasks (see appendix)[23]. At the T9000 of the ATM-IEEE interface module, these subqueues include ATM output buffer and IEEE 1355 input buffers. At the T9000 of the LAN-IEEE 1355 a similar configuration exists with Ethernet output buffer and IEEE 1355 input buffers. The output buffers here refer to the respective port where the T9000 is located as being the sender of packets, while the input buffers mean that the packets residing at this location come from other sources across the C104. The idea behind these buffers is also to detain the packets when the backpressure mechanism is applied by the C104 to refrain the flow of packets from flooding the switch.

Given that the T9000, specifically at the ATM port, could end up being the bottleneck in this interconnection scenario, two approaches were determined to alleviate the resulting side effects.

One requires providing a large memory buffer in anticipation of packet loss. The other consists of having an adequate speed for the T9000 clock in order to prevent slowing down the C104 while performing LAN Emulation for instance. The T9000 transputer processing speed/time at the ATM port is determined in function of:

$$ProcessingTime_{IEEE1355-Pkt} = \frac{Mult}{Rate_{Average}} \tag{5.1}$$

In this formula[5], the *Mult* parameter characterizes a multiplier factor for the processing speed/time of the T9000. The average traffic rate, in bps, amounts to

[5]Implemented only in the T9000 at the ATM port, because we assumed the bottleneck to occur primarily at the ATM port instead of the Ethernet.

the aggregate Ethernet traffic rate injected into the ATM port, while the IEEE 1355 packet processing time represents the time, in sec, to process an IEEE 1355 packet at the ATM port. Hence, with a Mult level of 0.4, the T9000 is expected to process packet two times faster than with a Mult level of 0.8. Further, it is established that $Mult > 1$ implies that packets are being processed at a slower rate than their arrival rate.

These two characteristics, above, involve some kind of tradeoff to derive the optimal point of performance of the switch. One effective way that has been devised to model the T9000 requires using preemptive queuing whereby IEEE 1355 packets which have been completely translated into the IEEE 802.3 format are given highest priority. What this means in reality, is that unless a HOQ packet has been completely serviced it will not be extracted by the transmission server of the T9000 [23].

5.4.4 Description of the C104 Model in OPNET

The picture in Fig. 5.6 illustrates the OPNET model of the C104 in the original 4-by-4 configuration of ALAX. There are 4 input queues and 4 output queues for each network port. However, in OPNET, this is implemented in the *Node* level as a queue, and in the same manner as in the T9000 case, there is an FSM and the operations are executed as Proto-C code. This can be summarized as follows[6], when a packet arrives at the input port of the C104, the header of the packet is read to decide the destination output subqueue. Then, the destination output subqueue is checked, if it is empty the packet is put into the subqueue. Otherwise, the packet is temporarily stored in the input subqueue of the corresponding input stream port until the output port is available [23]. This is done to replicate the combined input and output buffering implementation of the C104 as indicated in Chap. 4. A delay of 1 μsecond is inherent in the model to account for the minimum delay that the C104 bears [16].

[6] A pseudo-code of the C104 fabric as implemented in OPNET is available in [23].

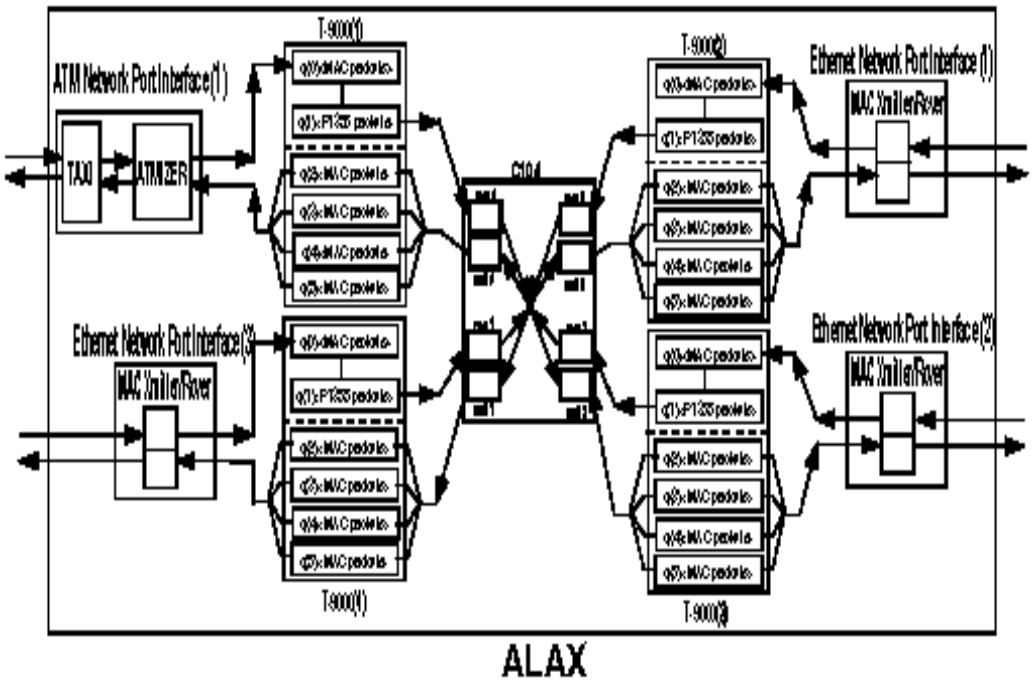

Figure 5.4: Diagram of the 4x4 ALAX switch with designated buffers for statistics monitoring

5.4.5 Traffic Characterization Models

Overview

Traffic modeling[7] is an important component of the simulation of any communication system. This is even more crucial for emerging high speed communication networks. As the design of a network depends to a great extent on the type of traffic it is expected to transport, it becomes necessary to appropriately characterize the traffic that the particular network is expected to carry. In the absence of these traffic models, the only way to validate and refine the network design would be to inject into

[7]This section of the thesis is largely due to the contributions and efforts of Sandeep Rao in developing the traffic models for ALAX.

the network model real life traffic. Even then, we still could not be ascertain with the results obtained that the network design behaves, routes and coordinates the traffic as it is supposed to.

It is for this reason that traffic models are used throughout network design simulation, for they allow a parameterization of the essential characteristics of expected network loads. Hence by generating the traffic under the rules set up by the traffic models in question, one can clearly establish with great confidence the validity of a network design model.[34]

Scientists/Mathematicians have been doing a lot of research with respect to traffic models. And through the years, several basic models have been developed and compiled, which accurately imitate any type of traffic source (i.e., data, voice, video etc.).

In simulating the operation of ALAX the traffic models used in the traffic/packet generators are based on the Long Range Dependent (LRD), Short Range Dependent (SRD) and VBR models. The SRD traffic profile implemented is based on the Markov Modulated Poisson Process (MMPP), the VBR traffic model used is the *StarWars*© movie, while the LRD traffic operates under some specified parameters for the $M/G/\infty$ queuing model. The main difference between SRD and LRD models, is that unlike SRD models, LRD models display correlation between packets separated very far apart in time [35]. Further details on the traffic models utilized for the ALAX simulation can be found in [23,35]. These traffic generators are represented in OPNET through processors at the *Node* level and FSMs at the *Process* level. With their respective characteristic parameters appropriately set they generate packets to be sent to some destinations across the C104.[23]

Burstiness in SRD and LRD Traffic Profiles

In generating any traffic, especially data, one important characteristic pertains

to the burstiness nature. This can tremendously affect the design requirements of a network system. Simply stated, burstiness reflects the behavior of a traffic to have short periods of intense activity followed by long period of idle time.

The average rate for generating the LRD-type traffic source based on the $M/G/\infty$ queuing process is given by[8]:

$$Rate_{Average} = \lambda \times \delta T \times E[\sigma] \tag{5.2}$$

The average rate here refers to the traffic rate in question. λ is the arrival rate into the $M/G/\infty$ queuing process. $E[\sigma]$ is the expected mean of the pareto distributed activity periods in the $M/G/\infty$. δT represents the time unit at which the $M/G/\infty$ is sampled (i.e., frequency).

The SRD traffic profile is implemented with two FSMs (Fig. 5.7) with equal time spent at both states. Each state generates packets at a certain rate so that the average of the two rates is a ratio parameter λ that bears the burstiness characteristic of the traffic. So, burstiness is embedded into the traffic rates as: $\lambda_1 : \lambda_2$ or simply, $xk : x$. Now, the average rate for generating the SRD-type traffic based on the MMPP process is given by[9]:

$$\frac{Rate_{Average}}{PktSize} = \frac{x + kx}{2} \tag{5.3}$$

In this formula, the average rate represents the overall rate of the traffic source in question. Pkt-Size is the size of each packet to be produced. x is the rate that packets are generated at one state, while k is the multiplier factor for the rate that packets are

[8]This formula was provided by Sandeep Rao. And the steps to derive it can be found in [35]

[9]This formula was provided by Sandeep Rao. And the steps to derive it can be found in [35].

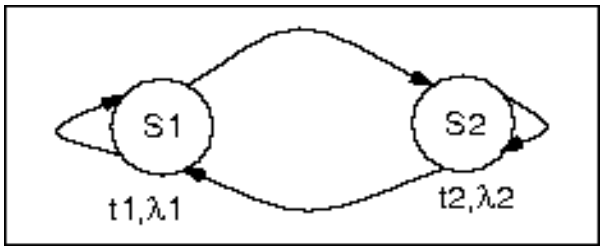

Figure 5.5: Representation of the MMPP Traffic Generator

generated at the other state of the MMPP. Hence in this case, the burstiness property of the traffic model is essentially varied through a ratio level, as opposed to a frequency level in the LRD case.

5.5 Summary Note

We have gone through the appropriateness of simulating the ALAX switch for collecting data to guide in the correct implementation of this system. Then, we have highlighted the design process involved in modeling this switch with particular attention given to key elements of the simulation model of ALAX: the T9000 and C104.

The next chapters outline the OPNET simulation runs and other related activities that shaped the experimentation. We will report on the data collected for various network scenarios.

Chapter 6

The ALAX Simulation Experiments- Part I

Using the ALAX Simulation model, we have endeavored to build and execute an extensive series of experiments and benchmarks that would provide our collaborators, ETRI and Modacom, with critical queuing sizes, optimal T9000 processing speed and end-to-end packet delay measurements. These numerical indicators would give guidance in selecting the most appropriate memory sizes and speeds for the T9000, while suggesting the expected levels of performance that could be achieved with the ALAX switch. The process of conducting these tests has opened up areas for further testing and researching within the life-cycle of the ALAX switch. One can expect then the ALAX simulation model to be refined and used concurrently during the prototype building phase to investigate and verify design options.

In this part of the thesis, we deal with data presentation and interpretation for the simulations involving the original 4-by-4 ALAX model developed during the 2 years of the ALAX project. The testing or experimental framework is outlined first. Then the results collected for each type of traffic profile fed into the 4-by-4 ALAX are analyzed from different angles. These results include average end-to-end packet delays and queuing sizes[1]. Overall, observations and conclusions about the architecture and

[1] Variances/standard deviations for the queuing sizes were also collected, however they are not part of the analysis. These variance/standard deviation measurements can be made available upon request.

design of this 4-port ALAX switch are derived.

6.1 Experimental/Testing Framework

6.1.1 Strategy/Methodology

Our study targeted a 100 Mbps-TAXI physical interface connector. This interface is integral to the ATM port as designed (see Chap. 4). Testing covered two general areas: end-to-end packet delay across the switch or latency, and queue sizes for both ATM and Ethernet ports. The tests caused the ALAX switch to service 4 network physical ports under three types of network packet generators, which are based on the LRD, SRD and Star Wars traffic models defined earlier (Chap. 5). For our case studies, there are one ATM port and three Ethernet ports. Traffic flow occurred bi-directionally between the ATM and all the Ethernet ports. Each physical switch port hosted a number of logical or virtual circuit connections to other ports in a mesh configuration (see Fig. 6.3). Each test was designed such that any physical port could send packets to any other ports, except itself. Ethernet frames with a maximum size of 1526 bytes (12208 bits) and ATM cells of 53 bytes (424 bits) [10,19,39] were the baselines utilized for the traffic generators at each network port across the switch.

The Ethernet port rate was fixed for all the simulation runs at 4 Mbps. This number was chosen because, even though the nominal speed of shared-Ethernet is theoretically 10 Mbps, on average a station can only expect to use successfully and effectively between 4 to 6 Mbps, owing to collisions and retransmissions.

The ATM port rate, even though designed for a 100 Mbps transmission with the TAXI interface, was simulated for the aggregate rate of all the corresponding Ethernet sending ports. Thus, its rate was set for 12 Mbps. The motivation for this decision resides in the fact that a worst-case condition was sought for the switch's operation. This would optimally give the queue sizes and latencies for design considerations. Hence

provisioning for the worst-case situations that may prevail within ALAX.

Finally, in order to monitor the behavior of the ALAX switching device, several burstiness levels were selected for each type of traffic generator used while four levels of T9000 processing speed were adopted. Consequently, in each simulation run these associated levels that affect both the processing speed and the burstiness of the traffic type at hand were varied accordingly.

6.1.2 Simulation Setup

The OPNET environment provides the capability to execute the programs within the GUI environment or in batch mode at the command prompt. We selected the batch mode for execution was much faster than in the GUI environment. Running a simulation in the batch mode, required several files. First, the simulation file itself, i.e. the file with the model description, as produced in OPNET with the extension *".sim"*, second an environment file, where the simulation parameters or variables are defined and can be modified, with the extension *".ef"* and finally the file(s) for defining or modifying the burstiness properties or characteristics of the traffic generator with the extension *".gdf"*. These files were integrated into a UNIX pipeline command as:

Alax.sim -duration "in sec" -ef AlaxEnv > datafile

The *datafile* refers to the ASCII file where the results are stored. The other files with the *".gdf"* extension were called internally by the traffic models as the simulation ran. These files are provided in the appendix for references.

One interesting parameter in the command statement is the duration of the simulation, defined in units of seconds. This simulation time as it is commonly called, must be specified since we can not run the simulation indefinitely due to logistic constraints (e.g., limited resources available). Moreover, it must be identified appropriately in order to remove any transient results. After several long run trials, this simulation time

was determined to be 10 seconds. Hence all the simulations were run for a duration of 10 seconds with the view that 10 seconds reflect the steady-state operation of the simulation model for ALAX.

6.1.3 Notation and Parameters Used in the Study

In the previous sections of this thesis, we have discussed several design characteristics that are inherent to the ALAX switch. Here, we review and define some terminologies pertaining to these properties as they relate to the presentation of the results.

First of all, as indicated earlier in Chap. 5, Sec. 5.4.2, the unit of measurement for the ALAX packets is referred to as *IEEE 1355 packet* (32 bytes). The queue or memory size estimates reported below are also given in terms of this unit. Secondly, the *Mult* factor which establishes the processing capability of the T9000 transputer, is implied under the denomination of *T9000 Processing Speed Level*. Thirdly, the parameter measure that affects the burstiness characteristic applicable to each traffic model is labeled accordingly as *burst frequency* or *burst ratio* levels for the LRD-type and SRD-type traffic profiles.

Finally, since each T9000 at a corresponding port, contains a queue for each sending port irrespective of the model being used, the numbering of each queue properly reflects and identifies the sending port. Hence, for the ATM port, *Q[0]* and *Q[1]* apply to the traffic stream going towards the C104; they are the locations where LANE is performed, local addressing for the MML occurs and also buffer space until the C104 is ready to accept packets within its input queues. However, *Q[2]*, *Q[3]*, *Q[4]* up to *Q[17]* indicate the receiver queues for traffic coming from the ATM port itself, Ethernet port 1, Ethernet port 2 and up to Ethernet port 15 (accounting for the maximum data port that can be implemented on ALAX). On the LAN sides, however, for each Ethernet port, *Q[0]* and *Q[1]* account only for MML and the buffer space until the

C104 is ready to accept packets. And, for the remaining queues from *Q[2]* and on, the same principle evoked above is applicable. The implications of queue numbering are exemplified in Figs. 6.1 and 6.2 .

6.1.4 Preliminary Findings, Observations and Related Modifications

During the initial simulation tests, the early results obtained using the first model of ALAX (i.e., 4-by-4) allowed the determination of essential parameter values or benchmarks to pursue further simulation studies, and pointed out many bottlenecks in the ALAX system design.

The T9000 processing speed level should range from 0.0 to 1.0, to be acceptable for a stable operation of the ALAX switch. This was seen in the previous chapter. Given that we could not possibly run simulations for the entire range of possible values, four points were selected to conduct the complete simulation sets: 0.1, 0.5, 0.8 and 1.0. Within these points, 0.8 satisfied the requirements for the worst-case scenario where the T9000 would still maintain a stable condition for the overall operation of the ALAX switch. Along with these speed levels, the burst frequency levels chosen for testing purposes in the case of the LRD-type traffic were: 7.5 secs, 15 secs, 30 secs, 60 secs and 120 secs. Additionally, the burst ratio levels associated with the SRD-type traffic were: 1:1, 10:1, 50:1 and 500:1.

Further, it was determined that having one T9000 processor at the ATM port led to heavy end-to-end packet delays and large queue build-up as indicated through the first runs. Hence the original design presented in Chap. 4 - Sec. 4.4, Figs. 4.11 and 4.16 - was revised and modified to include two T9000's instead of one. These changes did not affect the overall ALAX architecture. What this amounted to, was that the load at the ATM port would be divided between the two T9000's. Thus, one T9000 would carry unidirectional traffic stream from the ATM network port to the C104, using *Q[0]* and *Q[1]*, while the other would cater to the other unidirectional

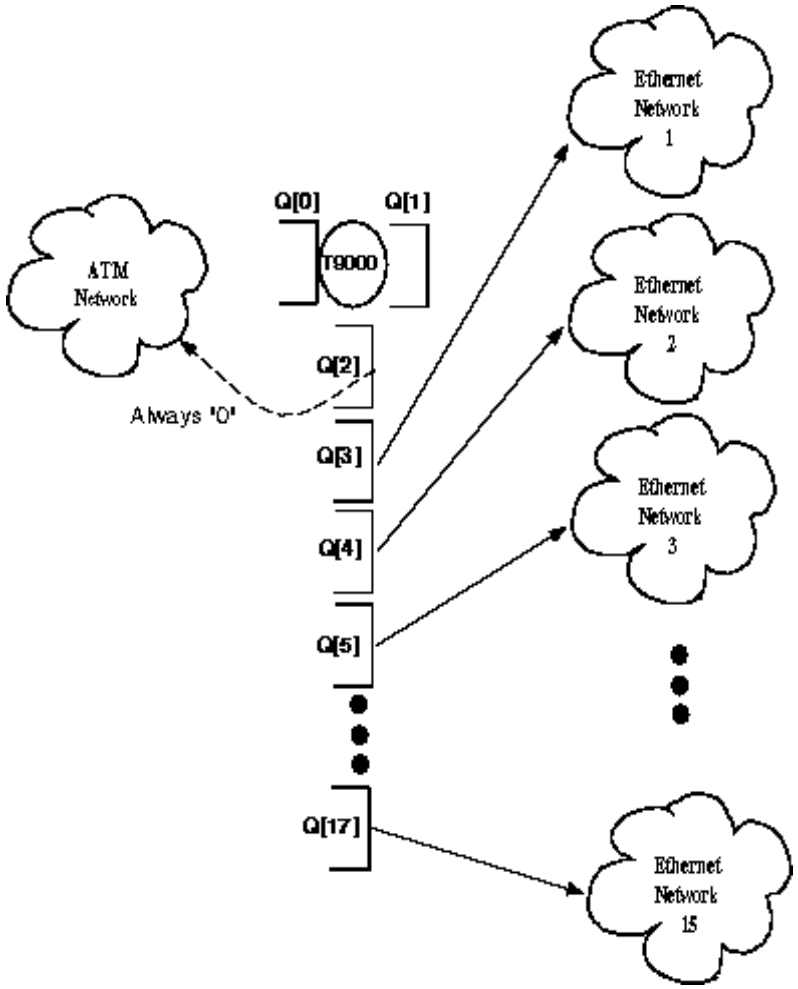

Figure 6.1: Implications of queue numbering at the ATM port

traffic stream from the C104 to the ATM network port, using *Q[2]*, *Q[3]* and so on depending on the model used. These changes where implemented into the original OPNET model of ALAX without too much effort. These T9000's are labeled T9000 # 1 and # 2 respectively.

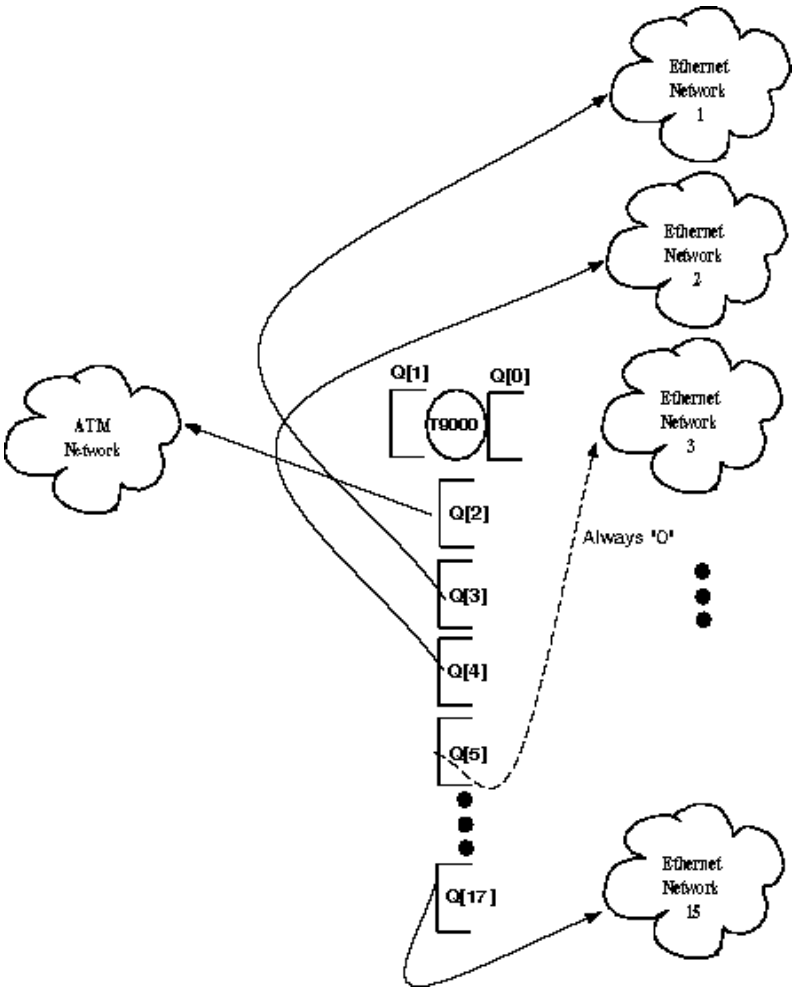

Figure 6.2: Implications of queue numbering at Ethernet port 3

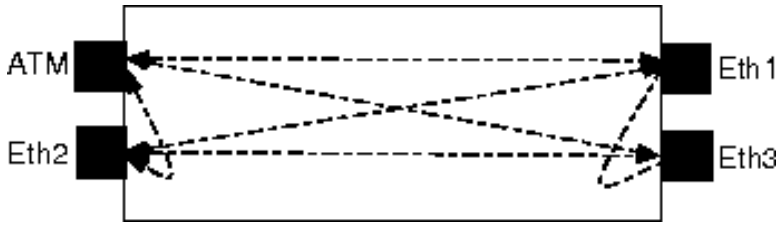

Figure 6.3: Mesh Connectivity inside the 4-by-4 ALAX model in OPNET

6.2 Outcomes under the LRD Traffic ($M/G/\infty$)

The experiments conducted with the LRD-type traffic generators based on the $M/G/\infty$ queuing process, involved two contexts. In the first context, the packets produced at any port of the switch were sent to every other port in a uniformly distributed manner (i.e., address destinations created upon the assumption of a uniform distribution among the number of ports). In the second context, however, packets were generated with their destinations addresses randomly distributed among the switch ports. The LRD-type traffic profile was used only in the case of the 4-by-4 ALAX switch model.

6.2.1 Testing with Destination Address Uniformly Distributed

For each of the four T9000 processing speed levels, the 4-by-4 model is monitored under five burst frequency levels of the LRD-type traffic: 7.5, 15, 30, 60 and 120 seconds. Hence the most frequent bursts would occur for 7.5 seconds, while 120 seconds would feature the least frequent bursts. The total number of simulation runs is 20 for this scenario. We analyze the results for the queuing and delay evaluation.

Queuing Analysis

At the ATM port, when the T9000 processing speed level is 0.1 (i.e., fastest), the observations are the following. In the T9000 handling traffic stream towards the

C104 switch (i.e., T9000 #1), the average and maximum occupancy for *Q[0]* remain constant at 0.08 and 1.0 IEEE 1355 packet respectively (see Fig. 6.4), throughout the burstiness variation from 7.5 seconds to 120 seconds. However, for *Q[1]*, the values vary depending on the burstiness intensity. From very bursty at 7.5 to least bursty at 120 (see Fig. 6.5 and 6.7), the average size decreases to 40%, while the maximum to about 20%. This is due to the backpressure from the C104 preventing T9000 #1 from sending too many packets than it can serve. Meanwhile, in T9000 #2, for each queue receiving packets from a sending port, the sizes are constant throughout the burstiness variation both for the average and maximum (see Table A.2 in the appendix). Slowing down the T9000 processing speed level from 0.1 to 0.5, results in a quick build up at *Q[0]* of T9000#1 as noted by the queue occupancies in Table A.3 and Fig. 6.8. The same phenomenon occurs at *Q[1]* (Fig. 6.9). This net increase is valid for both the average and maximum sizes. Now, instead of having a constant size at the average and maximum size of *Q[0]*, there is a variation dependent upon the burstiness level. Overall though, the pattern is such that as the burstiness level decreases, so does the average and maximum sizes at both *Q[0]* and *Q[1]*. In T9000 #2, one interesting revelation is that we have more or less the same occupancy levels at all three queues for the average (Figs. 6.10 and 6.11). And further, in the case of burstiness level being 7.5, the maximum size at all queues is the same as when the T9000 processing level was 0.1. But, the maximum sizes vary for all the other burstiness cases. Further, in the case where the T9000 processing speed level is 0.8, a net increase in the queue sizes for both the average and the maximum is observed in T9000 # 1 and #2. This increase is about the same order of magnitude as in the case where the T9000 processing speed is 0.5. The same trend of increasing queue sizes as burstiness level is high, and decreasing as burstiness level is low is noted. And again, as for when the T9000 processing speed level was 0.5, the average and maximum queue sizes at T9000 # 2 for all three queues at a specific burstiness level is the same. This should be expected since all the Ethernet

ports are sending packets at the same rate. Finally, at the T9000 processing speed level of 1.0, the same dependency of queue sizes on burstiness levels is observed. In T9000 #1, *Q[0]* maintains the same order of magnitude as when the T9000 processing speed level is 0.5. But, in *Q[1]* the sizes more than double sometimes for each burstiness case. In T9000 #2, the average and maximum occupancies more or less double in size in all the queues, *Q[3]*, *Q[4]* and *Q[5]*.

In each of the four cases of T9000 processing speed levels, at the four Ethernet ports the queue sizes at the respective T9000 of each Ethernet port stayed within the same range. Hence, we only report the results at Ethernet port 3. The results for the other Ethernet ports are available upon request. Throughout the process of changing the T9000 processing speed level from 0.1, 0.5, 0.8 and 1.0 at the ATM port, no significant effects were noted. As exemplified in tables A.9, A.10, A.11 and A.12 for Ethernet port 3, the values are more or less the same for the corresponding burst frequency levels. The average of *Q[0]* wanders around 0.002 to 0.004 (i.e., almost zero), while the maximum hovers at 1.0 . For *Q[1]*, the average sits at 0.3 and the maximum at 48.0 . In *Q[2]* though, there is a drastic increase in both the average and maximum sizes, because these packets come from the ATM port, which has a far greater rate of transmission than all the Ethernet ports. In *Q[3]* and *Q[4]*, the same patterns as noted earlier in *Q[1]* are seen. Overall, we can affirm that the T9000 processing speed level does not affect the queues inside the T9000 of an Ethernet port, except the queued receiving packets coming from the ATM port *Q[2]*. This is expected, since the processing speed level only applies to the ATM port and not the other switch ports.

Delay Analysis

The delays collected under the scenario of the LRD traffic with address destinations uniformly distributed among the 4-by-4 ALAX switch ports are shown in Tables

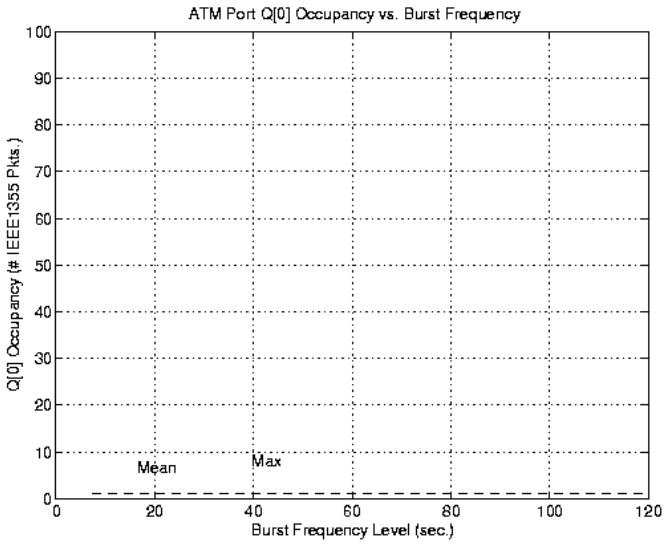

Figure 6.4: ATM port Q[0] Occupancy vs. Burst Frequency Level- T9000 Proc. Speed = 0.1

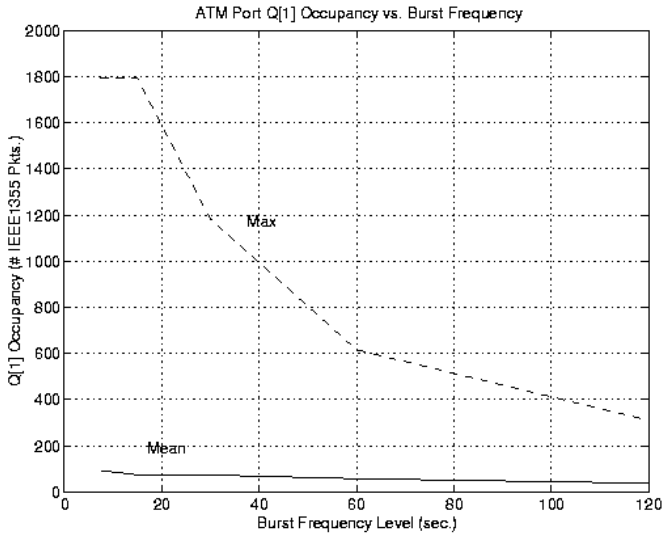

Figure 6.5: ATM port Q[1] Occupancy vs. Burst Frequency Level- T9000 Proc. Speed = 0.1

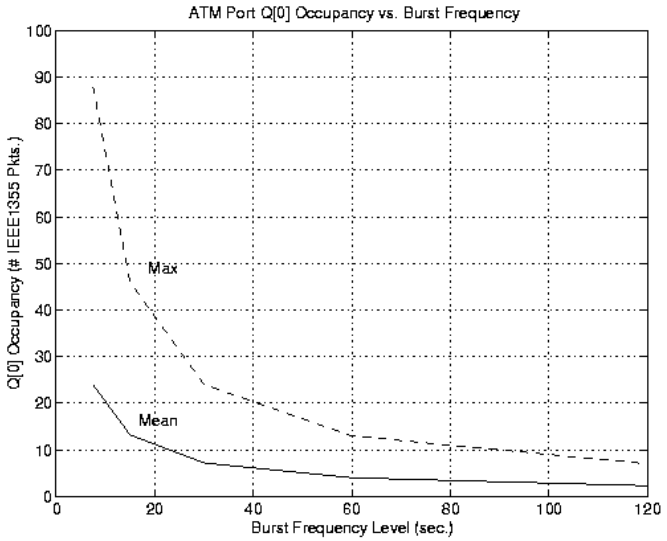

Figure 6.6: ATM port Q[0] Occupancy vs. Burst Frequency Level- T9000 Proc. Speed = 0.8

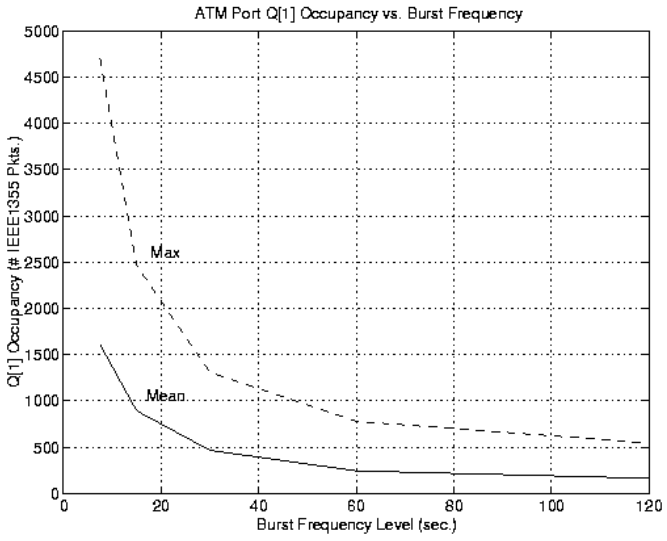

Figure 6.7: ATM port Q[1] Occupancy vs. Burst Frequency Level- T9000 Proc. Speed = 0.8

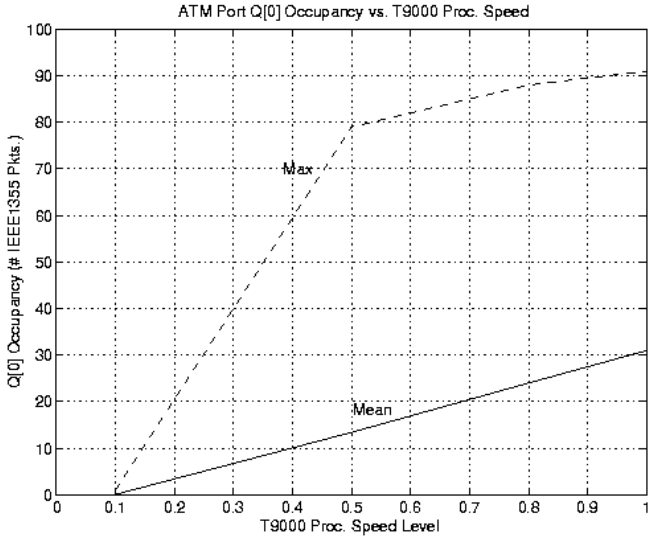

Figure 6.8: ATM port Q[0] Occupancy vs. T9000 Speed Level- Burst Frequency = 7.5 secs.

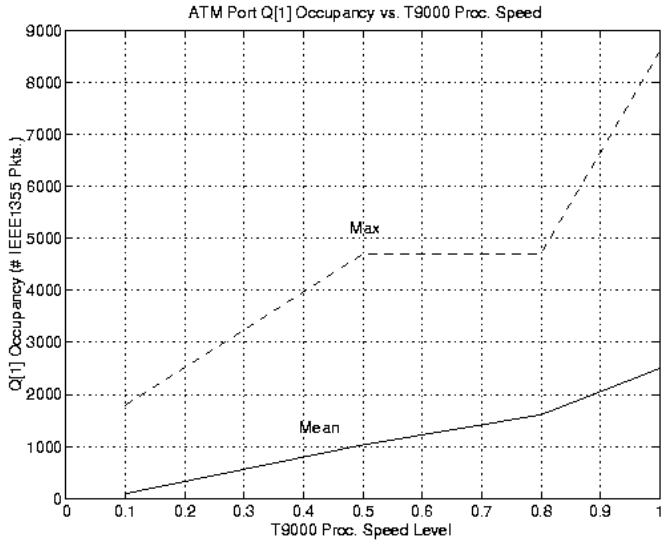

Figure 6.9: ATM port Q[1] Occupancy vs. T9000 Speed Level- Burst Frequency = 7.5 secs.

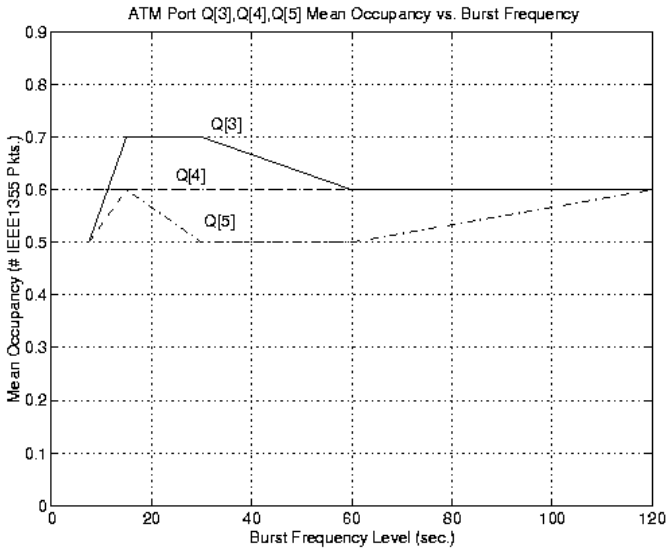

Figure 6.10: ATM port Q[3],Q[4],Q[5] Mean Occupancy vs. Burst Frequency- T9000 Proc. Speed = 0.1

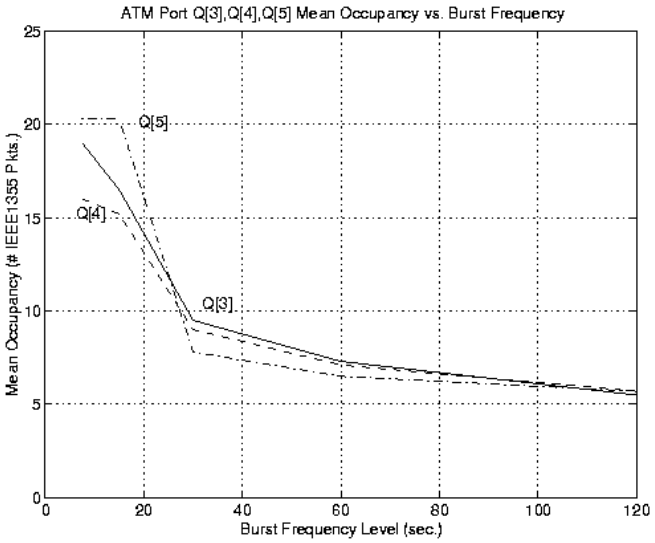

Figure 6.11: ATM port Q[3],Q[4],Q[5] Mean Occupancy vs. Burst Frequency- T9000 Proc. Speed = 0.8

A.13 and A.14 . The first table offers the average delays for packets leaving the ATM port towards any of the Ethernet ports. The second table indicates the average delays for packets leaving any of the Ethernet ports towards any other ports, meaning ATM or the other Ethernet ports.

For each of the five levels of burstiness intensity, as the speed of the T9000 at the ATM port is increased, the delay proportionally decreases. However, this pattern does not adapt well with regard to changing the burstiness intensity while keeping T9000 processing speed constant. For instance, in both cases where the T9000 processing speed level is higher, the delay stays more or less uniform across the burst frequency levels, respectively at 0.3 msec and 1 msec. This robustness in delay performance is quickly lost when the T9000 processing speed levels are 0.8 and 1.0 (Fig. 6.12). Indeed, the packet delay fluctuates with respect to the burstiness: the lower the burstiness level the lower the end-to-end packet delay. And, these delays have more than doubled compared to the previous ones. This would lead us to infer that the acceptable performance region of the T9000 under the worst-case traffic conditions that may arise is situated somewhere in the range of [0.0,0.8] as determined in earlier simulation runs.

At the Ethernet ports, when the T9000 processing speed level is increased, the delay decreases proportionally (Fig. 6.13). And unlike the ATM port, the packet delays vary for T9000 processing speed levels of 0.5, 0.8 and 1.0; while for the T9000 processing speed level of 0.1, the delays remain more or less constant throughout the burstiness variations.

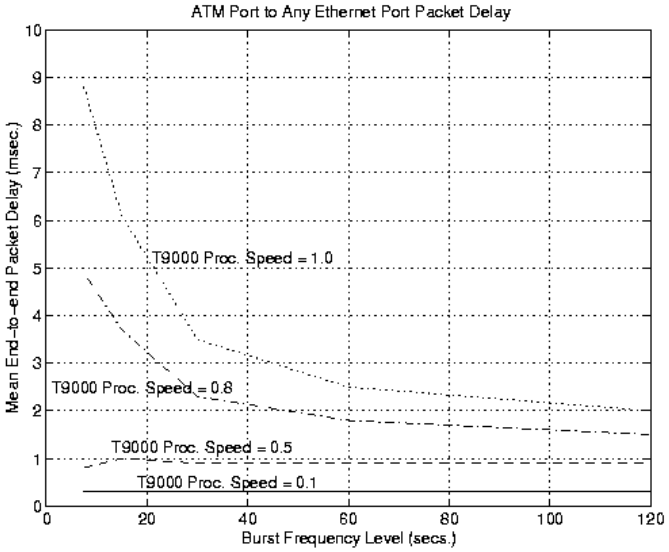

Figure 6.12: Average End-to-end Packet Delays from the ATM Port to Any Other

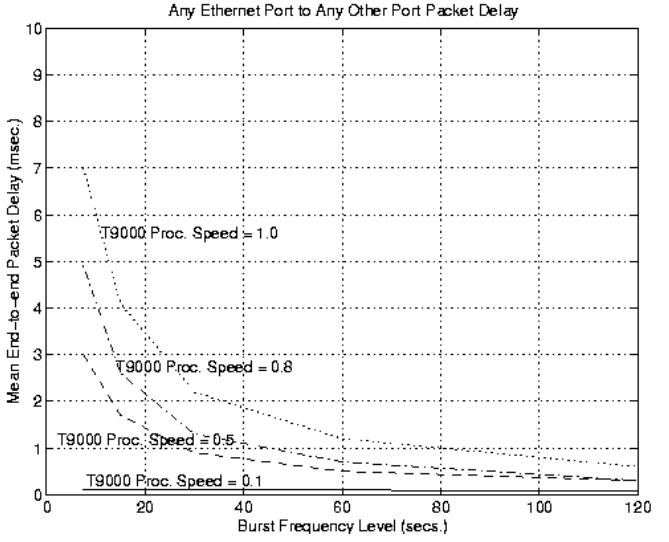

Figure 6.13: Average End-to-end Packet Delays from Any Ethernet Port to Any Other

	ATM Port	*Ethernet Port 1*	*Ethernet Port 2*	*Ethernet Port 3*
ATM Port	0	0.31	0.44	0.25
Ethernet Port 1	0.53	0	0.26	0.21
Ethernet Port 2	0.26	0.53	0	0.21
Ethernet Port 3	0.21	0.53	0.26	0

Table 6.1: Probability Matrix for Random Destination Address Selection

6.2.2 Testing with Destination Address Randomly Distributed

In this case, there is no need for the concept of burst frequency level that we used previously. Because, even though the meshed connectivity shown in Fig. 6.3 is still maintained, the LRD-type traffic generated at any port is now sent randomly to other destinations according to a probability distribution. The probabilities used in these tests appear in Table 6.1 . As usual, a port does not send to itself, i.e., the probability for these entries is zero. Knowing that, a T9000 processing speed level of 1.0 would not perform very well compared to the other speeds, the 4-by-4 ALAX model was evaluated for T9000 processing speed levels of 0.1, 0.5 and 0.8. Hence, the total number of simulation runs is only 3 for this setup. This accounts for all the three T9000 processing speeds simulated.

Queuing Analysis

In T9000 #1 of the ATM port, the queue occupancies are low for both the average and the maximum at the highest T9000 processing speed level (i.e., 0.1). But, they grow if the T9000 processing speed level decreases, as expected (Table A.15, Figs. 6.14 and 6.15). At T9000 #2, The average number of packets sent by Ethernet port 1 are greater than the average sent by the other two ports (Fig. 6.16). This should be so, since the probability of sending packets to the ATM port is higher for Ethernet port 1. Nevertheless, the maximum number of packets sent by all the Ethernet ports

to the ATM, is the same for each speed of the T9000 (Table A.16).

For the T9000 at Ethernet port 1 (Table A.17), the maximum queue occupancies remain the same for each case of T9000 processing speed. Since the probability of sending to Ethernet port 1 is higher for the Ethernet ports 2 and 3, the average queue occupancy should indeed be higher for *Q[4]* and *Q[5]*. However, this can not be because the ATM port is sending data at a much faster rate than the others. Consequently, the average of *Q[2]* is much higher. Noticeably, the averages at *Q[4]* and *Q[5]* are the same, due to the fact that both port have the same probability of sending to the Ethernet port 1. In *Q[0]* and *Q[1]* the values stay intact even as the speed changes for the T900 at the ATM port. There is no surprise there, since the Ethernet port has no direct relationship to the T9000 ATM port.

At Ethernet port 2 (Table A.18), the outcomes are somewhat similar, but applied with respect to the ports with the highest probabilities. In this case, it is the ATM port. The number of IEEE 1355 packets for its corresponding queue are higher than all the other queues compared. Aside from that fact, the same observations made earlier are true. Essentially, the average and maximum number of packets for all the queues, besides the one for the ATM port, remain the same even though the T9000 processing speed at the T9000 was varied.

The T9000 at Ethernet port 3 (Table A.19) behaves more or less in the same fashion as the previous other ones. Once again, the average and maximum number of IEEE 1355 packets are maintained throughout the variation of the T9000 processing speed level at the ATM port. Also, the average number of packets from the T9000 port is much higher than all the others.

Delay Analysis

For the simulations involving the LRD-type traffic with address destinations

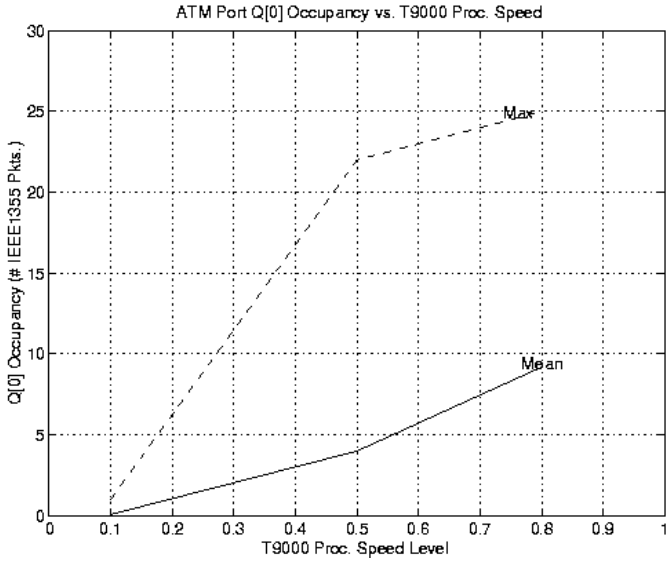

Figure 6.14: ATM Port Q[0] occupancy

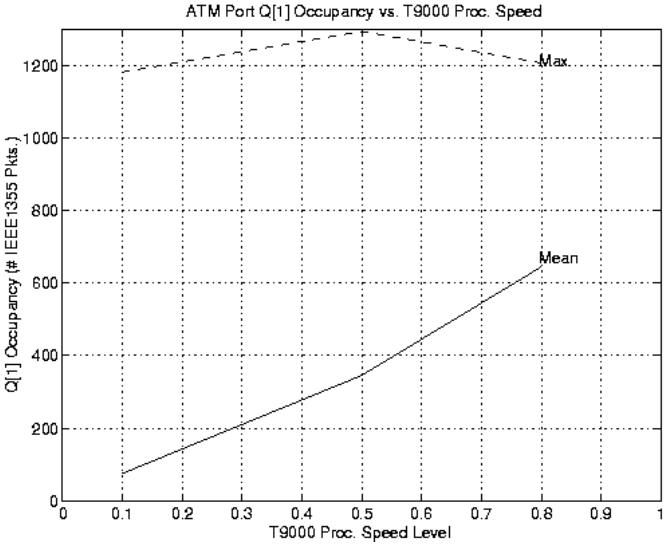

Figure 6.15: ATM Port Q[1] occupancy

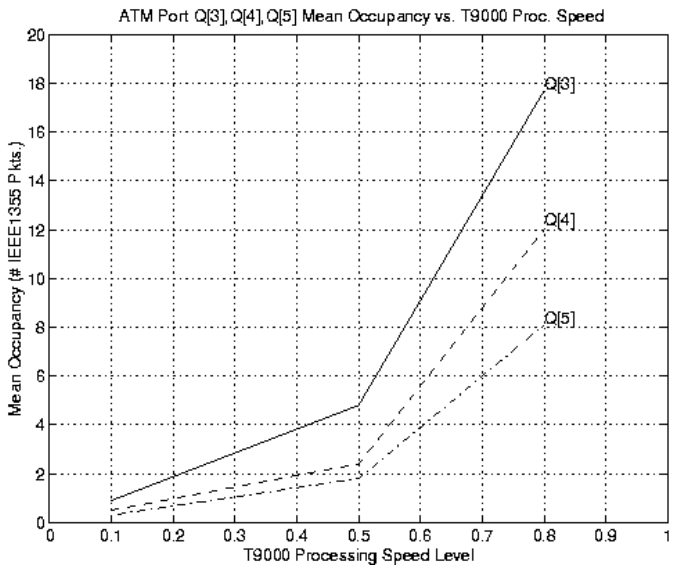

Figure 6.16: ATM Port Q[3], Q[4] and Q[5] occupancy

randomly determined, average packet latencies are low as the T9000 processing speed is high and vice versa. This is true on both the ATM and Ethernet ends (Table A.20). One important outcome worth acknowledging, is that more or less the average delays at the ATM port match the average delays for the Ethernet ports (Fig. 6.17). This attractive performance behavior would indicate that the ALAX architecture can comfortably accommodate real-world situations, because the delay at each port on average is uniform.

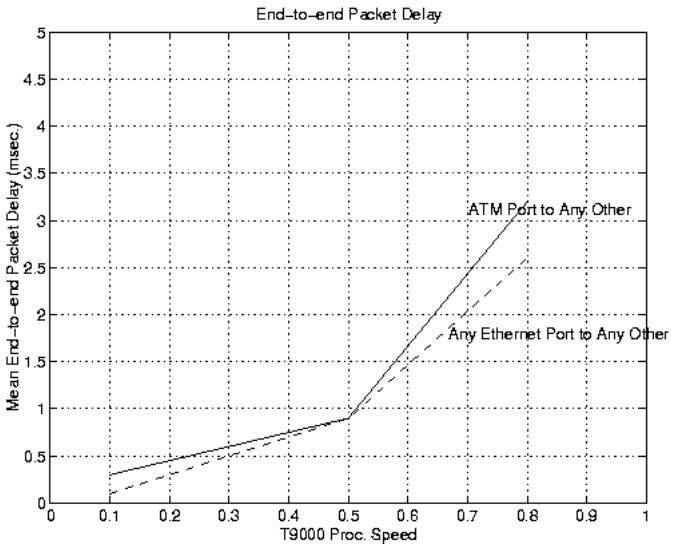

Figure 6.17: Average End-to-end Packet Delays

6.3 Outcomes under the SRD Traffic (MMPP)

The SRD-based traffic generators throughout the simulation runs involved also four levels of burst intensity, identified as ratios (see Chap. 5, Sec. 5.4). These ratios ranged from least bursty to most bursty at 1:1, 10:1, 50:1 and 500:1. Henceforth the experimentations were carried for each burst ratio and each T9000 processing speed level. Thus, the total number of simulation runs under the auspices of the SRD-type traffic for the 4-by-4 ALAX model is 16.

6.3.1 Queuing Analysis

For the ATM port, when the T9000 processing speed level is 0.1, at T9000 #1, the average size at *Q[0]* varies from 0.3 to 0.6, while the maximum size goes from 6.0 to 11.0 (Table A.21). This trend is observed as the burst ratio level is intensified accordingly from 1:1 to 500:1. Further in *Q[1]*, the average size also varies from 5.5 to 42.4, and the maximum size increases from 28.0 to 574.0. Indeed, here just like in the case of the LRD, seen previously, occupancies in *Q[1]* are much greater than *Q[0]*, as a result of the backpressure from the C104 switching device. Meanwhile, in T9000 #2 (Table A.22), the average sizes for *Q[3]*, *Q[4]* and *Q[5]* remain more or less steady at around 1.7, 1.2 and 1.0 respectively throughout the burstiness variation. However, the maximum size at these corresponding queues jump by about 20% as the burst ratio level is augmented.

Shifting the T9000 processing speed level from 0.1 to 0.5, results in a substantial increase in average and maximum sizes at *Q[0]* and *Q[1]* (see Table A.23, Figs. 6.18, 6.19, 6.20 and 6.21). Because, the processing speed is slowed down, more packets are now waiting to be served for LAN Emulation and MML at *Q[0]*; while at *Q[1]* more packets are prevented from moving on, due to the backpressure from the C104. A similar buildup is observed in T9000 #2 (Table A.24). However, the maximum size

at the queues remain the same as when the T9000 processing speed level was 0.1 . Stepping down further to slower processing speed levels at 0.8 and 1.0 (Tables A.25, A.26, A.27 and A.28), the growth in queue sizes is again noticeable (Fig. 6.22 and 6.23).

At all Ethernet ports, the queuing results are more or less the same. Hence, here we only acknowledge the results at Ethernet port 3. The statistics at the other Ethernet ports, for this scenario are available upon request. Tables A.29, A.30, A.31 and A.32 show that as the T9000 processing speed level is changed from 0.1, 0.5, 0.8 and 1.0 respectively, there was no direct or immediate effects on all the queues at Ethernet port 3, except for *Q[2]*. This is expected, since the processing speed level at the ATM port has no direct relationship to the other Ethernet ports. Nevertheless, it does affect the number of packets that they do receive from the ATM port, as revealed in the values for *Q[2]*. Eventually, as the processing speed decreases a substantial decrease in the average size at *Q[2]* is noted, but the maximum size remains the same throughout.

6.3.2 Delay Analysis

Tables A.33 and A.34 denote the average packet latencies at the ATM and Ethernet ports. In both tables, as the T9000 processing speed level goes down, the packet delay proportionally increases. So that the highest end-to-end delay is obtained for the slowest speed level at any burstiness ratio intensity (Figs. 6.24 and 6.25).

At the ATM port, when the T9000 processing speed level is 0.1, if the burst ratio is 1:1, the delay is 0.4 msec. Further, if the burstiness variation is introduced for 10:1, 50:1 and 500:1, there is an increase by 0.1 msec in each case from 0.4 msec. For a T9000 processing speed level of 0.5, the delay jumps to 1.4 msec, 1.6 msec, 1.7 msec and 1.8 msec respectively at 1:1, 10:1, 50:1 and 500:1 burst ratios. The same pattern

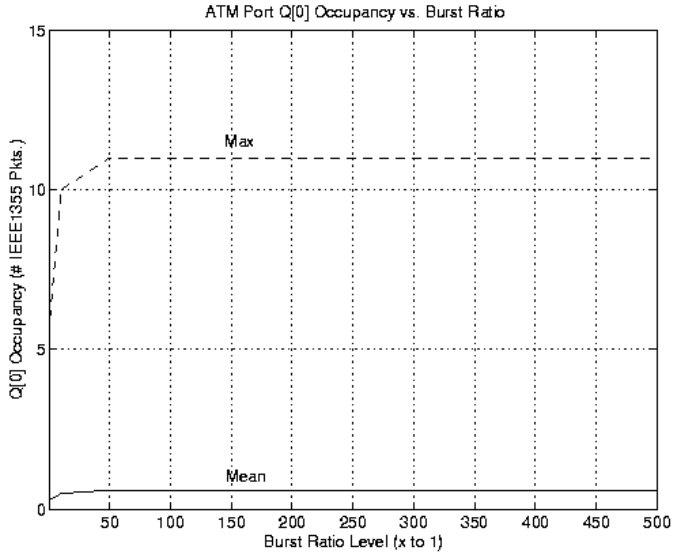

Figure 6.18: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

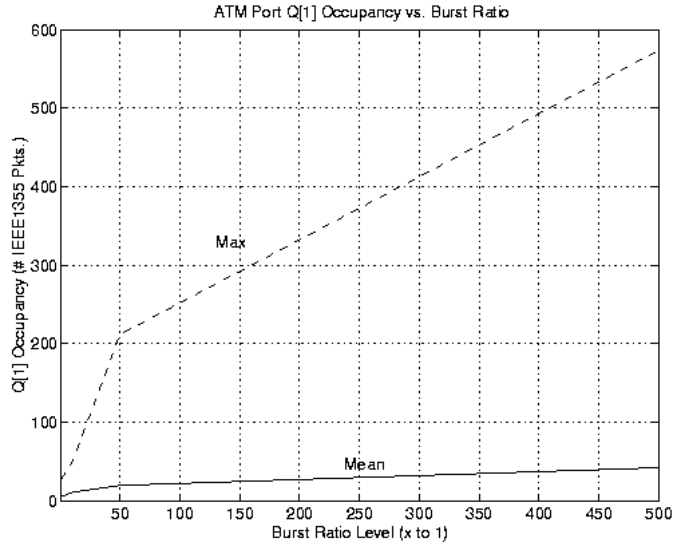

Figure 6.19: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

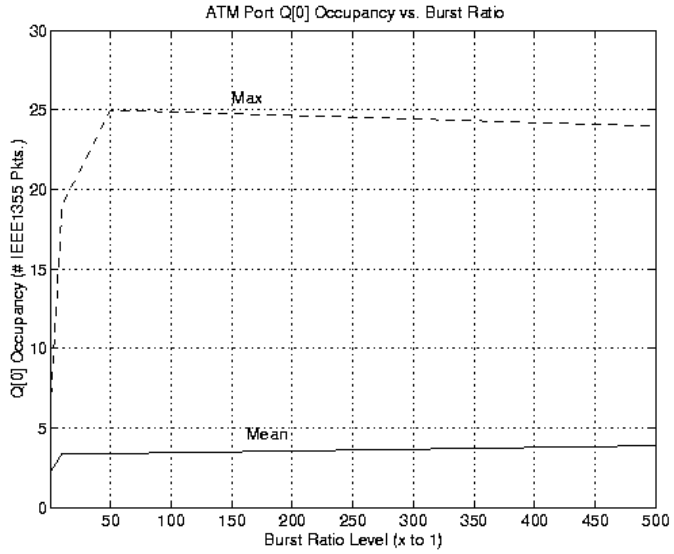

Figure 6.20: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

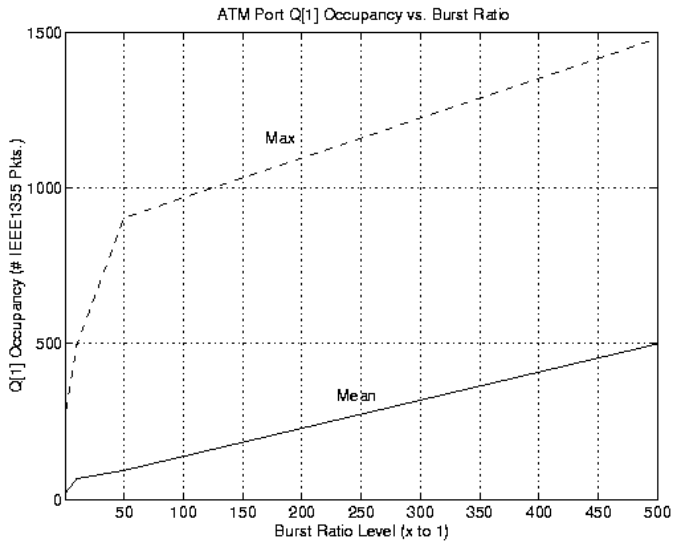

Figure 6.21: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

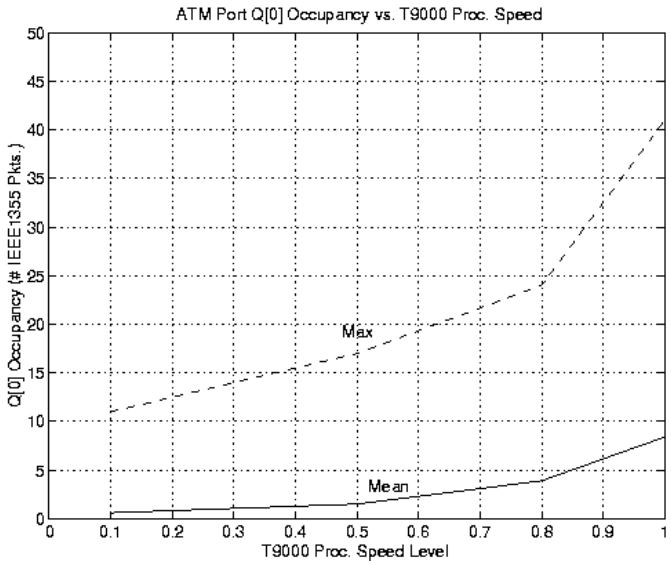

Figure 6.22: ATM port Q[0] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

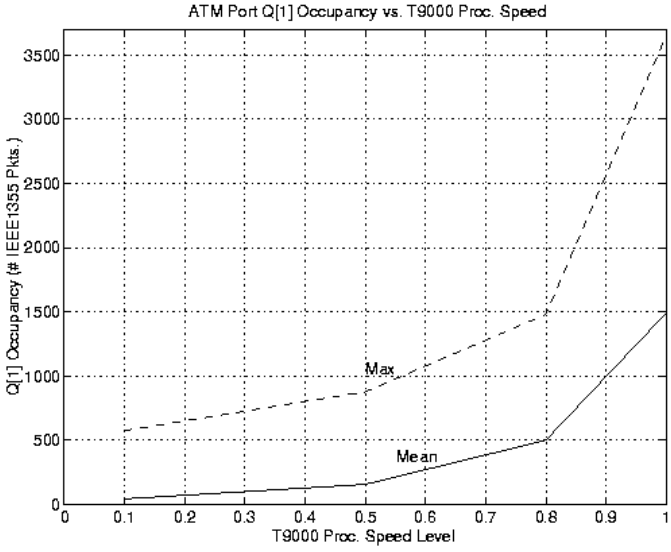

Figure 6.23: ATM port Q[1] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

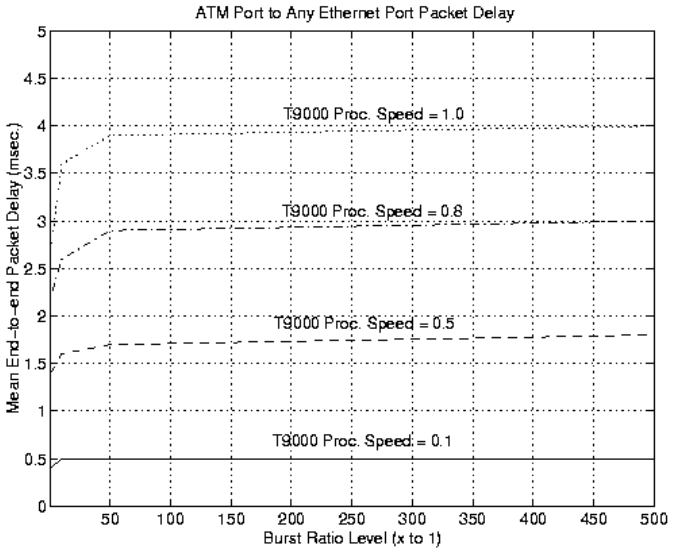

Figure 6.24: Average End-to-end Packet Delays from the ATM Port to Any Other

goes on for the two other speed levels.

Meanwhile at Ethernet port 3, the delays look at little bit more stable around the first decimal point, until the most bursty level at T9000 processing speed level of 0.8. Then, it degrades up to 3.4 msec. Overall, though these delays appear relatively small and acceptable.

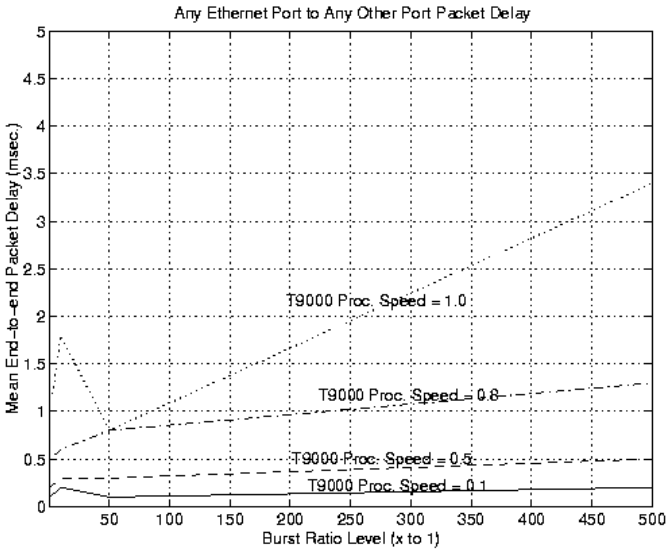

Figure 6.25: Average End-to-end Packet Delays from Any Ethernet Port to Any Other

6.4 Outcomes under the VBR Traffic (Star Wars Movie)

Within the context of simulating the 4-by-4 ALAX under the loads of a VBR traffic, the selected traffic was represented by the popular Star Wars movie. As the future holds the promise of video to the desktop over existing or emerging high-speed networks, this experiment is a projection of this imminent reality.

Each network port acted as a possible video server/client to all the other ports, so that there are in total 12 simultaneous video streams inside the ALAX switch. The T9000 processing speed level at the ATM port again, was varied from 0.1 to 1.0, and statistics were captured only for the average queue sizes and the average delays experienced by a packet going from one end port to any other of the ALAX switch. There are a total of 4 simulation runs for this case.

6.4.1 Queuing Analysis

Surveying T9000 #1 (Table A.85), the average queue sizes are seen inversely proportional to the T9000 processing speed levels (Fig. 6.53). Stated another way, as the T9000 processing speed level increases, the queue sizes decrease and vice versa. Further in T9000 #2 (Table A.86), the same characteristics are observed. But the queue sizes among *Q[3]*, *Q[4]* and *Q[5]* are of the same magnitude. In addition, they are more or less equivalent to the small sizes observed at *Q[0]* in T9000 #1 (Fig. 6.54).

On the other hand, all the Ethernet ports adopt the same behavior irrespective of the T9000 processing speed level (Tables A.87, A.88 and A.89). The queue sizes are the same for all these ports, except in *Q[2]* where there are some slight fluctuations. As previously seen, *Q[2]* is where packets coming from the ATM port are stored. Hence, these fluctuations reveal the variation of the traffic stream from the ATM port as the T9000 processing speed changes.

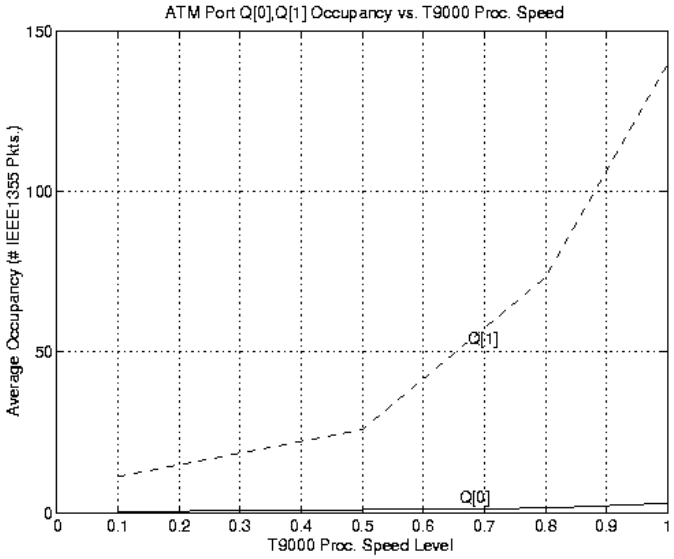

Figure 6.26: Average Occupancy for Q[0] and Q[1] at ATM port

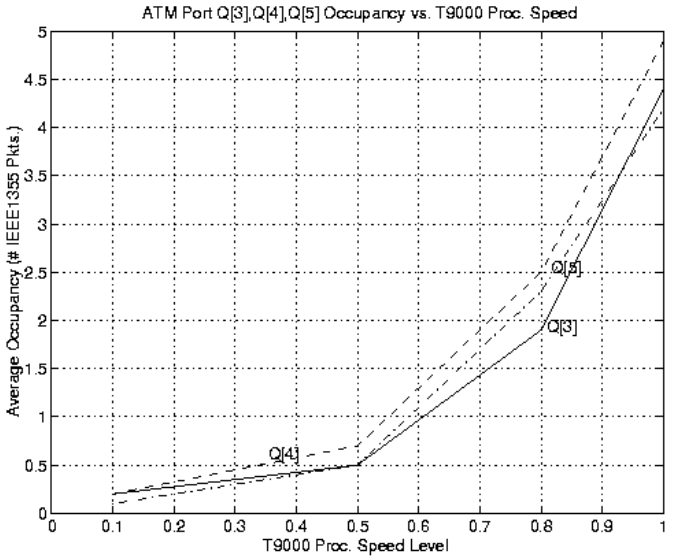

Figure 6.27: Average Occupancy for Q[3], Q[4] and Q[5] at ATM port

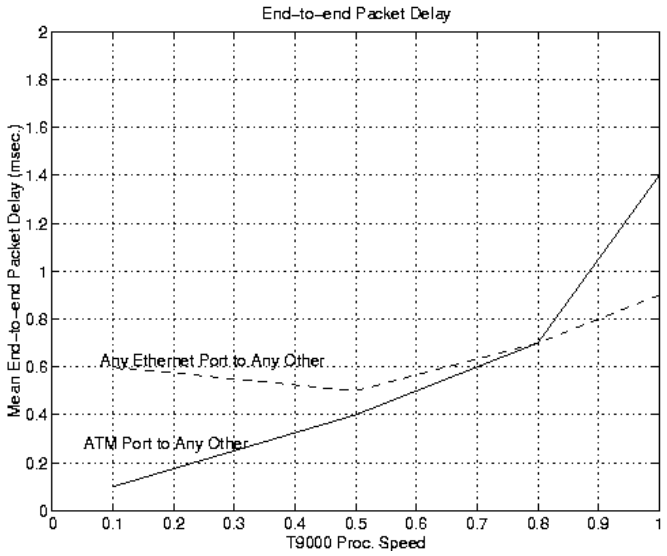

Figure 6.28: End-to-end Packet Delays

6.4.2 Delay Analysis

For the Star Wars traffic profile, the delay performance bear the same values at both Ethernet and ATM ports (see Table A.90 and Fig. 6.55). The same order of magnitude is noted overall for all the ports across the ALAX switch. Varying the T9000 processing speed level from slowest to fastest decreases the average end-to-end packet delays by almost 90% at the ATM port, while this decrease is approximately 30% at each Ethernet port. Broadly speaking, it seems that again a value between 0.0 and 0.8 could provide a superior performance of the ALAX switch. Note on the graph of Fig. 6.55 the curves for the ATM and Ethernet delays cross each other at the T9000 processing speed of 0.8.

6.5 Summary Note

The OPNET simulations for ALAX answered the questions and goals that it was set for. Relevant queue sizes were determined for the shared memory buffer

at each T9000 transputer. The most effective and optimal processing speed range for the T9000 was determined for the 4-by-4 ALAX model to be [0.0, 0.8]. For a small switch configuration with 4 data ports, the performance of the ALAX design and its architecture as reflected in the average end-to-end delays reported, seems well positioned overall, assuming of course the optimal speed is set and adequate memory provisioning is provided.

Chapter 7

The ALAX Simulation Experiments- Part II

This chapter outlines first the steps in modifying the original 4-by-4 ALAX simulation model into the 8-by-8, 12-by-12 and 16-by-16 models. Then, the methodology for conducting the simulation runs are discussed for all three of these cases indicated. These cases were studied using only the SRD-based traffic model. The results[1] are presented and interpreted for each case individually, and finally a comparison is made between all four sizes developed for the ALAX switch model.

7.1 Expanding the ALAX Simulation Model

7.1.1 Motivation

The creations of 8-port, 12-port and 16-port configurations of the ALAX switch are motivated by the interest to match comparable switch sizes on the market and also to test the scalability of the switch when the port size increases, because ALAX is designed with the capability to grow to a maximum of 16 data ports (see Chap. 4). Hence, having these models help in further establishing benchmarks in accordance

[1] These results include again average end-to-end packet delays and queuing sizes. Variances/standard deviations are not part of the analysis. They can be made available upon request.

with the industry, while providing data to assess the performance of the ALAX architecture when we go from a 4-by-4 to a 16-by-16 size.

7.1.2 Modifications Required

These models were built upon the original 4-by-4. There were minor changes to the architecture and organization, aside from an increase in the number of devices, e.g., transputers, for each Ethernet port added to the simulation model. Overall the system design was maintained, again accounting for the flexibility of the OPNET simulation tool. In all these models, one C104 was used. The only points where major changes occurred were in the internal operations of the C104 and the T9000 transputer. These alterations involved adding line of codes, increasing the internal number of sub-queues and some other general programming structures to accommodate the increase of ports and traffic in each case. The code for each model is provided in the appendix for references.

7.2 Experimental Methodology

The methodology for pursuing the simulations for the 8-by-8, 12-by-12 and 16-by-16 ALAX models are similar to the one discussed in the previous chapter. End-to-end packet delays and queuing sizes are still the parameters of interest. The same Ethernet and ATM packet sizes are maintained. And the tests are such that any port can send to any other port except itself. We still have one ATM port but the number of Ethernet ports grow with respect to the model at hand. Hence, there are 7, 11 and 15 Ethernet ports respectively for the 8-by-8, 12-by-12 and 16-by-16 ALAX models. The other significant difference is in the speed of the ATM port which changes for

each ALAX model, 28 Mbps, 44 Mbps and 60 Mbps while each Ethernet port speed remains at 4 Mbps. Finally, it is worth mentioning also that the steady-state operation of ALAX was maintained at 10 secs for the 8-by-8 case, however in the 12-by-12 and 16-by-16 models we used 15 secs as the most suitable value. In total there were 48 simulation runs executed for all these models, 16 for each.

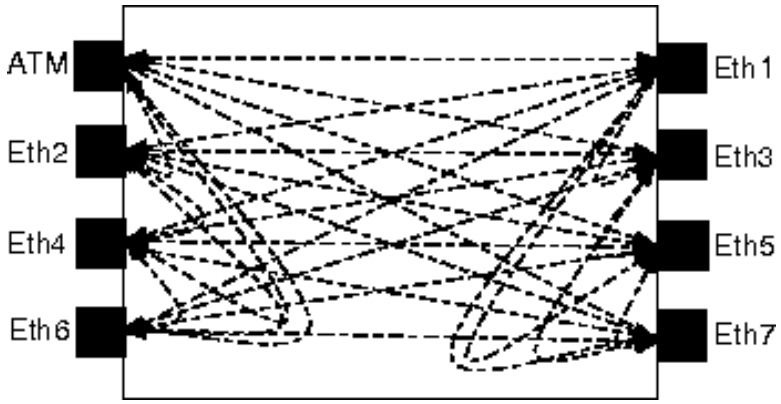

Figure 7.1: Mesh Connectivity inside the 8-by-8 ALAX model in OPNET

7.3 Testing with the 8-by-8 Model

7.3.1 Queuing Analysis

All the values collected for T9000 #1 (Tables A.35, A.37, A.39 and A.41) under each speed level of the T9000 indicate that as the burstiness is increased, there is a surge of packets that piles up into *Q[0]* and *Q[1]* (Figs. 7.2, 7.3, 7.4 and 7.5). However, *Q[1]* is much larger than *Q[0]* due to the backpressure from the C104. In T9000 #2 (Tables A.36, A.38, A.40 and A.42), the same effect happens for the queues storing incoming packets from other ports. The only obvious difference between these results and the ones from the 4-by-4 model seem to be that the queues have increased dramatically from their previous values. And naturally this is expected, for there are twice as many ports as before.

Just as in the previous cases, the Ethernet ports for the 8-by-8 model ranked the same in terms of maximum and average queue sizes. So we only report the results for Ethernet port 4. Again, the same values when T9000 processing speed level is 0.1 (Table A.43), are repeated more or less for T9000 processing speed levels of 0.5, 0.8 and 1.0 (Tables A.44, A.45 and A.46). The only queue that fluctuates is the one holding packets from the ATM port, *Q[2]* and here again the fluctuations are not

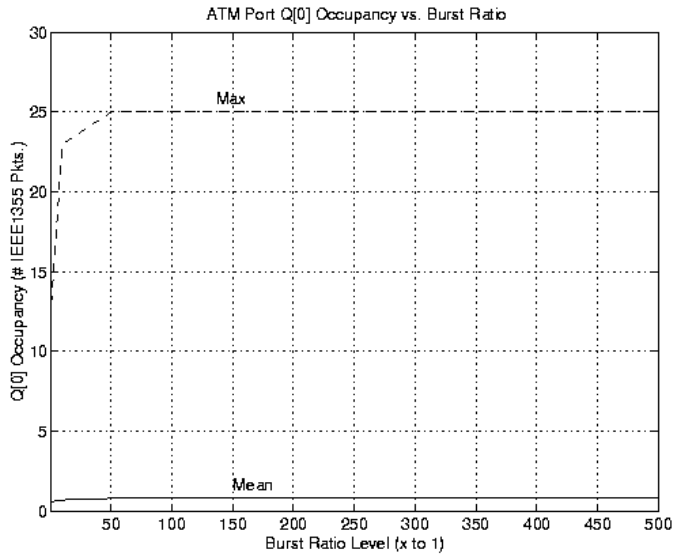

Figure 7.2: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

as accentuated as in the 4-by-4 model case. These values are very close to the ones observed in the 4-by-4 model for the all the Ethernet ports.

7.3.2 Delay Analysis

Tables A.47 and A.48 reflect the end-to-end packet delays for the ATM and Ethernet ports respectively. The striking feature in these results is noted at the Ethernet port, where for the high value of T9000 processing speed level, there are effectively low delays across the switch comparable to the ATM delays (Figs. 7.8 and 7.9). Nonetheless, as soon as the T9000 processing speed level hits the 1.0 mark, the immediate consequence is felt at the Ethernet ports, where the delays increase ten-fold.

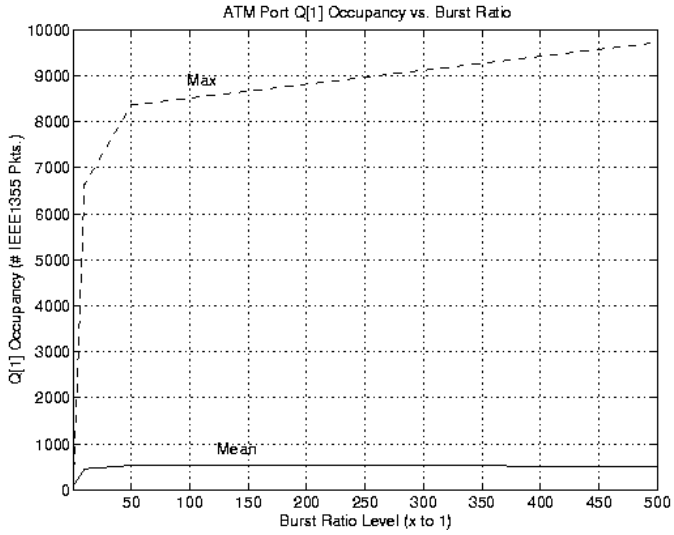

Figure 7.3: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

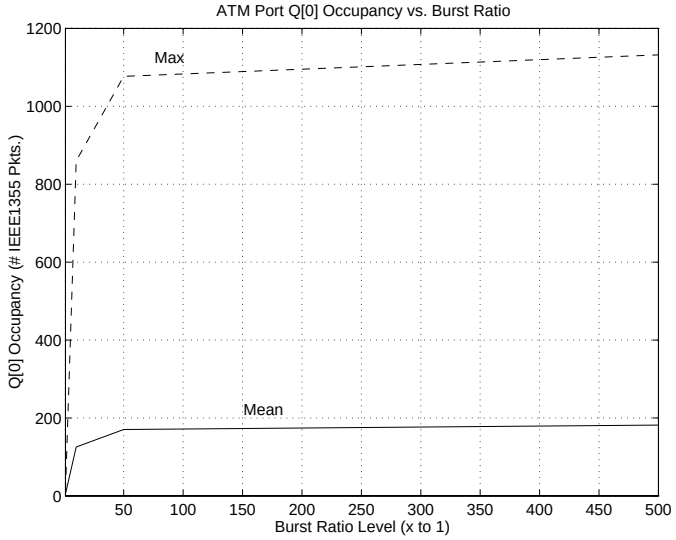

Figure 7.4: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

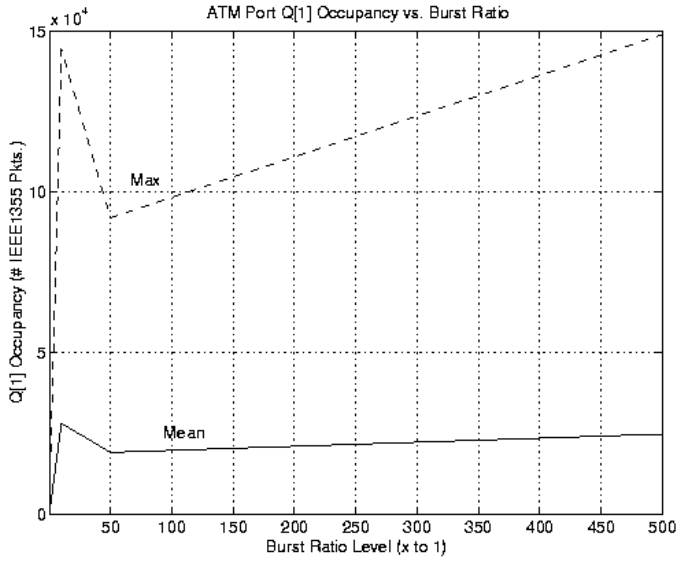

Figure 7.5: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

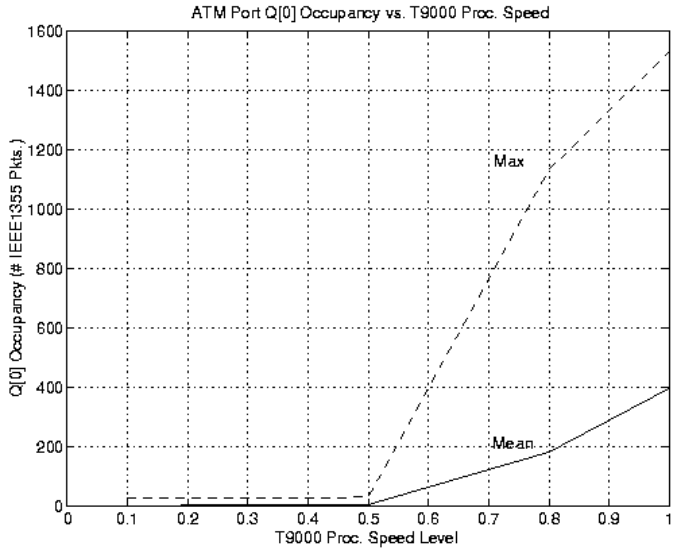

Figure 7.6: ATM port Q[0] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

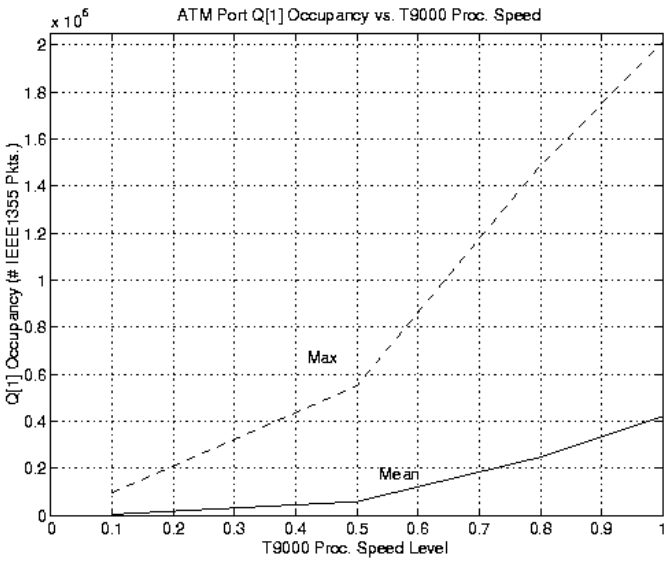

Figure 7.7: ATM port Q[1] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

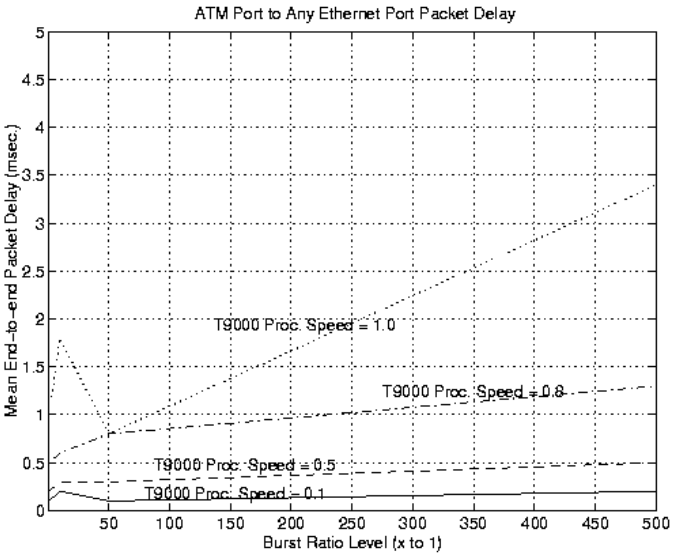

Figure 7.8: Average End-to-end Packet Delays from the ATM Port to Any Other

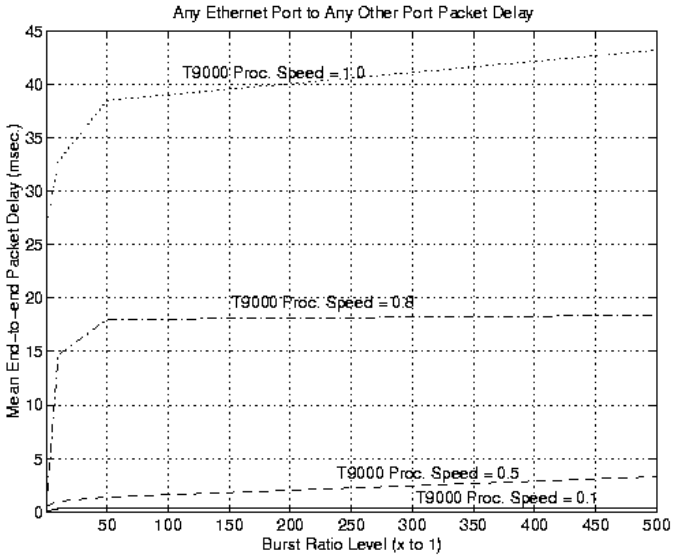

Figure 7.9: Average End-to-end Packet Delays from Any Ethernet Port to Any Other

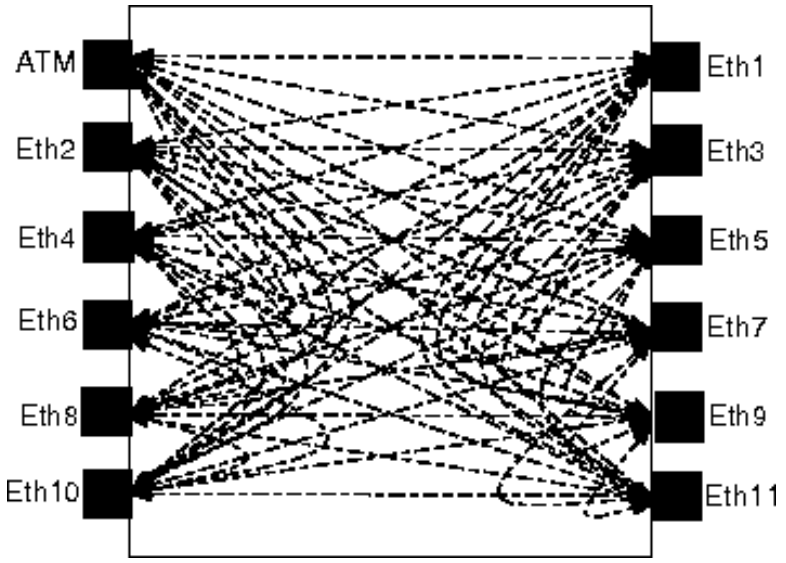

Figure 7.10: Mesh Connectivity inside the 12-by-12 ALAX model in OPNET

7.4 Testing with the 12-by-12 Model

7.4.1 Queuing Analysis

In T9000 #1, when the processing speed level is 0.1 (Table A.49), the average and maximum at *Q[0]* are relatively small compared to the ones with respect to *Q[1]* (Figs. 7.11 and 7.12). Once again, as acknowledged in previous cases, the large queue build-up at *Q[1]]* is most likely due to the C104 backpressure. When the processing speed is changed to 0.5, there is a small increase in the average at *Q[0]* (see Table A.51), while the maximum occupancy stays more or less the same. In the meantime, the consequences are different at *Q[1]*. A dramatic increase for both the average and maximum occupancy at *Q[1]* is noticed. The magnitude of the increase is dependent upon the burstiness level. A decrease in the speed of the T9000 to 0.8 (see Table A.53), results in further increase at *Q[1]* (Fig. 7.14). At *Q[0]*, for burstiness levels in the range of 10:1 to 500:1, as selected, a similar increase is felt (Fig. 7.13). However, for the burstiness level of 1:1, there is almost no increase observed since the same values are repeated. This could probably be a reflection of the low intensity exhibited by a

1:1 burst ratio. At a speed of 1.0 (Table A.55), the outcomes are identical to the ones for a speed of 0.8 .

For T9000 #2, there are no significant changes with regard to the occupancies of the queues under the conditions imposed by varying T9000 processing speed and burst levels. Tables A.50, A.52, A.54 and A.56 show that aside from slight fluctuations the queue occupancies are uniformly the same.

All 11 Ethernet ports for the 12-by-12 model ranked the same in terms of maximum and average queue sizes. Hence in Tables A.57, A.58, A.59 and A.60 we only show the results for Ethernet port 6. Results from the other Ethernet ports are available upon request.

Again, the same values when the T9000 processing speed level is 0.1, are repeated more or less for T9000 processing speed levels of 0.5, 0.8 and 1.0. This time though, there are no significant fluctuations for *Q[2]*, holding the packets from the ATM port. Note that in the 4-by-4 and 8-by-8 model cases, these fluctuations were more accentuated.

7.4.2 Delay Analysis

Tables A.61 and A.62 give an account on the packet latencies for the ATM and Ethernet ports respectively. These results are plotted in Figures 7.17 and 7.18.

In the first table, the delay increases proportionally as the T9000 processing speed level decreases for all the burstiness points selected. The figures for these delays stay stable throughout each processing speed level attained independently of the burstiness chosen. For the Ethernet ports, the delays also increase as the T9000 processing speed level decreases. Overall, we observe that when the T9000 processing speed is somewhere between 0.0 and 0.5, we can hope to achieve at the Ethernet port an almost identical delay performance as in the ATM port. But, when the speed is 0.8,

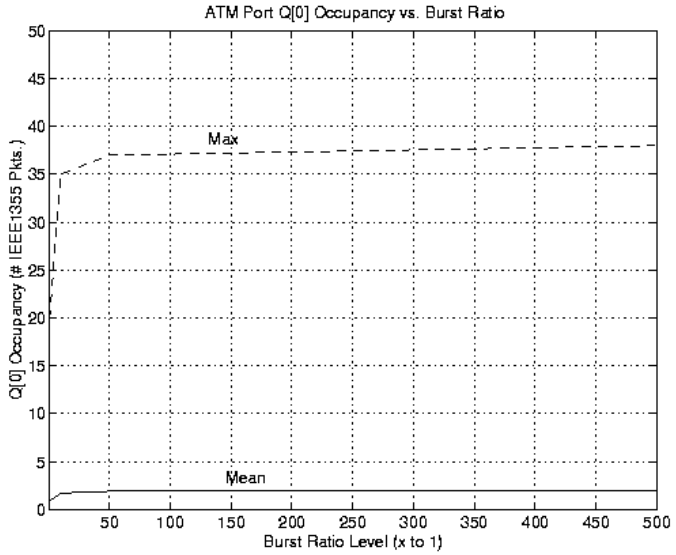

Figure 7.11: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

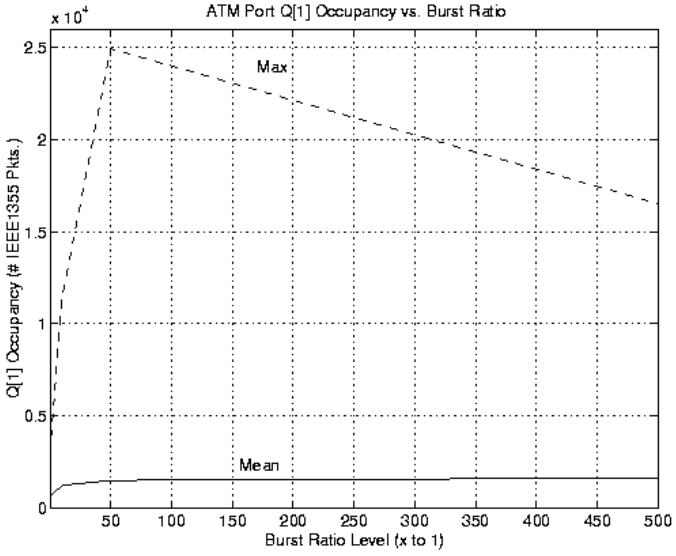

Figure 7.12: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

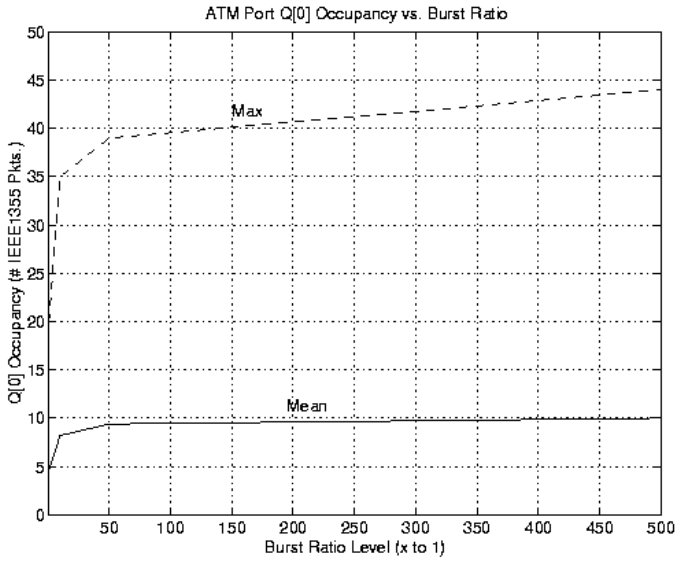

Figure 7.13: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

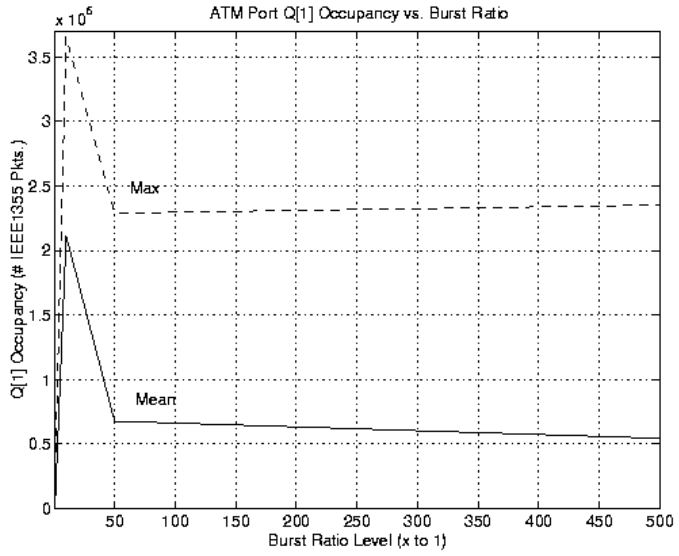

Figure 7.14: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

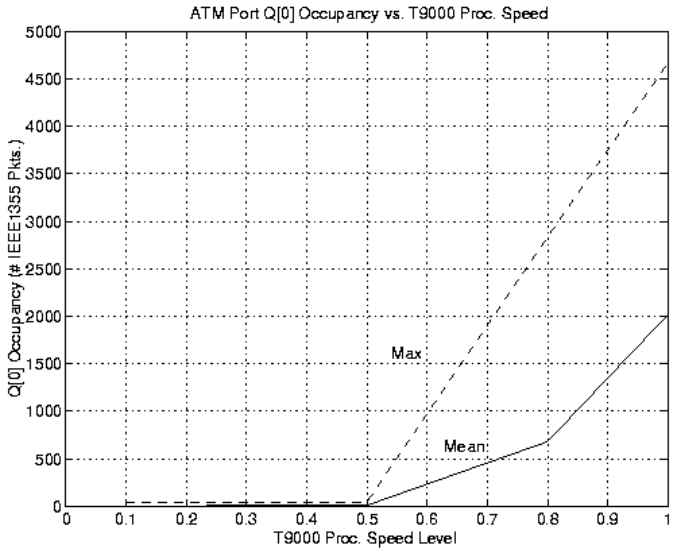

Figure 7.15: ATM port Q[0] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

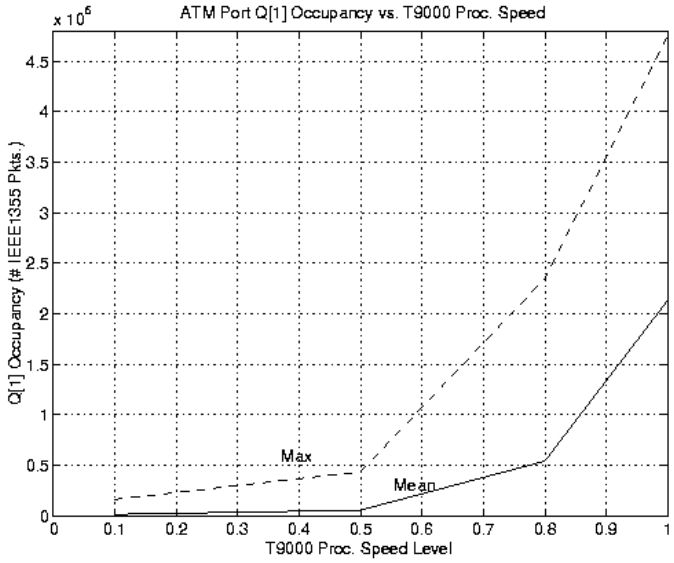

Figure 7.16: ATM port Q[1] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

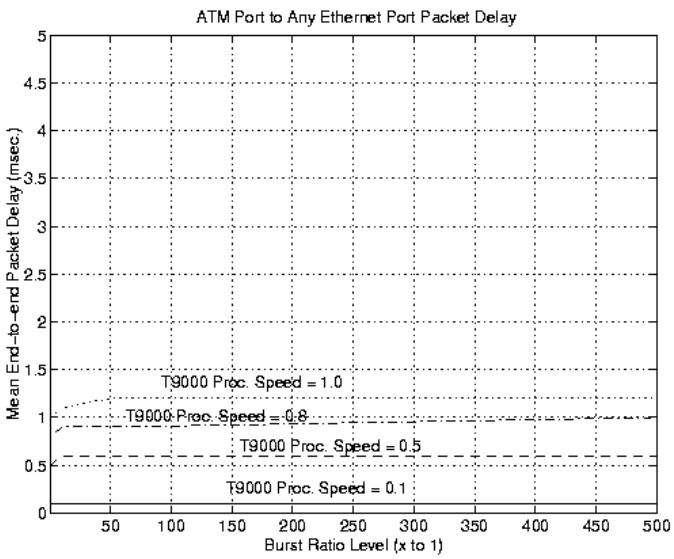

Figure 7.17: Average End-to-end Packet Delays from the ATM Port to Any Other

packets from the Ethernet ports face much more delay than the ATM one. And in this case, Ethernet delays above 30 msec should be rejected as this is an indication of poor performance compared to the accepted levels in the industry [43]. Beyond 0.8, the Ethernet packet delays can not be considered. Again, we can clearly see that the range of [0.0, 0.8] still holds, but [0.0, 0.5] may be even better.

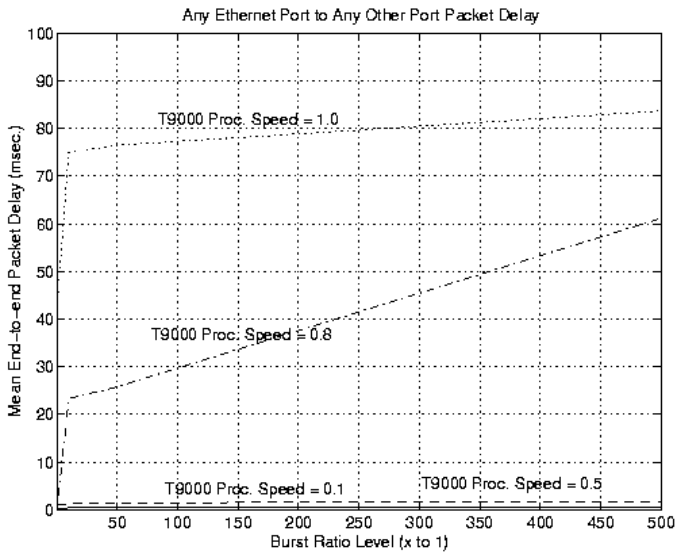

Figure 7.18: Average End-to-end Packet Delays from Any Ethernet Port to Any Other

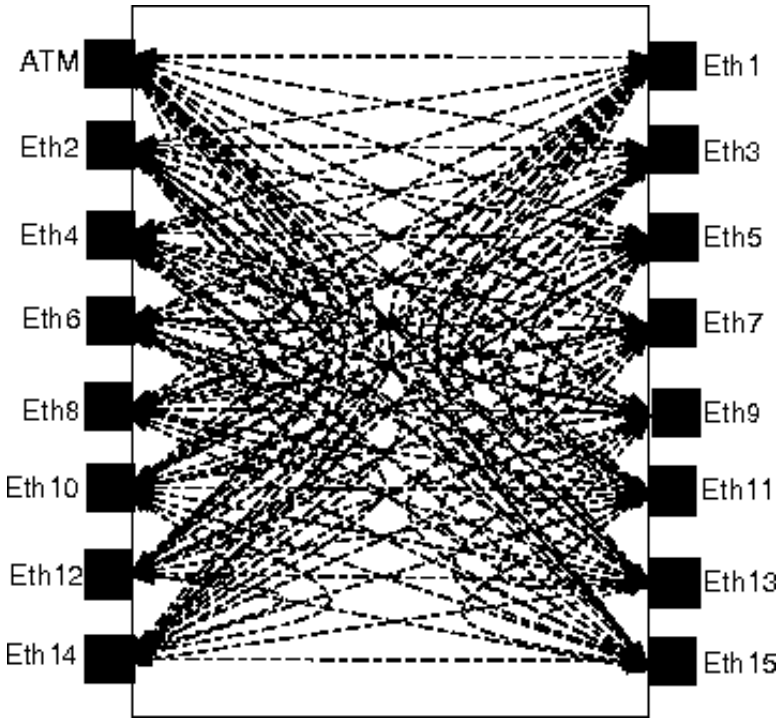

Figure 7.19: Mesh Connectivity inside the 16-by-16 ALAX model in OPNET

7.5 Testing with the 16-by-16 Model

7.5.1 Queuing Analysis

For T9000 # 1, when the T9000 processing speed level is 0.1, the average and maximum queue sizes at all burstiness levels for *Q[0]* are relatively small (Table A.63 and Figs. 7.20). Slowing down the speed to 0.5, increases the average queue size at *Q[0]* while the maximum size stays the same (Table A.66). In *Q[1]* the outcomes are different. For speeds, there is a net increase in average and maximum size proportionally to the burstiness levels. If the speed is 0.8 or 1.0, then the average and maximum queue sizes increase for both *Q[0]* and *Q[1]* (Tables A.69 and A.72) at all the burstiness points of interest (Figs. 7.22 and 7.23). The only exception can be extracted for *Q[0]* at a burstiness level of 1:1 .

Tables A.64, A.65, A.67, A.68, A.70, A.71, A.73 and A.74 summarize the effects

observed at the downstream T9000 queues (T9000 # 2). When the T9000 processing speed level is set at 0.1 or 0.5, as indicated in the first four tables above, the queue sizes are practically the same. The average size at each queue is very small, while the maximum is almost a thousand times what the average is. Conversely, if the speed is further decreased to 0.8 or 1.0, the outcomes are similar. Meaning that the average occupancy at each queue increases slightly, while the maximum remains the same.

As seen in the prior cases, all the Ethernet ports ranked the same in terms of magnitude of queue sizes. For this reason again, we only report the results at one port, Ethernet port 8. Data for other Ethernet ports are available upon request.

One general observation that can be made about the queue for Ethernet port 8, concerns the maximum occupancy. Aside from *Q[1]* where this maximum is relatively large, and *Q[0]* where this maximum is relatively small, for all the other queues (*Q[3]* to *Q[17*), the maximum occupancy approaches 70. And this is true for all four T9000 processing speeds (see Tables A.75 to A.82). Meanwhile, the average for all the queues at every speed level is of an order much less than 1; except at *Q[1]* and *Q[2]* where it oscillates between 1 and 2. These two queues (*Q[1]* and *Q[2]*) appear the most influenced by any changes in T9000 processing speeds, as expected. Indeed, the maximum occupancy at *Q[1]* is probably large because of the C104 backpressure, and as the T9000 speed decreases, the average slightly decreases while the maximum remain steady. *Q[2]* is directly linked to the packets sent by the ATM port, so any variation in T9000 speed there, would considerably affect *Q[2]*. This is noted in the slight increase or decrease of the average at *Q[2]* as the speed changes. One final observation can be traced to *Q[0]*. Since any Ethernet port only performs MML (contrary to the ATM) which is small with respect to delays, then *Q[0]* experiences a very small magnitude of occupancy for both the average and the maximum.

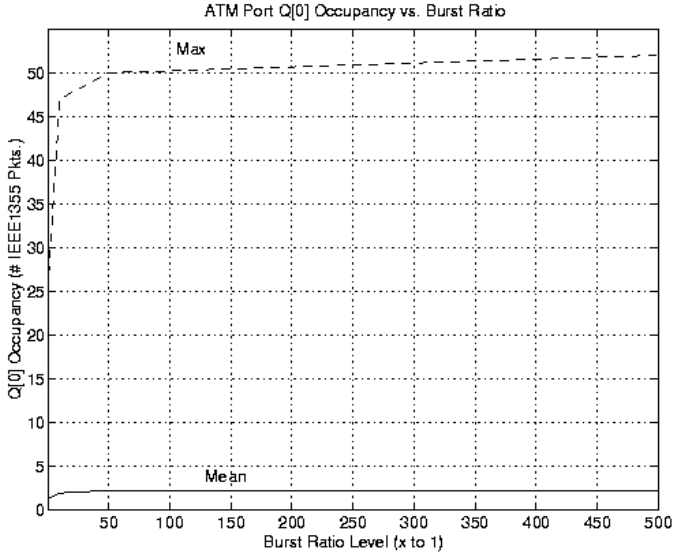

Figure 7.20: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

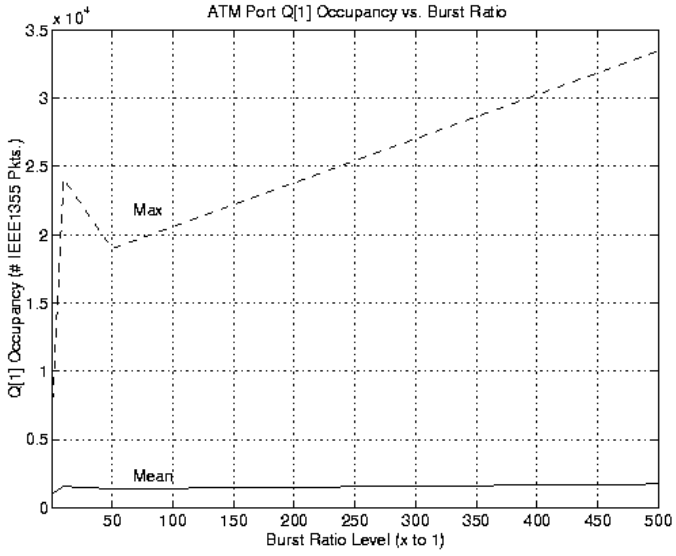

Figure 7.21: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.1

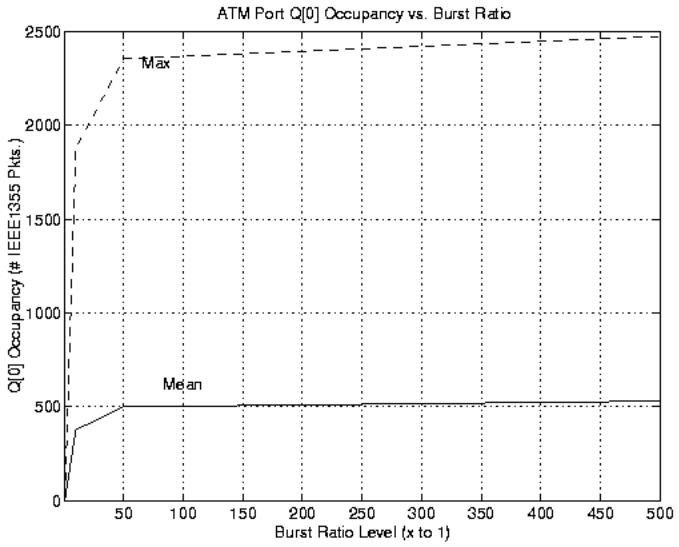

Figure 7.22: ATM port Q[0] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

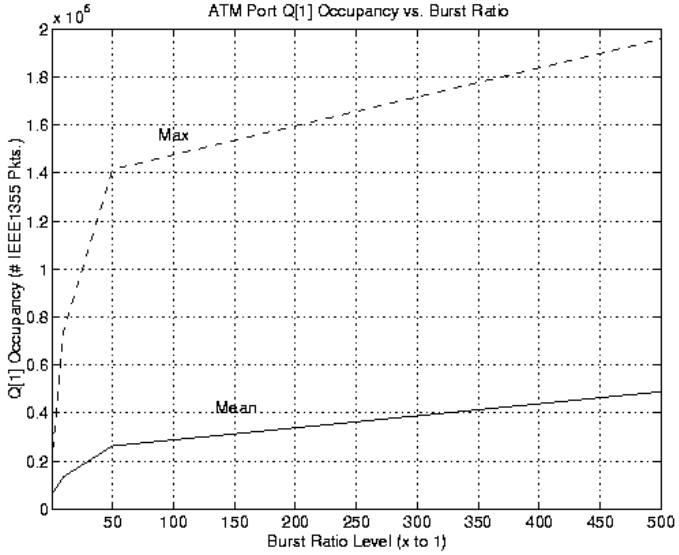

Figure 7.23: ATM port Q[1] Occupancy vs. Burst Ratio Level- T9000 Proc. Speed = 0.8

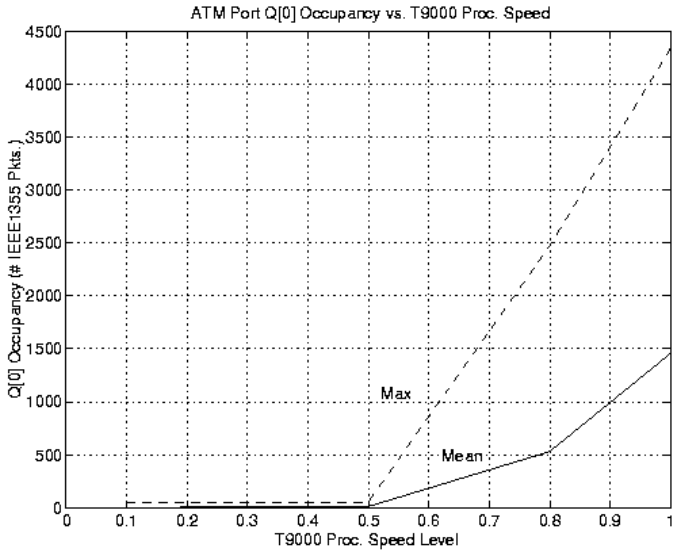

Figure 7.24: ATM port Q[0] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

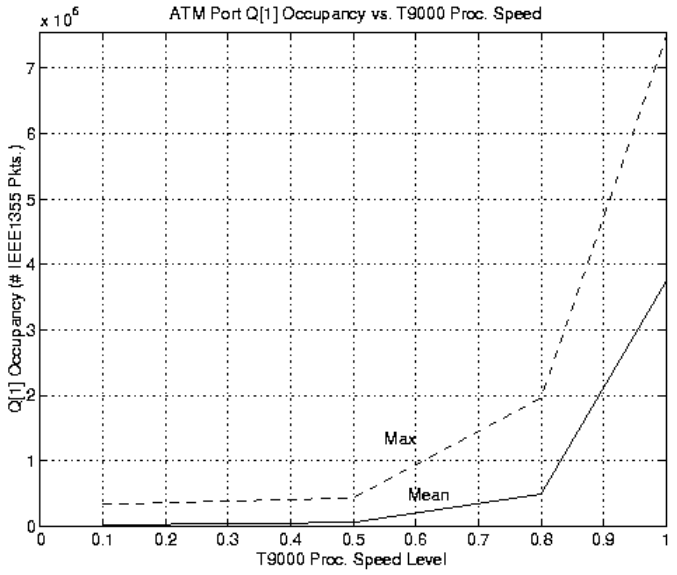

Figure 7.25: ATM port Q[1] Occupancy vs. T9000 Speed Level- Burst Ratio = 500:1

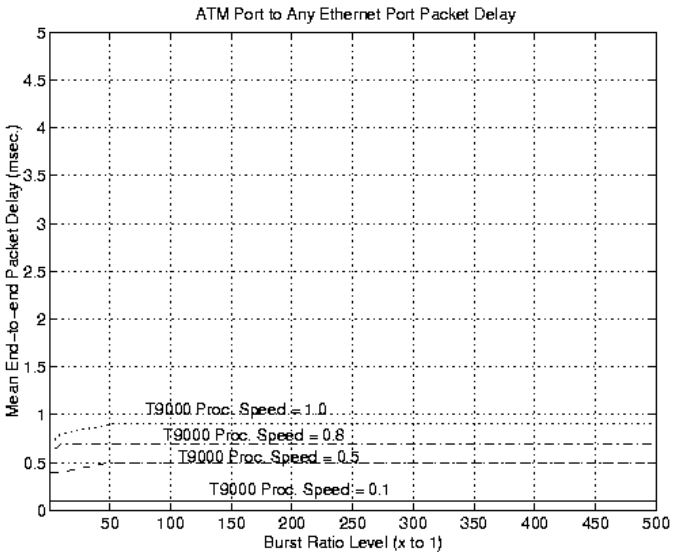

Figure 7.26: Average End-to-end Packet Delays from the ATM Port to Any Other

7.5.2 Delay Analysis

From the ATM port, the performance with regards to delay is great. Because across variations in traffic load burstiness or T9000 processing speed, the average end-to-end delay , experienced by packets coming from the ATM port and destined to any Ethernet ports of the 16-by-16 switch, is very small within 1 msec (see Table A.83 and the graph in Fig. 7.26)).

However, the average packet delay for any Ethernet port (Table A.84) varies in relation to the traffic burstiness or the T9000 processing speed level (See Fig. 7.27). At the T9000 processing speed level of 0.1, the delay stays well within 0.5 msec, no matter what the burstiness intensity is. But for 0.5 processing speed, it fluctuates between 1 and 1.5 msec. While for 0.8, the range of delay increases by a factor of 10, but still acceptable since it is below 30 msec [43]. When the speed becomes 1.0, the Ethernet delays are relatively large compared to the other faster speeds.

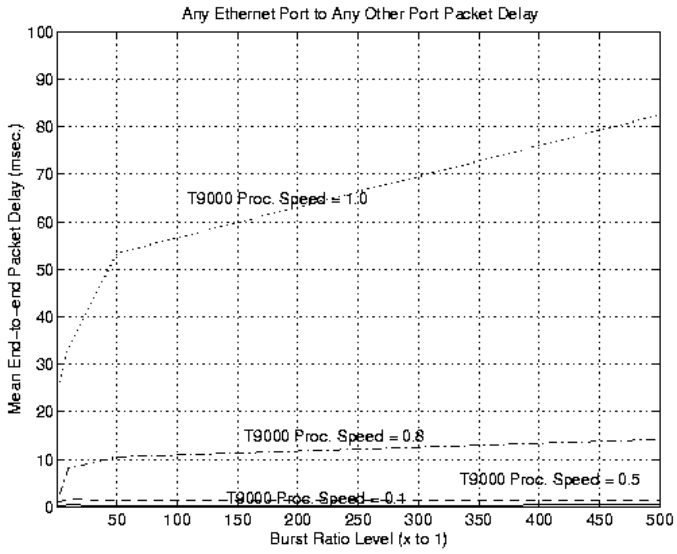

Figure 7.27: Average End-to-end Packet Delays from Any Ethernet Port to Any Other

7.6 Comparative Assessment of ALAX Sizes

In this section, we further investigate the effects inherent in the ALAX architecture as it is expanded from its basic 4-by-4 size to 8-by-8, 12-by-12 and 16-by-16.

7.6.1 Queuing Analysis

Looking at the ATM port results, the values for the maximum queue occupancy are repeated throughout T9000 # 2 of the 4-by-4, 8-by-8, 12-by-12 and 16-by-16 models. These values are not exactly the same figures, but they are well within the same order of magnitude and with very minimal fluctuations. This is a good sign, as it may be an acceptable indication for the scalability of the ALAX switch architecture. The average queue sizes though, vary depending on the size of the switch as much as it did with respect to the T9000 processing speed and traffic burstiness intensity. One obvious trend is that the average size of all queues at T9000 # 2 seem to decrease as the switch size is increased, perhaps due to the speed scalability of ATM. These patterns do not occur quite in the same fashion at T9000 # 1. The average queue size for *Q[1]* increases with the switch size and so does the maximum. This is expected since more packets are generated for every port increase in the switch, and the C104 still executes its backpressure mechanism to prevent flooding. It is worth noting also, that *Q[0]* in all four cases of the ALAX switch model is always maintained relatively small in size compared to *Q[1]*. This is again in accordance with our overall architecture as seen previously in the 4-by-4 case (Chapter 6).

When comparing all the results compiled for the T9000 at each Ethernet port, in each case of the ALAX switch sizes, one outstanding phenomenon that emerges is that the same maximum occupancy occurs in all the ALAX models. The only differences are detected in the average size fluctuations. This is probably a result of injecting

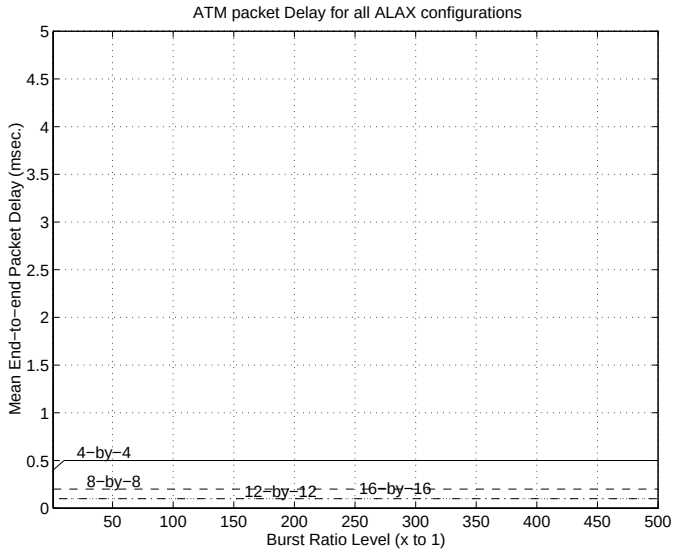

Figure 7.28: Average End-to-end Packet Delays at the ATM port for all ALAX sizes (with a T9000 Proc. Speed = 0.1)

packets at the same rate for each Ethernet port.

7.6.2 Delay Analysis

The average end-to-end delay for any packet leaving the ATM port seems to be well maintained at around 1 msec for all four ALAX switch models, within the T9000 speed range of 0 to 1.0 . The same can not be said for the average end-to-end delay at the Ethernet ports. In this case, the observations point out that large delays of the order of 40 to 80 msec can occur specifically when the T9000 processing speed is above 0.8, and especially in the 12-by-12 and 16-by-16 models. Given that the average latency tested and reported by the Tolly Group for the typical fastest Ethernet switch encountered in the industry lies around 30 msec [43], these delays are probably unacceptable. Taking this fact into account would lead us to favor a much smaller range [0.0,0.5] instead of [0.0,0.8] as previously established in the 4-by-4 case.

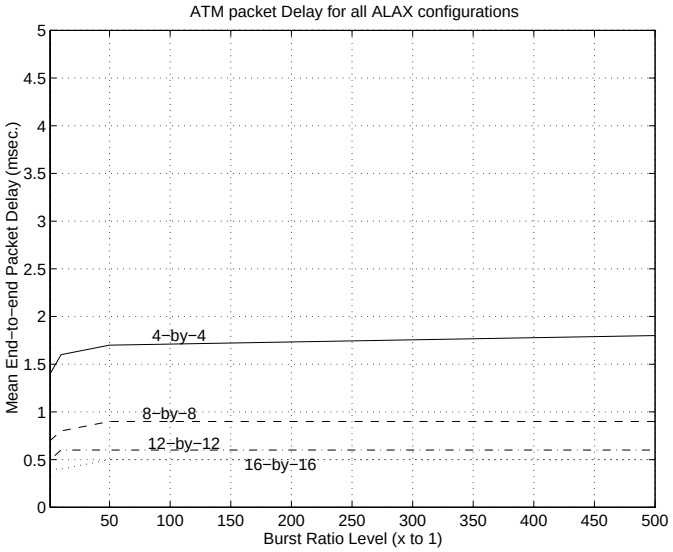

Figure 7.29: Average End-to-end Packet Delays at the ATM port for all ALAX sizes (with a T9000 Proc. Speed = 0.5)

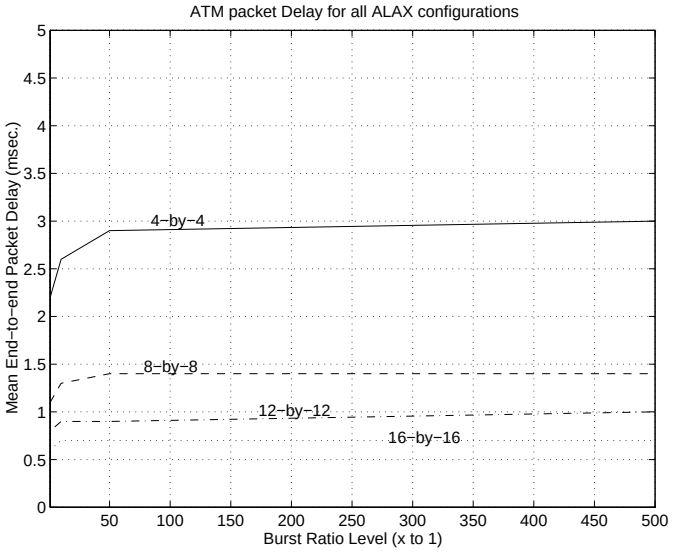

Figure 7.30: Average End-to-end Packet Delays at the ATM port for all ALAX sizes (with a T9000 Proc. Speed = 0.8)

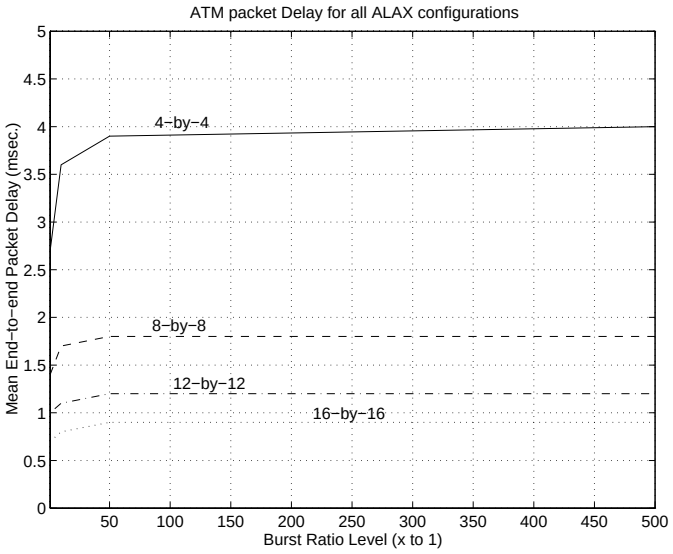

Figure 7.31: Average End-to-end Packet Delays at the ATM port for all ALAX sizes (with a T9000 Proc. Speed = 1.0)

7.7 Summary Note

With the flexibility of the OPNET simulation tool, it was not a much difficult task to modify the existing 4-by-4 ALAX model into 8-by-8, 12-by-12 and 16-by-16 configurations. We noticed that, in each individual ALAX model simulated that the performance level was still acceptable in the ranges previously established under the 4-by-4 case. The queue sizes did grow proportionally with the traffic load encountered on each case simulated and as the burstiness increased and the T9000 speed decreased. However, all the four models did not perform consistently in the same manner. Large delays and substantial queue buildup in the 12-by-12 model were experienced when the T9000 was running at the acceptable worst speed of 0.8. Nonetheless, we did not observe the same trend in the 16-by-16 case under the same conditions. So, this finding even though not convincing totally, could probably indicate the need to shrink the optimal speed range of the T9000 to [0.0, 0.5] instead of [0.0,0.8] as previously agreed. Note that further investigations are needed to understand why this behavior is occurring.

Overall, though relevant queue sizes have been determined for the shared memory buffer at each T9000 transputer. The performance of the ALAX switch itself growing from 4 to 16 ports is fairly scaleable to a great extent provided of course an adequate speed is selected from the range established and sufficient buffering is provided as indicated by our results.

Chapter 8

Conclusions and Future Growth

This chapter presents a summary of recommendations and conclusions to aid in the design implementation of ALAX. This summary is based on the analysis and results from previous chapters. The Future Growth section presents other ways to extend the architecture developed for ALAX into other areas.

8.1 Conclusions

Through the OPNET simulation experiments, several results have been collected for the design and implementation of the ALAX switch.

First of all, the number of T9000 transputers increased to 2 instead of one as in the original design. This modification was prompted by early indications of poor performance of the ALAX switch with one transputer at the ATM port. And consequently, the subsequent observations in the simulation experiments have shown much less bottleneck at the ATM port than as previously experienced.

Secondly, since the implementation of the ALAX prototype requires adequate capacity sizing of memory, good estimates for these memory buffers at each port of the switch would probably be the maximum sizes as reported in the tables with some provisioning of a margin of 15 to 20 % above the actual figures. The following set of

	Size (IEEE 1355 Pkts.)
Q[0]	88
Q[1]	4706
Shared Memory	4794 (153 KB)

Table 8.1: Recommended Queue Size Estimates for 4-by-4 ALAX with LRD-type Traffic, ATM side T9000 #1

	Size (IEEE 1355 Pkts.)
Q[3]	491
Q[4]	490
Q[5]	490
Shared Memory	1471 (50 KB)

Table 8.2: Recommended Queue Size Estimates for 4-by-4 ALAX with LRD-type Traffic, ATM side T9000 #2

tables highlight these estimated design figures for each of the scenarios tested[1].

Thirdly, the optimal speed for the worst-case scenario (i.e., least fast but stable operation of the ALAX switch) was determined to be 0.8, when the model to be implemented is a 4-by-4. When expanding the switch size by adding more ports, we still could maintain acceptable performance at the processing speed of 0.8 under the SRD-type traffic profile. But there are clear indications that if we are designing up to the 16 ports, this optimal speed should be 0.5 instead.

Finally, the ALAX switch architecture appears to be fairly scalable up to 16 ports. This can be inferred because overall the delays were low when the switch was augmented from 4, 8, 12 and 16 ports with a T9000 processing speed range of [0.0, 0.8]. Thus, if we were to provision the ATM port accordingly with respect to the other ports, ALAX would be able to achieve a scalable performance with minimal delays.

[1]KB = KiloByte, MB = MegaByte

	Size (IEEE 1355 Pkts.)
Q[0]	1
Q[1]	48
Q[2]	66
Q[3]	49
Q[4]	49
Shared Memory	213 (6.8 KB^2)

Table 8.3: Recommended Queue Size Estimates for 4-by-4 ALAX with LRD-type Traffic, Ethernet side T9000

	Size (IEEE 1355 Pkts.)
Q[0]	24
Q[1]	1480
Shared Memory	1540 (48 KB)

Table 8.4: Recommended Queue Size Estimates for 4-by-4 ALAX with SRD-type Traffic, ATM side T9000 #1

	Size (IEEE 1355 Pkts.)
Q[3]	588
Q[4]	441
Q[5]	441
Shared Memory	1470 (47 KB)

Table 8.5: Recommended Queue Size Estimates for 4-by-4 ALAX with SRD-type Traffic, ATM side T9000 #2

	Size (IEEE 1355 Pkts.)
Q[0]	11
Q[1]	538
Q[2]	67
Q[3]	67
Q[4]	67
Shared Memory	750 (24 KB)

Table 8.6: Recommended Queue Size Estimates for 4-by-4 ALAX with SRD-type Traffic, Ethernet side T9000

	Size (IEEE 1355 Pkts.)
Q[0]	1132
Q[1]	148852
Shared Memory	149984 (4.8 MB)

Table 8.7: Recommended Queue Size Estimates for 8-by-8 ALAX with SRD-type Traffic, ATM side T9000 #1

	Size (IEEE 1355 Pkts.)
Q[3]	441
Q[4]	539
Q[5]	441
Q[6]	686
Q[7]	441
Q[8]	490
Q[9]	539
Shared Memory	3577 (114 KB)

Table 8.8: Recommended Queue Size Estimates for 8-by-8 ALAX with SRD-type Traffic, ATM side T9000 #2

	Size (IEEE 1355 Pkts.)
Q[0]	11
Q[1]	538
Q[2]	67
Q[3]	66
Q[4]	66
Q[5]	66
Q[6]	66
Q[7]	66
Q[8]	66
Q[9]	66
Shared Memory	1078 (35 KB)

Table 8.9: Recommended Queue Size Estimates for 8-by-8 ALAX with SRD-type Traffic, Ethernet side T9000

	Size (IEEE 1355 Pkts.)
Q[0]	2835
Q[1]	228930
Shared Memory	231765 (7.4 MB)

Table 8.10: Recommended Queue Size Estimates for 12-by-12 ALAX with SRD-type Traffic, ATM side T9000 #1

	Size (IEEE 1355 Pkts.)
Q[3]	441
Q[4]	441
Q[5]	490
Q[6]	392
Q[7]	441
Q[8]	441
Q[9]	441
Q[10]	441
Q[11]	441
Q[12]	490
Q[13]	490
Shared Memory	4949 (158 KB)

Table 8.11: Recommended Queue Size Estimates for 12-by-12 ALAX with SRD-type Traffic, ATM side T9000 #2

	Size (IEEE 1355 Pkts.)
Q[0]	11
Q[1]	538
Q[2]	71
Q[3]	66
Q[4]	66
Q[5]	66
Q[6]	66
Q[7]	66
Q[8]	66
Q[9]	66
Q[10]	66
Q[11]	66
Q[12]	66
Q[13]	66
Shared Memory	1346 (43 KB)

Table 8.12: Recommended Queue Size Estimates for 12-by-12 ALAX with SRD-type Traffic, Ethernet side T9000

	Size (IEEE 1355 Pkts.)
Q[0]	2474
Q[1]	195890
Shared Memory	198364 (6.3 MB)

Table 8.13: Recommended Queue Size Estimates for 16-by-16 ALAX with SRD-type Traffic, ATM side T9000 #1

	Size (IEEE 1355 Pkts.)
Q[3]	441
Q[4]	441
Q[5]	392
Q[6]	490
Q[7]	343
Q[8]	392
Q[9]	343
Q[10]	539
Q[11]	539
Q[12]	294
Q[13]	441
Q[14]	441
Q[15]	441
Q[16]	343
Q[17]	392
Shared Memory	6272 (200 KB)

Table 8.14: Recommended Queue Size Estimates for 16-by-16 ALAX with SRD-type Traffic, ATM side T9000 #2

	Size (IEEE 1355 Pkts.)
Q[0]	11
Q[1]	538
Q[2]	69
Q[3]	66
Q[4]	66
Q[5]	66
Q[6]	66
Q[7]	66
Q[8]	66
Q[9]	66
Q[10]	66
Q[11]	66
Q[12]	66
Q[13]	66
Q[14]	66
Q[15]	66
Q[16]	66
Q[17]	66
Shared Memory	1542 (49 KB)

Table 8.15: Recommended Queue Size Estimates for 16-by-16 ALAX with SRD-type Traffic, Ethernet side T9000

8.2 Considerations for Future Growth

We could investigate other areas with the current architecture and the simulation model of ALAX. These include:

1. Use other performance metrics besides end-to-end delay such as utilization, throughput, efficiency or cost/performance ratio.

2. we could develop the approaches to integrate MPOA inside ALAX. And also explore the application of the concepts of Cells In Frame (CIF) into the ALAX architecture.

3. Using the current simulation of ALAX, we could conduct more simulation runs with 8-by-8, 12-by-12, 16-by-16 under the LRD Traffic, execute simulations at nominal speeds of each physical port as designed and finally model ALAX using a concurrent or parallel simulation approach, since ALAX is based on a multi-processor system with parallel architecture capabilities.

4. The current ALAX model can be enhanced by remodeling the Transputer with a dynamic/adaptive processing speed, modeling the control plane and add it to the current ALAX switch, expanding the LANE implementation and add it to the current simulation model and also, by inserting ALAX into an actual network scenario.

5. For extending the ALAX internetworking span, several considerations are: Frame Relay, Fast Ethernet, Token Ring, xDSL Family, MPEG, ISDN and Satellite on-board switching, this topic was properly addressed in a technical document titled "'IEEE 1355-Based Architecture for an ATM Switch- A Case for On-board Switching and Processing" [25,26].

Appendix A

ALAX Simulation Results

This appendix contains selected sets of results pertaining to the performance evaluation and the conclusions derived in Chapter 6. It is divided according to the three types of traffic used in the study : LRD, SRD and VBR. In the case of the SRD traffic, the results are further subdivided with respect to each model of the ALAX switch (i.e., 4-by-4, 8-by-8, 12-by-12 and 16-by-16). The complete set of the results including variance measurements can be made available upon request.

A.1 Outcomes under the LRD Traffic ($M/G/\infty$)

A.1.1 With the 4-by-4 Model (Destns. Uniformly Distributed)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	0.08	0.08	0.08	0.08	0.09
Max. @ Q[0]	1.0	1.0	1.0	1.0	1.0
Avg. @ Q[1]	93.0	73.7	73.2	57.1	37.9
Max. @ Q[1]	1794.0	1794.0	1182.0	616.0	310.0

Table A.1: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.1)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[3]	0.5	0.7	0.7	0.6	0.6
Max. @ Q[3]	49.0	49.0	49.0	49.0	49.0
Avg. @ Q[4]	0.5	0.6	0.6	0.6	0.6
Max. @ Q[4]	49.0	49.0	49.0	49.0	49.0
Avg. @ Q[5]	0.6	0.6	0.5	0.5	0.6
Max. @ Q[5]	49.0	49.0	49.0	49.0	49.0

Table A.2: Queue Occupancy in T9000 #2: traffic coming from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.1)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	13.4	7.4	4.0	2.2	1.3
Max. @ Q[0]	79.0	41.0	22.0	11.0	6.0
Avg. @ Q[1]	1032.1	611.4	339.7	184.0	107.1
Max. @ Q[1]	4698.0	2444.0	1248.0	650.0	328.0

Table A.3: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.5)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[3]	2.6	4.0	3.4	3.3	3.0
Max. @ Q[3]	49.0	245.0	147.0	98.0	98.0
Avg. @ Q[4]	2.2	3.5	2.9	3.2	3.0
Max. @ Q[4]	49.0	245.0	147.0	98.0	98.0
Avg. @ Q[5]	2.9	3.5	2.7	2.9	3.0
Max. @ Q[5]	49.0	196.0	147.0	98.0	98.0

Table A.4: Queue Occupancy in T9000 #2: traffic coming from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.5)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	24.0	13.2	7.2	4.0	2.3
Max. @ Q[0]	88.0	46.0	24.0	13.0	7.0
Avg. @ Q[1]	1610.8	894.2	466.6	244.2	163.1
Max. @ Q[1]	4706.0	2466.0	1310.0	780.0	544.0

Table A.5: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.8)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[3]	19.0	16.5	9.5	7.3	5.5
Max. @ Q[3]	491.0	490.0	262.0	164.0	115.0
Avg. @ Q[4]	16.0	15.2	9.0	7.1	5.7
Max. @ Q[4]	490.0	453.0	263.0	166.0	113.0
Avg. @ Q[5]	20.3	20.3	7.8	6.5	5.7
Max. @ Q[5]	490.0	490.0	257.0	180.0	114.0

Table A.6: Queue Occupancy in T9000 #2: traffic coming from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.8)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	31.0	17.0	9.2	5.0	2.9
Max. @ Q[0]	91.0	47.0	25.0	13.0	7.0
Avg. @ Q[1]	2502.2	1619.6	942.8	553.2	316.7
Max. @ Q[1]	8598.0	6281.0	3737.0	2786.0	2056.0

Table A.7: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 1.0)

Queue Size (# IEEE 1355 pkts)	Burstiness Frequency Level (sec.)				
	7.5	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[3]	35.0	27.7	14.8	10.4	7.6
Max. @ Q[3]	743.0	583.0	311.0	180.0	117.0
Avg. @ Q[4]	29.4	25.8	14.2	10.2	7.8
Max. @ Q[4]	735.0	551.0	294.0	178.0	113.0
Avg. @ Q[5]	37.1	24.0	12.2	9.3	7.8
Max. @ Q[5]	735.0	512.0	294.0	180.0	116.0

Table A.8: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 1.0)

Queue Size (# IEEE 1355 pkts)	Burstiness Frequency Level (sec.)				
	7.5	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	0.002	0.003	0.003	0.003	0.004
Max. @ Q[0]	1.0	N/A	N/A	N/A	1.0
Avg. @ Q[1]	0.3	0.2	0.3	0.3	0.2
Max. @ Q[1]	48.0	N/A	N/A	N/A	48.0
Avg. @ Q[2]	0.6	0.9	1.3	1.8	1.9
Max. @ Q[2]	67.0	N/A	N/A	N/A	67.0
Avg. @ Q[3]	0.1	0.1	0.1	0.09	0.09
Max. @ Q[3]	49.0	N/A	N/A	N/A	49.0
Avg. @ Q[4]	0.08	0.1	0.1	0.09	0.9
Max. @ Q[4]	49.0	N/A	N/A	N/A	49.0

Table A.9: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.1)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	0.002	0.003	0.003	0.003	0.004
Max. @ Q[0]	1.0	N/A	N/A	N/A	1.0
Avg. @ Q[1]	0.3	0.3	0.3	0.3	0.3
Max. @ Q[1]	48.0	N/A	N/A	N/A	48.0
Avg. @ Q[2]	3.4	3.8	4.8	5.0	3.7
Max. @ Q[2]	67.0	N/A	N/A	N/A	58.0
Avg. @ Q[3]	0.1	0.1	0.1	0.1	0.1
Max. @ Q[3]	49.0	N/A	N/A	N/A	49.0
Avg. @ Q[4]	0.08	0.1	0.1	0.09	0.09
Max. @ Q[4]	49.0	N/A	N/A	N/A	49.0

Table A.10: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.5)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	0.002	0.003	0.003	0.003	0.004
Max. @ Q[0]	1.0	N/A	N/A	N/A	1.0
Avg. @ Q[1]	0.3	0.3	0.3	0.3	0.2
Max. @ Q[1]	48.0	N/A	N/A	N/A	48.0
Avg. @ Q[2]	4.8	5.2	7.0	5.6	3.4
Max. @ Q[2]	66.0	N/A	N/A	N/A	52.0
Avg. @ Q[3]	0.1	0.1	0.1	0.1	0.1
Max. @ Q[3]	49.0	N/A	N/A	N/A	49.0
Avg. @ Q[4]	0.09	0.1	0.1	0.1	0.08
Max. @ Q[4]	49.0	N/A	N/A	N/A	49.0

Table A.11: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.8)

	Burstiness Frequency Level (sec.)				
Queue Size (# IEEE 1355 pkts)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
Avg. @ Q[0]	0.002	0.003	0.003	0.003	0.004
Max. @ Q[0]	1.0	N/A	N/A	N/A	1.0
Avg. @ Q[1]	0.3	0.3	0.3	0.3	0.2
Max. @ Q[1]	48.0	N/A	N/A	N/A	48.0
Avg. @ Q[2]	5.0	6.2	8.0	6.2	5.9
Max. @ Q[2]	67.0	N/A	N/A	N/A	67.0
Avg. @ Q[3]	0.1	0.1	0.1	0.09	0.09
Max. @ Q[3]	49.0	N/A	N/A	N/A	49.0
Avg. @ Q[4]	0.08	0.1	0.1	0.09	0.09
Max. @ Q[4]	49.0	N/A	N/A	N/A	49.0

Table A.12: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 1.0)

	Burstiness Frequency Level (sec.)				
End-to-end Packet Delay (msec.)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
When T9000 Proc. Speed Level = 0.1	0.3	0.3	0.3	0.3	0.3
When T9000 Proc. Speed Level = 0.5	0.8	1.0	0.9	0.9	0.9
When T9000 Proc. Speed Level = 0.8	4.9	3.7	2.3	1.8	1.5
When T9000 Proc. Speed Level = 1.0	8.8	6.1	3.5	2.5	2.0

Table A.13: End-to-end Delay for Packets leaving the ATM Port towards any Ethernet Port

	Burstiness Frequency Level (sec.)				
End-to-end Packet Delay (msec.)	*7.5*	*15.0*	*30.0*	*60.0*	*120.0*
When T9000 Proc. Speed Level = 0.1	0.1	0.1	0.1	0.1	0.07
When T9000 Proc. Speed Level = 0.5	3.0	1.7	0.9	0.5	0.3
When T9000 Proc. Speed Level = 0.8	4.9	2.6	1.3	0.7	0.3
When T9000 Proc. Speed Level = 1.0	7.0	4.1	2.2	1.2	0.6

Table A.14: End-to-end Delay for Packets leaving any Ethernet Port towards any other Ethernet or the ATM Port

A.1.2 With the 4-by-4 Model (Destns. Randomly Distributed)

	T9000 Proc. Speed Level		
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*
Avg. @ Q[0]	0.08	4.0	9.2
Max. @ Q[0]	1.0	22.0	25.0
Avg. @ Q[1]	75.4	345.8	645.0
Max. @ Q[1]	1182.0	1294.0	1207.0

Table A.15: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3

	T9000 Proc. Speed Level		
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*
Avg. @ Q[3]	0.9	4.8	17.7
Max. @ Q[3]	49.0	147.0	294.0
Avg. @ Q[4]	0.5	2.4	12.0
Max. @ Q[4]	49.0	147.0	294.0
Avg. @ Q[5]	0.3	1.8	8.1
Max. @ Q[5]	49.0	147.0	294.0

Table A.16: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3

	T9000 Proc. Speed Level		
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*
Avg. @ Q[0]	0.003	0.003	0.003
Max. @ Q[0]	1.0	1.0	1.0
Avg. @ Q[1]	0.2	0.2	0.2
Max. @ Q[1]	48.0	48.0	48.0
Avg. @ Q[2]	1.1	4.5	6.4
Max. @ Q[2]	67.0	67.0	67.0
Avg. @ Q[4]	0.2	0.2	0.2
Max. @ Q[4]	49.0	49.0	49.0
Avg. @ Q[5]	0.2	0.2	0.2
Max. @ Q[5]	49.0	49.0	49.0

Table A.17: Queue Occupancy in T9000 (@ Ethernet Port 1): traffic to/from ATM and Ethernet Port 2 & 3

	T9000 Proc. Speed Level		
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*
Avg. @ Q[0]	0.003	0.003	0.003
Max. @ Q[0]	1.0	1.0	1.0
Avg. @ Q[1]	0.3	0.4	0.3
Max. @ Q[1]	48.0	48.0	48.0
Avg. @ Q[2]	1.9	6.0	10.2
Max. @ Q[2]	67.0	67.0	67.0
Avg. @ Q[3]	0.06	0.06	0.06
Max. @ Q[3]	49.0	49.0	49.0
Avg. @ Q[5]	0.09	0.1	0.1
Max. @ Q[5]	49.0	49.0	49.0

Table A.18: Queue Occupancy in T9000 (@ Ethernet Port 2): traffic to/from ATM and Ethernet Port 1 & 3

	T9000 Proc. Speed Level		
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*
Avg. @ Q[0]	0.003	0.003	0.003
Max. @ Q[0]	1.0	1.0	1.0
Avg. @ Q[1]	0.3	0.4	0.4
Max. @ Q[1]	48.0	48.0	48.0
Avg. @ Q[2]	1.1	3.7	6.0
Max. @ Q[2]	69.0	65.0	67.0
Avg. @ Q[3]	0.04	0.04	0.04
Max. @ Q[3]	49.0	49.0	49.0
Avg. @ Q[4]	0.09	0.09	0.09
Max. @ Q[4]	49.0	49.0	49.0

Table A.19: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2

	T9000 Proc. Speed Level		
End-to-end Packet Delay (msec.)	*0.1*	*0.5*	*0.8*
From ATM Port to any Ethernet Port	0.3	0.9	3.2
From any Ethernet Port to any other Ethernet or the ATM Port	0.1	0.9	2.6

Table A.20: End-to-end Delay for Packets

A.2 Outcomes under the SRD Traffic (MMPP)

A.2.1 With the 4-by-4 Model

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.3	0.5	0.6	0.6
Max. @ Q[0]	6.0	10.0	11.0	11.0
Avg. @ Q[1]	5.5	11.0	19.7	42.4
Max. @ Q[1]	28.0	49.0	212.0	574.0

Table A.21: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	1.3	1.7	1.7	1.8
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	1.2	1.1	1.2	1.2
Max. @ Q[4]	343.0	441.0	441.0	430.0
Avg. @ Q[5]	1.2	0.9	0.9	0.9
Max. @ Q[5]	343.0	392.0	405.0	441.0

Table A.22: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	1.4	2.5	1.0	1.5
Max. @ Q[0]	6.0	11.0	14.0	17.0
Avg. @ Q[1]	9.7	20.5	49.4	155.0
Max. @ Q[1]	37.0	114.0	295.0	877.0

Table A.23: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	6.6	8.4	8.9	8.9
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	6.3	5.9	6.2	6.2
Max. @ Q[4]	343.0	441.0	441.0	441.0
Avg. @ Q[5]	6.0	4.4	4.6	4.8
Max. @ Q[5]	343.0	392.0	392.0	441.0

Table A.24: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	2.3	3.4	3.4	3.9
Max. @ Q[0]	7.0	19.0	25.0	24.0
Avg. @ Q[1]	23.6	67.2	93.8	499.6
Max. @ Q[1]	287.0	496.0	905.0	1480.0

Table A.25: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	10.8	14.8	16.0	16.4
Max. @ Q[3]	294.0	539.0	539.0	588.0
Avg. @ Q[4]	10.5	10.0	10.9	11.0
Max. @ Q[4]	343.0	441.0	441.0	441.0
Avg. @ Q[5]	10.0	7.6	8.0	8.1
Max. @ Q[5]	343.0	392.0	392.0	441.0

Table A.26: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	4.0	6.2	8.1	8.4
Max. @ Q[0]	12.0	24.0	36.0	41.0
Avg. @ Q[1]	64.6	124.7	202.7	1491.7
Max. @ Q[1]	330.0	845.0	2068.0	3655.0

Table A.27: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	14.1	20.4	22.1	22.9
Max. @ Q[3]	294.0	588.0	588.0	637.0
Avg. @ Q[4]	13.6	13.8	15.3	15.9
Max. @ Q[4]	343.0	441.0	441.0	441.0
Avg. @ Q[5]	12.8	10.0	10.8	11.2
Max. @ Q[5]	343.0	392.0	441.0	441.0

Table A.28: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 3 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.003	0.002	0.002	0.002
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	1.0	1.1	1.1	1.4
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	3.2	3.7	3.9	3.9
Max. @ Q[2]	59.0	67.0	66.0	67.0
Avg. @ Q[3]	0.1	0.2	0.2	0.2
Max. @ Q[3]	67.0	67.0	67.0	67.0
Avg. @ Q[4]	0.1	0.1	0.1	0.1
Max. @ Q[4]	67.0	67.0	67.0	67.0

Table A.29: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.003	0.002	0.002	0.002
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	1.1	1.1	1.4	1.6
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	3.8	4.4	4.7	5.3
Max. @ Q[2]	49.0	67.0	67.0	67.0
Avg. @ Q[3]	0.1	0.2	0.2	0.2
Max. @ Q[3]	66.0	67.0	67.0	67.0
Avg. @ Q[4]	0.1	0.1	0.1	0.1
Max. @ Q[4]	67.0	67.0	67.0	67.0

Table A.30: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.5)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.003	0.002	0.002	0.002
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	1.5	1.4	1.6	1.9
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	5.1	7.1	8.0	5.8
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	5.1	0.2	0.2	0.2
Max. @ Q[3]	67.0	67.0	67.0	67.0
Avg. @ Q[4]	0.1	0.2	0.1	0.1
Max. @ Q[4]	66.0	67.0	67.0	67.0

Table A.31: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 0.8)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.003	0.002	0.002	0.002
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	2.0	1.7	1.7	2.8
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	8.2	9.2	9.7	6.8
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.2	0.3	0.3	0.3
Max. @ Q[3]	66.0	67.0	67.0	67.0
Avg. @ Q[4]	0.1	0.2	0.2	0.2
Max. @ Q[4]	67.0	67.0	67.0	67.0

Table A.32: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2 (T9000 Proc. Speed Level = 1.0)

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.4	0.5	0.5	0.5
When T9000 Proc. Speed Level = 0.5	1.4	1.6	1.7	1.8
When T9000 Proc. Speed Level = 0.8	2.2	2.6	2.9	3.0
When T9000 Proc. Speed Level = 1.0	2.7	3.6	3.9	4.0

Table A.33: End-to-end Delay for Packets leaving the ATM Port towards any Ethernet Port

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.1	0.2	0.1	0.2
When T9000 Proc. Speed Level = 0.5	0.2	0.3	0.3	0.5
When T9000 Proc. Speed Level = 0.8	0.5	0.6	0.8	1.3
When T9000 Proc. Speed Level = 1.0	1.0	1.8	0.8	3.4

Table A.34: End-to-end Delay for Packets leaving any Ethernet Port towards any other Ethernet or the ATM Port

A.2.2 With the 8-by-8 Model

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.6	0.7	0.8	0.8
Max. @ Q[0]	13.0	23.0	25.0	25.0
Avg. @ Q[1]	107.2	457.5	531.8	516.5
Max. @ Q[1]	424.0	6637.0	8365.0	9725.0

Table A.35: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.1)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	0.5	0.6	0.7	0.7
Max. @ Q[3]	294.0	343.0	392.0	392.0
Avg. @ Q[4]	0.6	0.7	0.8	0.8
Max. @ Q[4]	441.0	441.0	490.0	490.0
Avg. @ Q[5]	0.5	0.6	0.7	0.7
Max. @ Q[5]	245.0	343.0	392.0	392.0
Avg. @ Q[6]	0.6	0.8	0.9	0.9
Max. @ Q[6]	441.0	441.0	490.0	490.0
Avg. @ Q[7]	0.6	0.8	0.9	0.9
Max. @ Q[7]	294.0	372.0	392.0	392.0
Avg. @ Q[8]	0.4	0.5	0.6	0.6
Max. @ Q[8]	294.0	343.0	392.0	392.0
Avg. @ Q[9]	0.5	0.6	0.7	0.7
Max. @ Q[9]	294.0	343.0	392.0	392.0

Table A.36: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.1)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	3.0	3.5	4.0	4.3
Max. @ Q[0]	13.0	23.0	25.0	29.0
Avg. @ Q[1]	1001.8	1720.0	2338.0	5935.5
Max. @ Q[1]	6448.0	19596.0	23234.0	55126.0

Table A.37: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	2.8	3.6	4.0	4.0
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	3.2	4.4	4.9	5.0
Max. @ Q[4]	490.0	490.0	539.0	539.0
Avg. @ Q[5]	2.6	3.7	4.0	4.1
Max. @ Q[5]	294.0	392.0	441.0	441.0
Avg. @ Q[6]	3.2	4.9	5.5	5.6
Max. @ Q[6]	490.0	490.0	539.0	539.0
Avg. @ Q[7]	3.3	4.9	5.5	5.6
Max. @ Q[7]	294.0	441.0	441.0	441.0
Avg. @ Q[8]	2.5	3.0	3.3	3.4
Max. @ Q[8]	343.0	441.0	490.0	490.0
Avg. @ Q[9]	2.9	3.8	4.2	4.4
Max. @ Q[9]	294.0	392.0	441.0	441.0

Table A.38: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	4.8	125.5	170.4	181.85
Max. @ Q[0]	13.0	860.0	1077.0	1132.0
Avg. @ Q[1]	2539.2	28210.5	19185.3	24857.2
Max. @ Q[1]	14370.0	144212.0	92025.0	148852.0

Table A.39: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	4.5	5.9	6.4	6.6
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	5.2	7.1	8.0	8.2
Max. @ Q[4]	490.0	490.0	539.0	539.0
Avg. @ Q[5]	4.2	6.0	6.5	6.7
Max. @ Q[5]	294.0	392.0	441.0	441.0
Avg. @ Q[6]	5.3	8.1	9.0	9.3
Max. @ Q[6]	490.0	588.0	686.0	686.0
Avg. @ Q[7]	5.3	7.9	9.0	9.2
Max. @ Q[7]	294.0	441.0	441.0	441.0
Avg. @ Q[8]	4.1	5.0	5.4	5.5
Max. @ Q[8]	343.0	441.0	490.0	490.0
Avg. @ Q[9]	4.6	6.1	6.89	7.2
Max. @ Q[9]	294.0	441.0	490.0	539.0

Table A.40: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	7.2	322.8	380.6	395.8
Max. @ Q[0]	20.0	1258.0	1475.0	1530.0
Avg. @ Q[1]	66881.0	66251.0	53543.2	42111.3
Max. @ Q[1]	201714.0	230387.0	201652.0	201188.0

Table A.41: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	5.6	7.4	8.1	8.4
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	6.5	9.0	10.2	10.4
Max. @ Q[4]	490.0	490.0	539.0	539.0
Avg. @ Q[5]	5.3	7.6	8.4	8.6
Max. @ Q[5]	294.0	392.0	441.0	441.0
Avg. @ Q[6]	6.7	10.4	11.7	12.0
Max. @ Q[6]	539.0	637.0	735.0	735.0
Avg. @ Q[7]	6.7	10.0	11.6	11.9
Max. @ Q[7]	294.0	441.0	441.0	441.0
Avg. @ Q[8]	5.2	6.3	6.9	7.0
Max. @ Q[8]	343.0	441.0	490.0	490.0
Avg. @ Q[9]	5.8	7.8	8.8	9.2
Max. @ Q[9]	294.0	588.0	637.0	686.0

Table A.42: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.005	0.005
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.0	1.3	1.3
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.3	1.4	1.3	1.3
Max. @ Q[2]	49.0	67.0	67.0	67.0
Avg. @ Q[3]	0.03	0.03	0.03	0.04
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.04	0.06	0.06	0.06
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.04	0.05	0.05	0.05
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.05	0.06	0.07	0.07
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.04	0.05	0.05	0.05
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.03	0.03	0.04	0.04
Max. @ Q[9]	67.0	66.0	66.0	66.0

Table A.43: Queue Occupancy in T9000 (@ Ethernet Port 4): traffic to/from ATM and Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.005	0.005
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.2	1.44	1.5
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	3.3	2.0	1.4	1.7
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.03	0.04	0.04	0.05
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.04	0.06	0.06	0.07
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.04	0.05	0.05	0.05
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.06	0.07	0.08	0.08
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.05	0.04	0.05	0.05
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.04	0.05	0.04	0.05
Max. @ Q[9]	66.0	66.0	66.0	66.0

Table A.44: Queue Occupancy in T9000 (@ Ethernet Port 4): traffic to/from ATM and Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.005	0.005
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.3	1.5	1.5
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.5	2.3	1.5	1.7
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.04	0.04	0.04	0.04
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.05	0.05	0.06	0.07
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.04	0.05	0.04	0.05
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.06	0.08	0.09	0.09
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.04	0.04	0.04	0.05
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.03	0.05	0.05	0.06
Max. @ Q[9]	66.0	66.0	66.0	66.0

Table A.45: Queue Occupancy in T9000 (@ Ethernet Port 4) traffic to/from ATM and Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.005	0.005
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.3	1.4	1.5
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	3.6	3.4	2.9	3.7
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.04	0.04	0.04	0.04
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.05	0.06	0.06	0.06
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.04	0.05	0.05	0.05
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.05	0.08	0.09	0.09
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.04	0.04	0.05	0.05
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.04	0.05	0.05	0.06
Max. @ Q[9]	66.0	66.0	66.0	66.0

Table A.46: Queue Occupancy in T9000 (@ Ethernet Port 4): traffic to/from ATM and Ethernet Port 1 thru 7 (T9000 Proc. Speed Level = 1.0)

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.2	0.2	0.2	0.2
When T9000 Proc. Speed Level = 0.5	0.7	0.8	0.9	0.9
When T9000 Proc. Speed Level = 0.8	1.1	1.3	1.4	1.4
When T9000 Proc. Speed Level = 1.0	1.4	1.7	1.8	1.8

Table A.47: End-to-end Delay for Packets leaving the ATM Port towards any Ethernet Port

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.08	0.3	0.3	0.3
When T9000 Proc. Speed Level = 0.5	0.6	1.0	1.4	3.3
When T9000 Proc. Speed Level = 0.8	1.4	14.7	18.0	18.4
When T9000 Proc. Speed Level = 1.0	27.0	32.8	38.5	43.2

Table A.48: End-to-end Delay for Packets leaving any Ethernet Port towards any other Ethernet or the ATM Port

A.2.3 With the 12-by-12 Model

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.9	1.7	1.9	2.0
Max. @ Q[0]	20.0	35.0	37.0	38.0
Avg. @ Q[1]	658.2	1227.6	1496.9	1625.8
Max. @ Q[1]	3761.0	11427.0	24949.0	16528.0

Table A.49: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	0.2	0.3	0.3	0.4
Max. @ Q[3]	245.0	343.0	343.0	343.0
Avg. @ Q[4]	0.2	0.3	0.3	0.34
Max. @ Q[4]	245.0	294.0	343.0	294.0
Avg. @ Q[5]	0.2	0.2	0.2	0.2
Max. @ Q[5]	245.0	343.0	340.0	303.0
Avg. @ Q[6]	0.2	0.3	0.3	0.3
Max. @ Q[6]	203.0	294.0	294.0	294.0
Avg. @ Q[7]	0.2	0.3	0.28	0.28
Max. @ Q[7]	196.0	294.0	343.0	343.0
Avg. @ Q[8]	0.2	0.2	0.3	0.3
Max. @ Q[8]	263.0	312.0	313.0	294.0
Avg. @ Q[9]	0.2	0.2	0.3	0.3
Max. @ Q[9]	245.0	343.0	343.0	343.0
Avg. @ Q[10]	0.2	0.3	0.3	0.3
Max. @ Q[10]	245.0	294.0	294.0	294.0
Avg. @ Q[11]	0.2	0.3	0.4	0.4
Max. @ Q[11]	245.0	343.0	343.0	343.0
Avg. @ Q[12]	0.2	0.2	0.2	0.2
Max. @ Q[12]	366.0	343.0	392.0	392.0
Avg. @ Q[13]	0.2	0.3	0.3	0.3
Max. @ Q[13]	294.0	343.0	392.0	392.0

Table A.50: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	4.6	8.2	9.4	10.0
Max. @ Q[0]	20.0	35.0	39.0	44.0
Avg. @ Q[1]	3017.7	4459.2	5267.8	5779.2
Max. @ Q[1]	12850.0	52127.0	60879.0	43621.0

Table A.51: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	1.3	2.0	2.3	2.4
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	1.2	2.0	2.1	2.2
Max. @ Q[4]	294.0	392.0	441.0	441.0
Avg. @ Q[5]	1.2	1.4	1.5	1.5
Max. @ Q[5]	343.0	490.0	490.0	490.0
Avg. @ Q[6]	1.2	1.6	1.8	1.8
Max. @ Q[6]	245.0	343.0	392.0	392.0
Avg. @ Q[7]	1.3	1.7	1.8	1.9
Max. @ Q[7]	245.0	343.0	392.0	392.0
Avg. @ Q[8]	1.4	1.6	1.8	1.8
Max. @ Q[8]	343.0	441.0	441.0	441.0
Avg. @ Q[9]	1.2	1.6	1.7	1.7
Max. @ Q[9]	294.0	441.0	441.0	441.0
Avg. @ Q[10]	1.3	1.7	1.9	1.9
Max. @ Q[10]	294.0	392.0	441.0	441.0
Avg. @ Q[11]	1.5	2.2	2.4	2.5
Max. @ Q[11]	294.0	441.0	441.0	441.0
Avg. @ Q[12]	1.3	1.3	1.5	1.5
Max. @ Q[12]	490.0	441.0	490.0	490.0
Avg. @ Q[13]	1.3	1.7	1.8	1.86
Max. @ Q[13]	343.0	441.0	490.0	490.0

Table A.52: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	7.3	446.9	631.4	680.9
Max. @ Q[0]	20.0	2024.0	2667.0	2835.0
Avg. @ Q[1]	5597.6	211179.0	67631.4	54342.0
Max. @ Q[1]	30998.0	365312.0	228930.0	235216.0

Table A.53: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	2.2	3.4	3.8	4.0
Max. @ Q[3]	343.0	392.0	441.0	441.0
Avg. @ Q[4]	2.0	3.2	3.5	3.7
Max. @ Q[4]	294.0	392.0	441.0	441.0
Avg. @ Q[5]	1.9	2.3	2.5	2.5
Max. @ Q[5]	343.0	490.0	490.0	490.0
Avg. @ Q[6]	2.0	2.7	3.0	3.0
Max. @ Q[6]	245.0	343.0	392.0	392.0
Avg. @ Q[7]	2.2	2.7	3.0	3.0
Max. @ Q[7]	245.0	343.0	392.0	392.0
Avg. @ Q[8]	2.3	2.6	3.0	3.0
Max. @ Q[8]	343.0	441.0	441.0	441.0
Avg. @ Q[9]	2.0	2.6	2.8	2.8
Max. @ Q[9]	294.0	441.0	441.0	441.0
Avg. @ Q[10]	2.1	2.7	3.0	3.2
Max. @ Q[10]	294.0	392.0	441.0	441.0
Avg. @ Q[11]	2.4	3.6	3.9	4.0
Max. @ Q[11]	294.0	441.0	441.0	441.0
Avg. @ Q[12]	2.2	2.2	2.4	2.4
Max. @ Q[12]	490.0	441.0	490.0	490.0
Avg. @ Q[13]	2.0	2.7	3.0	3.0
Max. @ Q[13]	343.0	441.0	490.0	490.0

Table A.54: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	10.5	1640.8	1944.9	2019.6
Max. @ Q[0]	29.0	3817.0	4500.0	4668.0
Avg. @ Q[1]	304942.8	89829.4	271501.7	213597.4
Max. @ Q[1]	573624.0	239978.0	560687.0	475928.0

Table A.55: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	2.7	4.2	4.8	5.0
Max. @ Q[3]	343.0	392.0	441.0	441.0
Avg. @ Q[4]	2.5	4.0	4.5	4.6
Max. @ Q[4]	294.0	392.0	441.0	441.0
Avg. @ Q[5]	2.4	2.8	3.1	3.1
Max. @ Q[5]	343.0	490.0	490.0	490.0
Avg. @ Q[6]	2.5	3.3	3.7	3.8
Max. @ Q[6]	245.0	343.0	392.0	392.0
Avg. @ Q[7]	2.7	3.4	3.8	3.8
Max. @ Q[7]	245.0	343.0	392.0	392.0
Avg. @ Q[8]	2.8	3.3	3.8	3.8
Max. @ Q[8]	343.0	441.0	441.0	441.0
Avg. @ Q[9]	2.5	3.0	3.5	3.6
Max. @ Q[9]	294.0	441.0	441.0	441.0
Avg. @ Q[10]	2.7	3.3	3.9	4.0
Max. @ Q[10]	294.0	343.0	441.0	441.0
Avg. @ Q[11]	3.0	4.4	5.0	5.1
Max. @ Q[11]	294.0	441.0	441.0	441.0
Avg. @ Q[12]	2.7	2.8	3.0	3.0
Max. @ Q[12]	490.0	441.0	490.0	490.0
Avg. @ Q[13]	2.6	3.4	3.7	3.8
Max. @ Q[13]	343.0	441.0	490.0	490.0

Table A.56: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.7	0.9	1.0	1.2
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.0	1.3	1.5	1.4
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.03	0.03	0.04	0.04
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.03	0.02	0.02	0.02
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.03	0.03
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.03	0.03	0.03
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.03	0.03	0.03	0.02
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.04	0.02	0.03	0.03
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[10]	0.04	0.04	0.03	0.04
Max. @ Q[10]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.04
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.03	0.03	0.03	0.04
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.03	0.04	0.03	0.03
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.57: Queue Occupancy in T9000 (@ Ethernet Port 6): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.7	1.0	1.2	1.1
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.2	2.3	1.3	1.9
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.03	0.04	0.04	0.05
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.03
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.03	0.03
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.03	0.03	0.03
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.03	0.03	0.02	0.03
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.03	0.03	0.04	0.04
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[10]	0.04	0.04	0.05	0.04
Max. @ Q[10]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.04	0.04	0.04	0.04
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.03	0.03	0.04	0.04
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.03	0.03	0.04	0.04
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.58: Queue Occupancy in T9000 (@ Ethernet Port 6): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.0	1.1	1.0
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.9	1.7	1.7	2.2
Max. @ Q[2]	67.0	67.0	67.0	71.0
Avg. @ Q[3]	0.03	0.04	0.04	0.04
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.03
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.03	0.03
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.03	0.03	0.04
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.03	0.02	0.03	0.03
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.03	0.03	0.04	0.04
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[10]	0.04	0.04	0.04	0.04
Max. @ Q[10]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.03
Max. @ Q[11]	66.0	67.0	66.0	66.0
Avg. @ Q[12]	0.03	0.04	0.03	0.03
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.03	0.03	0.04	0.04
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.59: Queue Occupancy in T9000 (@ Ethernet Port 6): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.6	1.2	1.2
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	2.4	2.7	3.0	3.0
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.03	0.06	0.05	0.05
Max. @ Q[3]	66.0	67.0	66.0	66.0
Avg. @ Q[4]	0.02	0.05	0.02	0.02
Max. @ Q[4]	66.0	67.0	66.0	66.0
Avg. @ Q[5]	0.02	0.05	0.03	0.03
Max. @ Q[5]	66.0	67.0	66.0	66.0
Avg. @ Q[6]	0.03	0.04	0.03	0.04
Max. @ Q[6]	66.0	67.0	66.0	66.0
Avg. @ Q[7]	0.03	0.05	0.04	0.03
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.03	0.05	0.04	0.04
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[10]	0.04	0.04	0.04	0.04
Max. @ Q[10]	66.0	67.0	66.0	66.0
Avg. @ Q[11]	0.03	0.09	0.04	0.04
Max. @ Q[11]	66.0	67.0	66.0	66.0
Avg. @ Q[12]	0.03	0.05	0.03	0.04
Max. @ Q[12]	66.0	67.0	66.0	66.0
Avg. @ Q[13]	0.03	0.05	0.04	0.04
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.60: Queue Occupancy in T9000 (@ Ethernet Port 6): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 1.0)

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.1	0.1	0.1	0.1
When T9000 Proc. Speed Level = 0.5	0.5	0.6	0.6	0.6
When T9000 Proc. Speed Level = 0.8	0.8	0.9	0.9	1.0
When T9000 Proc. Speed Level = 1.0	1.0	1.1	1.2	1.2

Table A.61: End-to-end Delay for Packets leaving the ATM Port towards any Ethernet Port

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.2	0.4	0.4	0.5
When T9000 Proc. Speed Level = 0.5	1.0	1.3	1.5	1.6
When T9000 Proc. Speed Level = 0.8	1.8	23.3	25.8	61.2
When T9000 Proc. Speed Level = 1.0	44.4	75.0	76.5	83.7

Table A.62: End-to-end Delay for Packets leaving any Ethernet Port towards any other Ethernet or the ATM Port

A.2.4 With the 16-by-16 Model

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	1.3	1.9	2.2	2.2
Max. @ Q[0]	27.0	47.0	50.0	52.0
Avg. @ Q[1]	1058.1	1556.7	1379.5	1729.8
Max. @ Q[1]	7775.0	24086.0	18966.0	33474.0

Table A.63: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 15 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	0.1	0.1	0.1	0.1
Max. @ Q[3]	245.0	294.0	294.0	310.0
Avg. @ Q[4]	0.1	0.1	0.1	0.1
Max. @ Q[4]	230.0	248.0	245.0	245.0
Avg. @ Q[5]	0.09	0.1	0.1	0.1
Max. @ Q[5]	196.0	245.0	265.0	284.0
Avg. @ Q[6]	0.1	0.2	0.2	0.2
Max. @ Q[6]	245.0	248.0	280.0	271.0
Avg. @ Q[7]	0.09	0.1	0.1	0.1
Max. @ Q[7]	194.0	196.0	245.0	245.0
Avg. @ Q[8]	0.1	0.1	0.1	0.1
Max. @ Q[8]	196.0	245.0	245.0	245.0
Avg. @ Q[9]	0.09	0.09	0.1	0.1
Max. @ Q[9]	196.0	245.0	245.0	294.0
Avg. @ Q[10]	0.1	0.1	0.2	0.2
Max. @ Q[10]	294.0	294.0	294.0	294.0
Avg. @ Q[11]	0.1	0.1	0.2	0.2
Max. @ Q[11]	343.0	343.0	392.0	392.0
Avg. @ Q[12]	0.09	0.1	0.1	0.1
Max. @ Q[12]	181.0	196.0	203.0	196.0
Avg. @ Q[13]	0.1	0.1	0.2	0.2
Max. @ Q[13]	245.0	300.0	338.0	343.0

Table A.64: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.1	0.1	0.1	0.1
Max. @ Q[14]	245.0	245.0	294.0	294.0
Avg. @ Q[15]	0.1	0.1	0.1	0.1
Max. @ Q[15]	201.0	261.0	245.0	245.0
Avg. @ Q[16]	0.09	0.09	0.1	0.1
Max. @ Q[16]	196.0	245.0	235.0	236.0
Avg. @ Q[17]	0.1	0.1	0.1	0.1
Max. @ Q[17]	196.0	294.0	294.0	294.0

Table A.65: Queue Occupancy in T9000 #2: traffic from Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	6.1	9.3	10.7	11.3
Max. @ Q[0]	27.0	47.0	51.0	55.0
Avg. @ Q[1]	4057.7	7187.8	6074.0	5752.6
Max. @ Q[1]	25881.0	54685.0	55669.0	43076.0

Table A.66: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 15 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	0.8	0.8	0.9	0.9
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	0.7	0.9	0.9	1.0
Max. @ Q[4]	294.0	428.0	441.0	426.0
Avg. @ Q[5]	0.7	1.0	1.1	1.1
Max. @ Q[5]	245.0	343.0	343.0	392.0
Avg. @ Q[6]	0.8	1.1	1.2	1.2
Max. @ Q[6]	343.0	441.0	490.0	490.0
Avg. @ Q[7]	0.6	0.8	0.8	0.9
Max. @ Q[7]	294.0	294.0	343.0	343.0
Avg. @ Q[8]	0.7	0.9	0.9	1.0
Max. @ Q[8]	294.0	343.0	392.0	392.0
Avg. @ Q[9]	0.7	0.7	0.7	0.7
Max. @ Q[9]	294.0	343.0	343.0	343.0
Avg. @ Q[10]	0.8	1.0	1.2	1.2
Max. @ Q[10]	441.0	441.0	490.0	490.0
Avg. @ Q[11]	0.7	1.2	1.3	1.4
Max. @ Q[11]	490.0	490.0	539.0	539.0
Avg. @ Q[12]	0.7	0.8	0.8	0.8
Max. @ Q[12]	245.0	294.0	294.0	294.0
Avg. @ Q[13]	0.7	1.0	1.2	1.2
Max. @ Q[13]	294.0	392.0	441.0	441.0

Table A.67: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.5)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.7	0.8	1.0	1.0
Max. @ Q[14]	343.0	392.0	441.0	441.0
Avg. @ Q[15]	0.7	1.0	1.0	1.1
Max. @ Q[15]	294.0	441.0	441.0	441.0
Avg. @ Q[16]	0.6	0.7	0.8	0.8
Max. @ Q[16]	245.0	297.0	343.0	343.0
Avg. @ Q[17]	0.7	1.0	1.0	1.0
Max. @ Q[17]	245.0	392.0	392.0	392.0

Table A.68: Queue Occupancy in T9000 #2: traffic from Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.5)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	9.8	377.8	500.3	530.9
Max. @ Q[0]	27.0	1876.0	2354.0	2474.0
Avg. @ Q[1]	6644.7	13577.7	26337.1	48910.8
Max. @ Q[1]	24858.0	74321.0	141543.0	195890.0

Table A.69: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 15 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	1.3	1.3	1.4	1.5
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	1.2	1.4	1.5	1.6
Max. @ Q[4]	294.0	441.0	441.0	441.0
Avg. @ Q[5]	1.1	1.6	1.8	1.9
Max. @ Q[5]	245.0	343.0	343.0	392.0
Avg. @ Q[6]	1.2	1.8	2.0	2.0
Max. @ Q[6]	343.0	441.0	490.0	490.0
Avg. @ Q[7]	1.0	1.3	1.4	1.4
Max. @ Q[7]	294.0	343.0	343.0	343.0
Avg. @ Q[8]	1.1	1.4	1.5	1.6
Max. @ Q[8]	294.0	343.0	392.0	392.0
Avg. @ Q[9]	1.1	1.0	1.2	1.2
Max. @ Q[9]	245.0	343.0	343.0	343.0
Avg. @ Q[10]	1.4	1.8	2.0	2.0
Max. @ Q[10]	460.0	490.0	539.0	539.0
Avg. @ Q[11]	1.2	2.0	2.2	2.3
Max. @ Q[11]	490.0	490.0	539.0	539.0
Avg. @ Q[12]	1.0	1.3	1.4	1.4
Max. @ Q[12]	245.0	294.0	294.0	294.0
Avg. @ Q[13]	1.1	1.8	2.0	2.0
Max. @ Q[13]	294.0	392.0	441.0	441.0

Table A.70: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.8)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	1.2	1.4	1.6	1.6
Max. @ Q[14]	343.0	440.0	441.0	441.0
Avg. @ Q[15]	1.2	1.6	1.8	1.9
Max. @ Q[15]	294.0	441.0	441.0	441.0
Avg. @ Q[16]	1.0	1.2	1.4	1.4
Max. @ Q[16]	245.0	343.0	343.0	343.0
Avg. @ Q[17]	1.0	1.6	1.7	1.8
Max. @ Q[17]	245.0	392.0	392.0	392.0

Table A.71: Queue Occupancy in T9000 #2: traffic from Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.8)

Queue Size (# IEEE 1355 pkts)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	13.6	1184.2	1404.7	1458.2
Max. @ Q[0]	36.0	3550.0	4182.0	4339.0
Avg. @ Q[1]	50936.0	161836.0	187674.7	374593.1
Max. @ Q[1]	159642.0	273327.0	378466.0	748523.0

Table A.72: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 15 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[3]	1.6	1.7	1.8	1.9
Max. @ Q[3]	294.0	392.0	441.0	441.0
Avg. @ Q[4]	1.5	1.8	2.0	2.0
Max. @ Q[4]	294.0	441.0	441.0	441.0
Avg. @ Q[5]	1.4	2.0	2.3	2.4
Max. @ Q[5]	245.0	343.0	343.0	392.0
Avg. @ Q[6]	1.6	2.3	2.6	2.6
Max. @ Q[6]	343.0	441.0	490.0	490.0
Avg. @ Q[7]	1.3	1.6	1.7	1.8
Max. @ Q[7]	294.0	343.0	343.0	343.0
Avg. @ Q[8]	1.4	1.8	1.9	2.0
Max. @ Q[8]	294.0	343.0	392.0	392.0
Avg. @ Q[9]	1.4	1.4	1.5	1.6
Max. @ Q[9]	245.0	343.0	343.0	343.0
Avg. @ Q[10]	1.8	2.3	2.5	2.5
Max. @ Q[10]	490.0	490.0	539.0	539.0
Avg. @ Q[11]	1.6	2.5	2.8	2.9
Max. @ Q[11]	490.0	490.0	539.0	539.0
Avg. @ Q[12]	1.4	1.6	1.7	1.8
Max. @ Q[12]	245.0	294.0	294.0	294.0
Avg. @ Q[13]	1.4	2.2	2.4	2.5
Max. @ Q[13]	294.0	392.0	441.0	441.0

Table A.73: Queue Occupancy in T9000 #2: traffic from Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	1.5	1.8	2.0	2.0
Max. @ Q[14]	343.0	441.0	490.0	490.0
Avg. @ Q[15]	1.5	2.0	2.3	2.4
Max. @ Q[15]	294.0	441.0	441.0	441.0
Avg. @ Q[16]	1.2	1.5	1.7	1.8
Max. @ Q[16]	245.0	343.0	381.0	380.0
Avg. @ Q[17]	1.4	2.0	2.2	2.2
Max. @ Q[17]	245.0	392.0	392.0	392.0

Table A.74: Queue Occupancy in T9000 #2: traffic from Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.1	1.4	1.2
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	1.2	0.7	0.7	0.6
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.02	0.02	0.02	0.02
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.02
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.04	0.04
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.04	0.05	0.03
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.02	0.02	0.02	0.02
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.02	0.02	0.02	0.02
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.02	0.02	0.02	0.03
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.03
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.01	0.02	0.02	0.02
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.01	0.02	0.02	0.01
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.75: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.03	0.02	0.02	0.02
Max. @ Q[14]	66.0	66.0	66.0	66.0
Avg. @ Q[15]	0.02	0.02	0.02	0.02
Max. @ Q[15]	66.0	66.0	66.0	66.0
Avg. @ Q[16]	0.02	0.02	0.03	0.02
Max. @ Q[16]	66.0	66.0	66.0	66.0
Avg. @ Q[17]	0.02	0.02	0.02	0.02
Max. @ Q[17]	66.0	66.0	66.0	66.0

Table A.76: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.1)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.04	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.1	1.3	1.2
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	1.4	0.8	0.9	1.2
Max. @ Q[2]	67.0	67.0	67.0	67.0
Avg. @ Q[3]	0.02	0.02	0.02	0.02
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.02
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.03	0.04
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.02	0.04	0.04	0.04
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.02	0.02	0.03	0.02
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.02	0.02	0.02	0.02
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.02	0.02	0.03	0.03
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.03
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.02	0.02	0.02	0.02
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.01	0.01	0.02	0.01
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.77: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.02	0.02	0.02	0.03
Max. @ Q[14]	66.0	66.0	66.0	66.0
Avg. @ Q[15]	0.02	0.02	0.02	0.02
Max. @ Q[15]	66.0	66.0	66.0	66.0
Avg. @ Q[16]	0.02	0.02	0.03	0.03
Max. @ Q[16]	66.0	66.0	66.0	66.0
Avg. @ Q[17]	0.02	0.03	0.02	0.02
Max. @ Q[17]	66.0	66.0	66.0	66.0

Table A.78: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.5)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.7	1.1	1.2	1.3
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	1.9	1.7	1.1	1.3
Max. @ Q[2]	67.0	72.0	70.0	69.0
Avg. @ Q[3]	0.01	0.02	0.02	0.02
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.02
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.03	0.04	0.03
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.03	0.03	0.04
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.02	0.02	0.02	0.03
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.02	0.02	0.02	0.02
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.02	0.02	0.03	0.02
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.04
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.02	0.02	0.02	0.02
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.01	0.01	0.01	0.02
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.79: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.02	0.02	0.02	0.02
Max. @ Q[14]	66.0	66.0	66.0	66.0
Avg. @ Q[15]	0.02	0.02	0.02	0.02
Max. @ Q[15]	66.0	66.0	66.0	66.0
Avg. @ Q[16]	0.02	0.03	0.03	0.02
Max. @ Q[16]	66.0	66.0	66.0	66.0
Avg. @ Q[17]	0.02	0.02	0.02	0.03
Max. @ Q[17]	66.0	66.0	66.0	66.0

Table A.80: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 0.8)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[0]	0.004	0.004	0.004	0.004
Max. @ Q[0]	10.0	10.0	11.0	11.0
Avg. @ Q[1]	0.8	1.1	1.2	1.3
Max. @ Q[1]	489.0	489.0	538.0	538.0
Avg. @ Q[2]	1.8	1.8	1.8	1.7
Max. @ Q[2]	67.0	67.0	67.0	69.0
Avg. @ Q[3]	0.02	0.02	0.02	0.02
Max. @ Q[3]	66.0	66.0	66.0	66.0
Avg. @ Q[4]	0.02	0.02	0.02	0.02
Max. @ Q[4]	66.0	66.0	66.0	66.0
Avg. @ Q[5]	0.03	0.04	0.04	0.04
Max. @ Q[5]	66.0	66.0	66.0	66.0
Avg. @ Q[6]	0.03	0.02	0.03	0.03
Max. @ Q[6]	66.0	66.0	66.0	66.0
Avg. @ Q[7]	0.02	0.02	0.03	0.03
Max. @ Q[7]	66.0	66.0	66.0	66.0
Avg. @ Q[8]	0.02	0.02	0.02	0.02
Max. @ Q[8]	66.0	66.0	66.0	66.0
Avg. @ Q[9]	0.02	0.02	0.03	0.03
Max. @ Q[9]	66.0	66.0	66.0	66.0
Avg. @ Q[11]	0.03	0.04	0.04	0.04
Max. @ Q[11]	66.0	66.0	66.0	66.0
Avg. @ Q[12]	0.02	0.02	0.02	0.02
Max. @ Q[12]	66.0	66.0	66.0	66.0
Avg. @ Q[13]	0.01	0.01	0.01	0.02
Max. @ Q[13]	66.0	66.0	66.0	66.0

Table A.81: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 1 thru 11 (T9000 Proc. Speed Level = 1.0)

	Burstiness Ratio Level			
Queue Size (# IEEE 1355 pkts)	*1:1*	*10:1*	*50:1*	*500:1*
Avg. @ Q[14]	0.02	0.02	0.02	0.02
Max. @ Q[14]	66.0	66.0	66.0	66.0
Avg. @ Q[15]	0.02	0.02	0.02	0.02
Max. @ Q[15]	66.0	66.0	66.0	66.0
Avg. @ Q[16]	0.02	0.03	0.03	0.03
Max. @ Q[16]	66.0	66.0	66.0	66.0
Avg. @ Q[17]	0.02	0.03	0.02	0.02
Max. @ Q[17]	66.0	66.0	66.0	66.0

Table A.82: Queue Occupancy in T9000 (@ Ethernet Port 8): traffic to/from ATM and Ethernet Port 12 thru 15 (T9000 Proc. Speed Level = 1.0)

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.1	0.1	0.1	0.1
When T9000 Proc. Speed Level = 0.5	0.4	0.4	0.5	0.5
When T9000 Proc. Speed Level = 0.8	0.6	0.7	0.7	0.7
When T9000 Proc. Speed Level = 1.0	0.7	0.8	0.9	0.9

Table A.83: End-to-end Delay for Packets leaving the ATM Port towards any Ethernet Port

End-to-end Packet Delay (msec.)	Burstiness Ratio Level			
	1:1	*10:1*	*50:1*	*500:1*
When T9000 Proc. Speed Level = 0.1	0.3	0.4	0.3	0.4
When T9000 Proc. Speed Level = 0.5	1.0	1.6	1.4	1.3
When T9000 Proc. Speed Level = 0.8	1.5	8.1	10.5	14.2
When T9000 Proc. Speed Level = 1.0	25.1	33.7	53.4	82.5

Table A.84: End-to-end Delay for Packets leaving any Ethernet Port towards any other Ethernet or the ATM Port

A.3 Outcomes under a VBR Traffic (Star Wars Movie)

	T9000 Proc. Speed Level			
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*	*1.0*
Avg. @ Q[0]	0.4	1.0	1.5	3.0
Avg. @ Q[1]	11.3	25.9	73.3	139.7

Table A.85: Queue Occupancy in T9000 #1: traffic destined to Ethernet Port 1 thru 3

	T9000 Proc. Speed Level			
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*	*1.0*
Avg. @ Q[3]	0.2	0.5	1.9	4.4
Avg. @ Q[4]	0.2	0.7	2.5	4.9
Avg. @ Q[5]	0.1	0.5	2.3	4.2

Table A.86: Queue Occupancy in T9000 #2: traffic coming from Ethernet Port 1 thru 3

	T9000 Proc. Speed Level			
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*	*1.0*
Avg. @ Q[0]	0.003	0.003	0.003	0.003
Avg. @ Q[1]	0.3	0.29	0.3	0.3
Avg. @ Q[2]	3.6	4.4	4.9	5.2
Avg. @ Q[4]	0.1	0.1	0.1	1.0
Avg. @ Q[5]	0.1	0.1	0.1	0.1

Table A.87: Queue Occupancy in T9000 (@ Ethernet Port 1): traffic to/from ATM and Ethernet Port 2 & 3

	T9000 Proc. Speed Level			
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*	*1.0*
Avg. @ Q[0]	0.003	0.003	0.003	0.003
Avg. @ Q[1]	0.3	0.3	0.3	0.29
Avg. @ Q[2]	2.9	3.5	4.9	3.9
Avg. @ Q[3]	0.09	0.09	0.1	0.094
Avg. @ Q[5]	0.1	0.1	0.1	0.13

Table A.88: Queue Occupancy in T9000 (@ Ethernet Port 2): traffic to/from ATM and Ethernet Port 1 & 3

	T9000 Proc. Speed Level			
Queue Size (# IEEE 1355 pkts)	*0.1*	*0.5*	*0.8*	*1.0*
Avg. @ Q[0]	0.003	0.003	0.003	0.003
Avg. @ Q[1]	0.3	0.3	0.3	0.3
Avg. @ Q[2]	3.5	4.2	4.6	4.8
Avg. @ Q[3]	0.1	0.1	0.1	0.1
Avg. @ Q[4]	0.1	0.1	0.1	0.1

Table A.89: Queue Occupancy in T9000 (@ Ethernet Port 3): traffic to/from ATM and Ethernet Port 1 & 2

	T9000 Proc. Speed Level			
End-to-end Packet Delay (msec.)	*0.1*	*0.5*	*0.8*	*1.0*
From ATM to any Ethernet Port	0.1	0.4	0.7	1.4
From any Ethernet to any other Ethernet or the ATM Port	0.6	0.5	0.7	0.9

Table A.90: End-to-end Delay for Packets

Appendix B

ALAX OPNET (Selected) Simulation Programs

This appendix contains selected components and modules of the ALAX simulation programs as implemented in OPNET. The complete set of programs can be made available upon request

B.1 ALAX Models

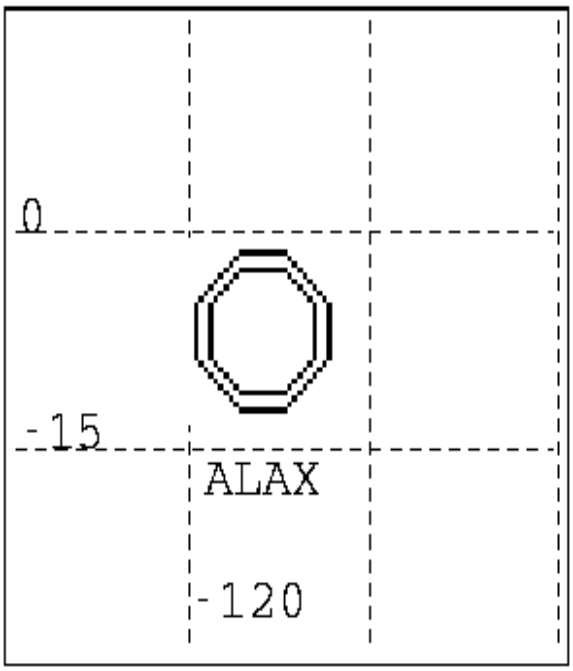

Figure B.1: ALAX Model at the Network Level in OPNET (for all configurations implemented)

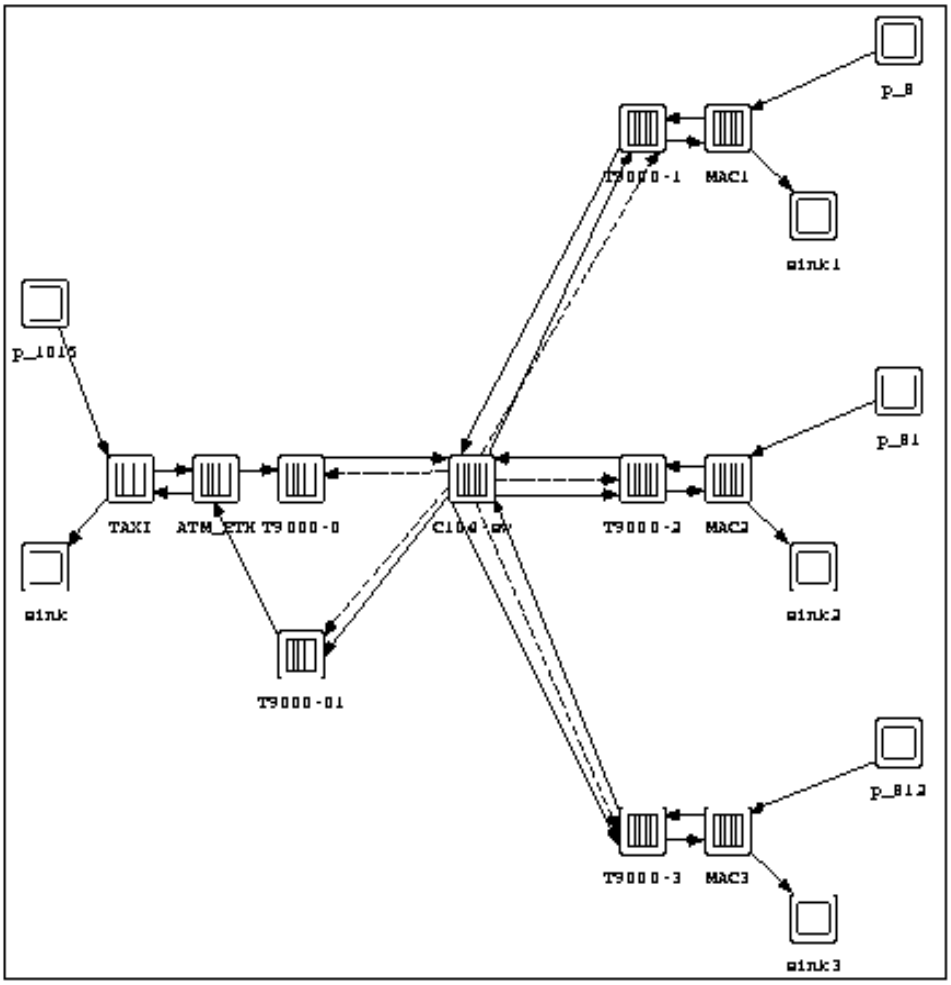

Figure B.2: ALAX Model at the Node Level in OPNET (4-port configuration)

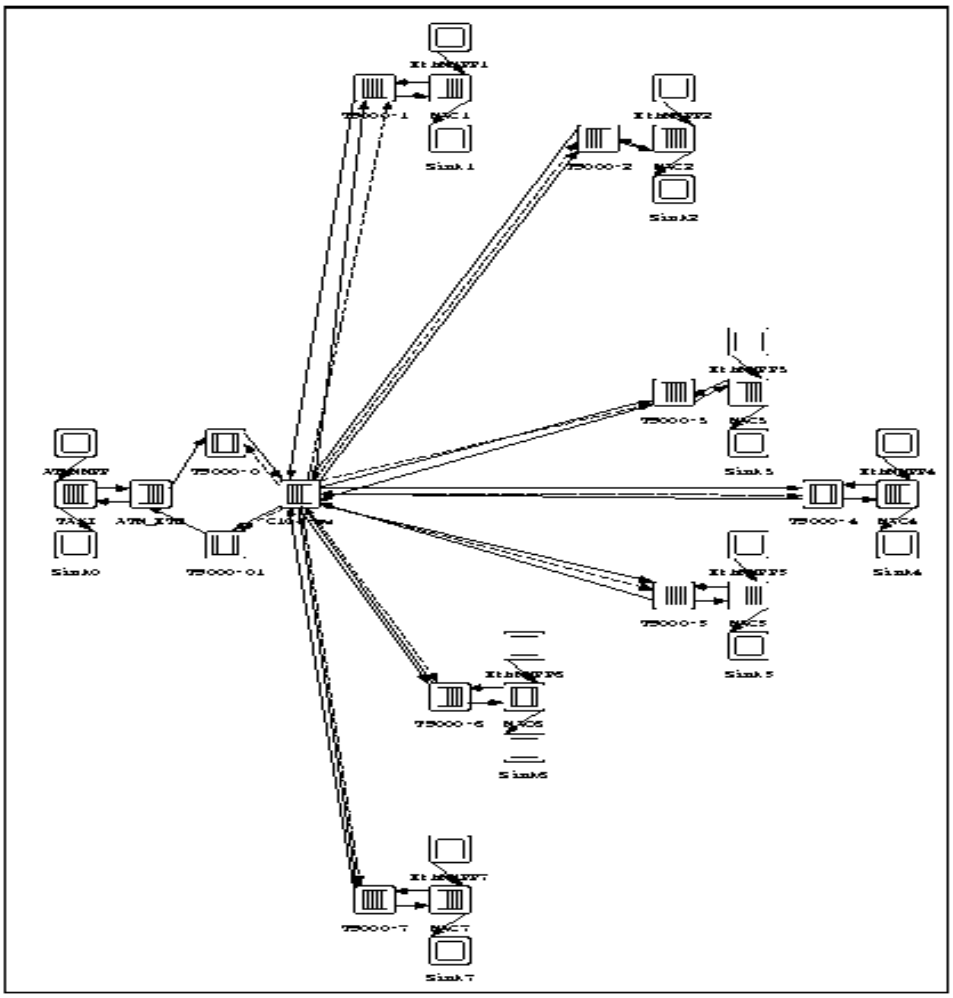

Figure B.3: ALAX Model at the Node Level in OPNET (8-port configuration)

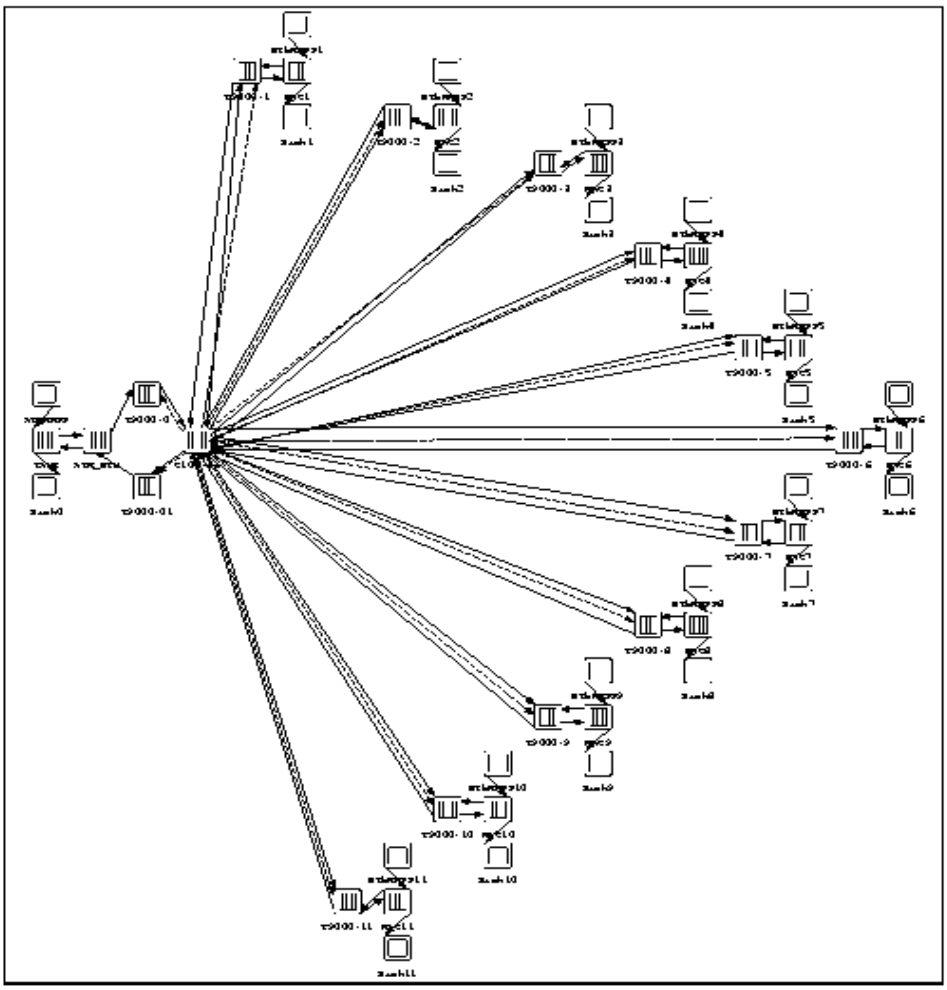

Figure B.4: ALAX Model at the Node Level in OPNET (12-port configuration)

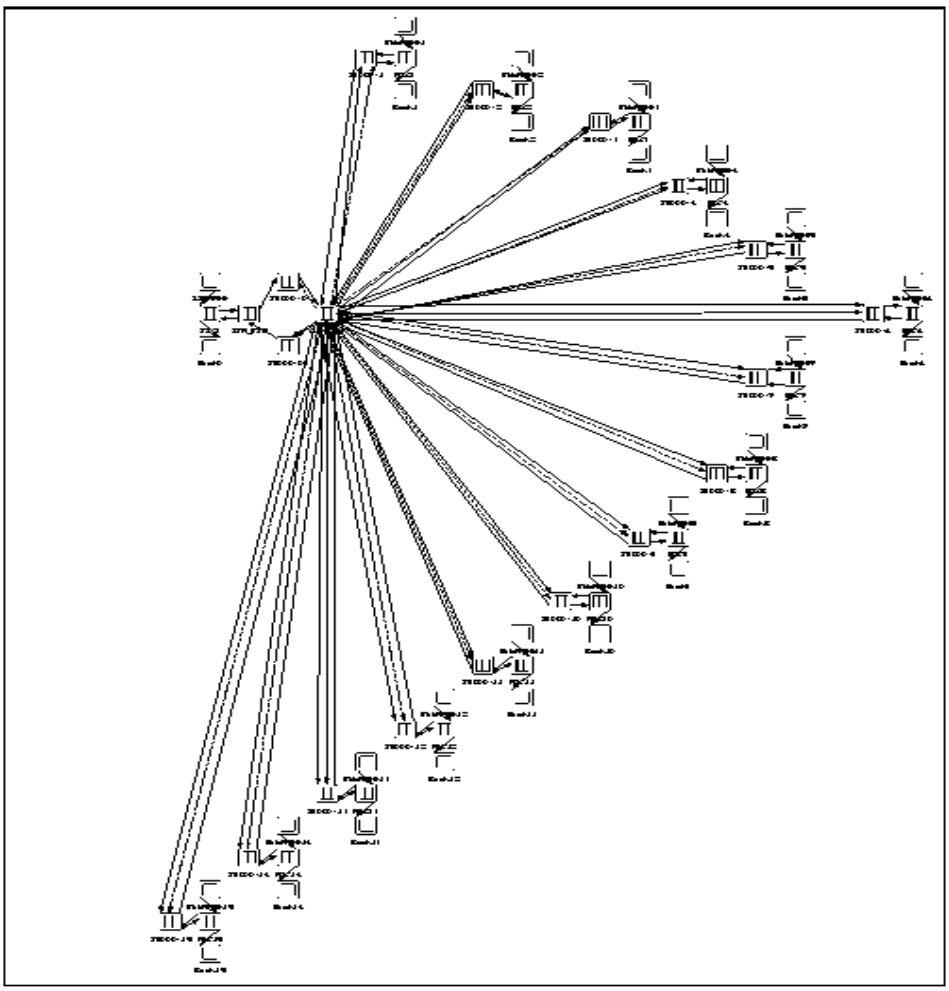

Figure B.5: ALAX Model at the Node Level in OPNET (16-port configuration)

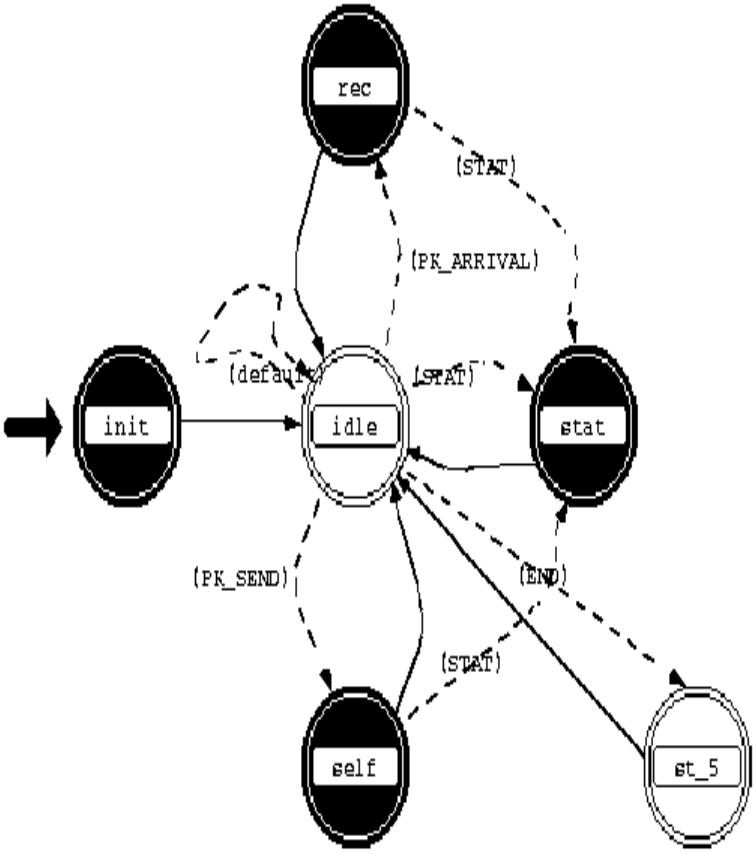

Figure B.6: T9000 FSM Model at the Process Level in OPNET (for all configurations implemented)

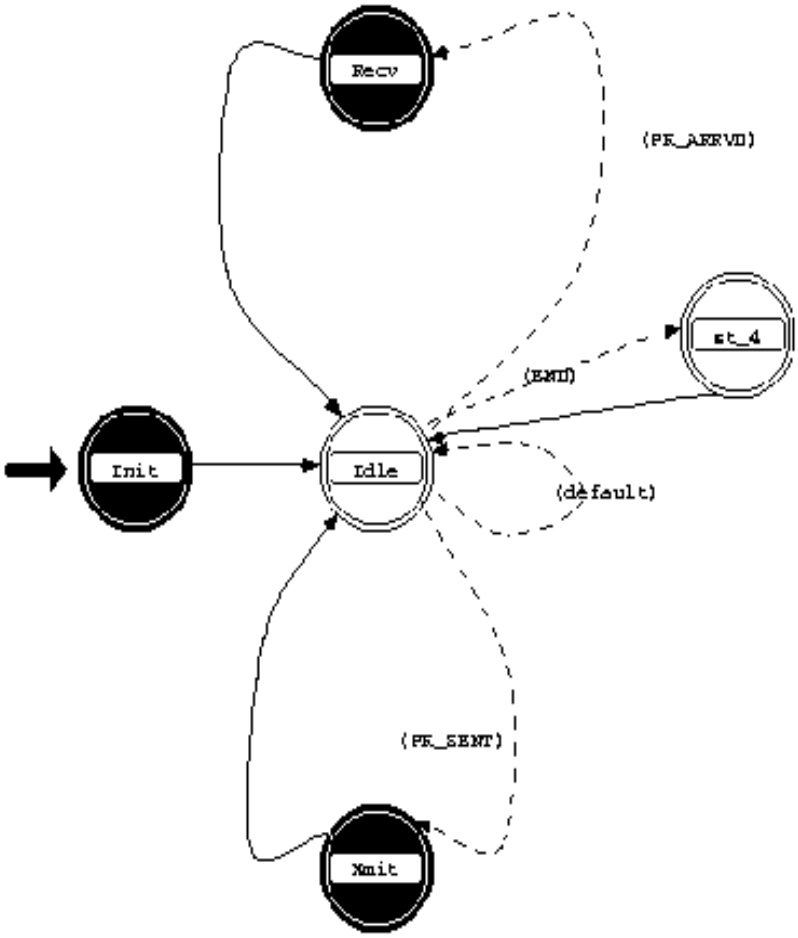

Figure B.7: C104 FSM Model at the Process Level in OPNET (for all configurations implemented)

B.1.1 Programming Code for the ATM Port T9000 (4-by-4 Case)

...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
INTERARRIVAL_ATM	promoted	double	3.533319E-05
MAX_EXT_MODULES	promoted	integer	4
T_DEL	promoted	double	1.28E-06
mult	promoted	double	1.0
OUTPUT_START	promoted	double	6.0

Header Block

```
#define STAT_WIRE0 0
#define THRESHOLD  1
#define STAT (op_intrpt_type() == OPC_INTRPT_STAT)
#define PK_ARRIVAL (op_intrpt_type() == OPC_INTRPT_STRM)
#define PK_SEND (op_intrpt_type() == OPC_INTRPT_SELF)
#define END op_intrpt_type()==OPC_INTRPT_ENDSIM
#define MAC_QUEUE 0
#define P1355_QUEUE 1
#define CODE0 0
#define CODE1 1
#define CODE2 2
#define CODE3 3
#define CODE4 4
/*#define OUTPUT_START 2.0  */
/*OUTPUT_START is now passed by the user thru ef or at run time */
```

State Variable Block

```
int      \conch;
int      \flag_HOQ_down[64];  /*we won't exceed 64 for MAX_EXT_MODULES*/
int      \subqueue;
double   \last_arrival_time;
int      \MAX_EXT_MODULES;
double   \T_DEL;
double   \T_LANE;
double   \SERV_DIFF;
double   \UPDATE;
int      \queries;
double   \SERV_TIME_DOWN;
double   \P1355_SEND_TIME;
double   \t_last;
double   \t_1;
int      \stat_flag;
int      \self_flag;
Gshandle  \ete_gsh1;
Gshandle  \ete_gsh3;
Gshandle  \ete_gsh4;
Gshandle  \ete_gsh5;
double   \mult;
double \mean_q[10],\prev_q[10],\prev_time;
double \tot_pkt[10];
double \OUTPUT_START;
double \atmhigh_q[10],\atmvar_q[10];
```

…
…

Temporary Variable Block

```
int       queue_condition;
int       i,j,k;
int       max_num_p1355;
double     time;
Packet*    pkptr;
Packet*    pkptr1;
Packet*    pkptr2;
int       service;
int       rounds;
int       origen;
int       condition;
int       condition1;
int       condition2;
int       eth_size;
double     interarrival_atm;
double     t_lane;
double     t_trans;
int       flag;
int       address;
int       N;
double       bit_interarrival_atm;
```

unforced state **idle**

attribute	*value*	*type*	*default value*
name	idle	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

enter execs **idle**

```
/*****************************************************************************/
/*we'll send a P1355 to the C104 if (1.) the subqueue is not empty,   */
/*(2.) there is no backpressure, and (3.) the last P1355 has not been */
/*sent in less than P1355_SEND_TIME seconds                          */
/*****************************************************************************/
if (((op_subq_stat(P1355_QUEUE,OPC_QSTAT_PKSIZE)>0)&&(stat_flag==0))&&((op_sim_time()-t_1)>P1355_SEND_TIME)
pkptr = op_subq_pk_access(P1355_QUEUE,OPC_QPOS_HEAD);

/*t_trans = t_SENT + t_TRANS   */
/*it will have been written at the preceding stage*/
op_pk_nfd_get(pkptr,"TRANS",&t_trans);

/*****************************************************************/
/* present op_sim_time() will take into account the delay for*/
/* t_LANE and t_DEL(first 16 bytes) by virtue of the self    */
/* interrupt initiated when the MAC gets serviced           */
/*****************************************************************/
t_lane = op_sim_time();

/*this sends a P1355 to the C104*/
if (t_lane>t_trans) {
   pkptr = op_subq_pk_remove(P1355_QUEUE,OPC_QPOS_HEAD);
   op_pk_send(pkptr,0);
```

...
...

```
    t_1 = op_sim_time();
25  }
    if (op_sim_time()>OUTPUT_START) {op_stat_global_write(ete_gsh1,op_subq_stat(1,OPC_QSTAT_PKSIZE));}
}

else if (self_flag==0) {
30 op_intrpt_schedule_self(op_sim_time() +  UPDATE,CODE3);
self_flag=1;
}

if (op_subq_stat(10,OPC_QSTAT_PKSIZE)>0) {
35 pkptr = op_subq_pk_remove(10,OPC_QPOS_HEAD);
op_pk_send(pkptr,1);
}
```

exit execs **idle**

```
/*if(op_sim_time()>OUTPUT_START)
{op_stat_local_write(0,op_subq_stat(0,OPC_QSTAT_PKSIZE));
 op_stat_local_write(1,op_subq_stat(1,OPC_QSTAT_PKSIZE));
op_stat_local_write(2,op_subq_stat(3,OPC_QSTAT_PKSIZE));
5 op_stat_local_write(3,op_subq_stat(4,OPC_QSTAT_PKSIZE));
op_stat_local_write(4,op_subq_stat(5,OPC_QSTAT_PKSIZE));
}
  */

10 /*The above stuff writes onto output vectors and has been disabled for t
the present*/

 for(i=0;i<10;i=i+1){
mean_q[i]=mean_q[i]+prev_q[i]*(op_sim_time()-prev_time);
15 prev_q[i]=op_subq_stat(i,OPC_QSTAT_PKSIZE);
tot_pkt[i]=tot_pkt[i]+1;
if (atmhigh_q[i]<prev_q[i]) {atmhigh_q[i]=prev_q[i];}
atmvar_q[i]=atmvar_q[i]+ prev_q[i]*prev_q[i]*(op_sim_time()-prev_time);
}
20
prev_time=op_sim_time();
```

transition **idle -> self**

attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	PK_SEND	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> rec**

attribute	*value*	*type*	*default value*
name	tr_3	string	tr
condition	PK_ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

transition **idle -> stat**			
attribute	*value*	*type*	*default value*
name	tr_8	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> idle**			
attribute	*value*	*type*	*default value*
name	tr_10	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> st_5**			
attribute	*value*	*type*	*default value*
name	tr_13	string	tr
condition	END	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **rec**			
attribute	*value*	*type*	*default value*
name	rec	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **rec**

```
/* get packet from stream */
pkptr = op_pk_get(op_intrpt_strm());

/*****************************************************************************/
/*  ETH packets from ATM_ETH-->T9000 or MAC-->T9000                          */
/*  the emptiness or non_emptiness of MAC_QUEUE subqueue implies whether     */
/*  T_DEL will be appended to the T_LANE when service is done                */
/*  if MAC_QUEUE is nonempty, then it is assumed that the 16 bytes that      */
/*  contribute to T_DEL will have already arrived by the time LANE starts    */
/*  total delay - suggested to be 80% interarrival time                      */
/*****************************************************************************/
/* (a.) MAC_QUEUE subqueue is empty */
if ((op_intrpt_strm()==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)==0)) {
   flag=0;
   op_pk_nfd_set(pkptr,"FLAG",flag);
   op_subq_pk_insert(MAC_QUEUE,pkptr,OPC_QPOS_TAIL);
   }

/*(b.) MAC_QUEUE subqueue is nonempty*/
```

...
...

```
20  else if ((op_intrpt_strm()==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)>0)) {
      flag=1;
      op_pk_nfd_set(pkptr,"FLAG",flag);
      op_subq_pk_insert(MAC_QUEUE,pkptr,OPC_QPOS_TAIL);
    }
25
    /*********************************************************************/
    /*P1355 packets from C104 --> T9000 to be put into subqueues that */
    /*correspond to their origination                                   */
    /*********************************************************************/
30  else if (op_intrpt_strm()==1) {
      op_pk_nfd_get(pkptr,"ORIGEN",&origen);
      op_pk_nfd_get(pkptr,"dest_address",&address);
      op_subq_pk_insert(origen+2,pkptr,OPC_QPOS_TAIL);
      pkptr = op_subq_pk_access(origen+2,OPC_QPOS_HEAD);
35    op_pk_nfd_get(pkptr,"MAX_NUM_P1355",&max_num_p1355);
       if  ((op_sim_time()>OUTPUT_START)&&(origen==1)) {op_stat_global_write(ete_gsh3,op_subq_stat(origen+2,OPC_QSTA
       else if ((op_sim_time()>OUTPUT_START)&&(origen==2)) {op_stat_global_write(ete_gsh4,op_subq_stat(origen+2,OPC_QS
       else if ((op_sim_time()>OUTPUT_START)&&(origen==3)) {op_stat_global_write(ete_gsh5,op_subq_stat(origen+3,OPC_QS

40  /*************************************************************************/
    /*  if service has been given to the HOQ P1355 and this new addition     */
    /*  puts the subqueue occupancy over the edge(i.e. MAX_NUM_P1355 required) */
    /*  then send out an ETH packet and destroy MAX_NUM_P1355 from the HOQ   */
    /*************************************************************************/
45    if ((flag_HOQ_down[origen+2]==1)&&(op_subq_stat(origen+2,OPC_QSTAT_PKSIZE)>=max_num_p1355)) {
        pkptr1 = op_subq_pk_remove(origen+2,OPC_QPOS_HEAD);
        i=1;
        while (i<max_num_p1355) {
          op_pk_destroy(pkptr1);
50        pkptr1 = op_subq_pk_remove(origen+2,OPC_QPOS_HEAD);
          if (op_subq_stat(origen+2,OPC_QSTAT_PKSIZE)==0) {i=max_num_p1355;}
          i++;
          }
        op_pk_nfd_strip(pkptr1,"P1355");
55      op_pk_nfd_set(pkptr1,"ETH");
        op_pk_nfd_set(pkptr1,"NUM",max_num_p1355); /*ETH doesn't have weight the way Sandeep set it... so indicate size this way*/
        op_pk_nfd_get(pkptr1,"NUM",&max_num_p1355);
        op_subq_pk_insert(10,pkptr1,OPC_QPOS_TAIL);
        flag_HOQ_down[origen+2]=0;
60      if ((op_sim_time()>OUTPUT_START)&&(origen==1)) {op_stat_global_write(ete_gsh3,op_subq_stat(origen+2,OPC_QSTA
        else if ((op_sim_time()>OUTPUT_START)&&(origen==2)) {op_stat_global_write(ete_gsh4,op_subq_stat(origen+2,OPC_QS
        else if ((op_sim_time()>OUTPUT_START)&&(origen==3)) {op_stat_global_write(ete_gsh5,op_subq_stat(origen+2,OPC_QS
       }
    }
65
```

transition **rec -> idle**			
attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

transition **rec -> stat**

attribute	*value*	*type*	*default value*
name	tr_12	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **self**

attribute	*value*	*type*	*default value*
name	self	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **self**

```
    /*****************************************************************************************/
    /*****************************************************************************************/
    /**  C104 is the center of the roving server.  When condition C104 is true then the    **/
    /**  server has been released from serving the up-side(CODE1) or the                   **/
 5  /**  down-side(CODE2) of the T9000.  Upon a CODE0 interrupt, the roving server will    **/
    /**  alternate(i.e., up->down or down->up) and service the corresponding part of       **/
    /**  the T9000.  If this part of the T9000 doesn't require service, the roving server  **/
    /**  reverts back to the former part of the T9000.  If neither part requires service,  **/
    /**  then a self interrupt CODE0 is originated for SERV_DIFF seconds after the present **/
10  /**  time.  For example, (1.) just serviced top, (2.) switch and try service at bottom,**/
    /**  (3.) if service is not required at bottom, go to top and try service, (4.) if     **/
    /**  service is not required at top or bottom, then originate self interrupt for       **/
    /**  SERV_DIFF seconds into the future                                                  **/
    /*****************************************************************************************/
15  /*****************************************************************************************/
    if  (op_intrpt_code()==CODE0)  {
         queue_condition=0;
         for (i=0;i<MAX_EXT_MODULES;i++) {
         if ((op_subq_stat(i+2,OPC_QSTAT_PKSIZE)>0)&&(flag_HOQ_down[i+2]==0)) {
20         queue_condition++;              /*(queue_condition>0)implies we can service a subqueue on bottom*/
           }
         }

    /*****************************************************************************************/
25  /* The roving server is on top, and there is non-zero queue occupancy in SUBQ0          */
    /*****************************************************************************************/
    if ((conch==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)>0)) {
       pkptr = op_subq_pk_access(MAC_QUEUE,OPC_QPOS_HEAD);
       op_pk_nfd_get(pkptr,"FLAG",&flag);       /*flag queue occupancy empty (=0) or non-zero(=1)*/
30     condition1 = (flag==1)&&(last_arrival_time<=op_sim_time());
       condition2 = (flag==1)&&(last_arrival_time>op_sim_time());
        /*************************************************************/
        /* MAC packet in top of T9000 has not been given service yet*/
        /*************************************************************/
35       if (flag==0) {
         op_intrpt_schedule_self(op_sim_time()+T_LANE+T_DEL,CODE1);
           }
        /*****************************************************************************************/
        /*MAC packet has been given service, and "the end of the last packet has arrived"      */
```

```
40      /*****************************************************************************************/
        else if (condition1) {
          op_intrpt_schedule_self(op_sim_time()+T_LANE,CODE1);
        }
        /*****************************************************************************************/
45      /* MAC packet has been given service, but "the end of the last packet has not arrived" */
        /*****************************************************************************************/
        else if ((condition2)&&(last_arrival_time>=op_sim_time()+T_LANE)) {
          op_intrpt_schedule_self(last_arrival_time,CODE1);
        }
50      else if ((condition2)&&(last_arrival_time<op_sim_time()+T_LANE)) {
          op_intrpt_schedule_self(op_sim_time()+T_LANE,CODE1);
        }
        queries=0;
      }
55
/*****************************************************************************************/
/* the roving server is at the bottom, and condition>0 ==> there exists some queue with */
/* non-zero queue occupancy in SUBQ2-SUBQ"MAX"                                          */
/*****************************************************************************************/
60  else if ((conch==1)&&(queue_condition>0)) {
      op_intrpt_schedule_self(op_sim_time()+SERV_TIME_DOWN,CODE2);
      queries=0;
    }

65  /*****************************************************************************************/
    /* we have checked either top or bottom but not both                                    */
    /*****************************************************************************************/
    else if (queries<1) {
          conch=conch^1;
70        op_intrpt_schedule_self(op_sim_time(),CODE0);
          queries++;
      }

    /*****************************************************************************************/
75  /* we have checked both top and bottom to no avail... don't waste time or interrupts... */
    /* come back and check again after a delayed self interrupt                             */
    /*****************************************************************************************/
    else if (queries==1) {
          conch=conch^1;
80        time = op_sim_time()+SERV_DIFF;
          op_intrpt_schedule_self(time,CODE0);
          queries=0;
          }
    }
85
    /*****************************************************************************************/
    /*****************************************************************************************/
    /** A true condition for CODE1 signifies that the top half(i.e. T9000->C104 flow) **/
    /**   is being given service.                                                         **/
90  /*****************************************************************************************/
    /*****************************************************************************************/
    else if (op_intrpt_code()==CODE1) {
        pkptr1 = op_subq_pk_remove(MAC_QUEUE,OPC_QPOS_HEAD);
        eth_size = op_pk_nfd_size(pkptr1,"ETH");
95          max_num_p1355 = eth_size/(32*8);   /*change this to ceiling function(.) (?)*/
            /*************************************************************************/
            /* send first p1355 with flag=1, and field information about max_num_p1355 */
            /* NB: flag and MAX_NUM_P1355 fields ought not have a nonzero bit size     */
```

```
        /*****************************************************************************/
100     pkptr2 = op_pk_copy(pkptr1);
        op_pk_nfd_set(pkptr2,"P1355",1);
        op_pk_nfd_strip(pkptr2,"ETH");
        op_pk_nfd_set(pkptr2,"MAX_NUM_P1355",max_num_p1355);
        op_subq_pk_insert(P1355_QUEUE,pkptr2,OPC_QPOS_TAIL);
105     /*****************************************************************************/
        /* send max_num_p1355 p1355s with flag=0 for all P1355 packets except the first one*/
        /*****************************************************************************/
        for (i=1;i<max_num_p1355;i++) {
        pkptr2 = op_pk_copy(pkptr1);
110     op_pk_nfd_set(pkptr2,"P1355",0);
        op_pk_nfd_set(pkptr2,"MAX_NUM_P1355",max_num_p1355);
        op_pk_nfd_strip(pkptr2,"ETH");
        op_pk_nfd_get(pkptr2,"TRANS",&last_arrival_time);
        op_subq_pk_insert(P1355_QUEUE,pkptr2,OPC_QPOS_TAIL);
115     }
        op_pk_destroy(pkptr1);
        if (op_sim_time()>OUTPUT_START) {op_stat_global_write(ete_gsh1,op_subq_stat(1,OPC_QSTAT_PKSIZE));}


120     /*************************************************/
        /* XOR the roving server: (0->1) or (1->0)       */
        /* go immediately back to CODE0... control center */
        /*************************************************/
        conch=conch^1;
125     time = op_sim_time();
        op_intrpt_schedule_self(time,CODE0);
    }


130
    /*****************************************************************************************/
    /*****************************************************************************************/
    /** A true condition for CODE2 signifies that the bottom half(i.e. C104->T9000 flow) **/
    /**   is being given service                                                          **/
135 /*****************************************************************************************/
    /*****************************************************************************************/
    else if (op_intrpt_code()==CODE2) {
        /*initialize: ~locally*/
        service=0;
140     rounds=0;
        /*********************************************************************/
        /*give the next subqueue a chance for the service and send back to */
        /*the beginning if the cardinality of destinations has been reached*/
        /*********************************************************************/
145     if (subqueue<MAX_EXT_MODULES+1) {subqueue++;}
        else {subqueue=2;}
        /*****************************************************************************/
        /*jump out of loop if no service after checking "cardinality of destinations"*/
        /*number of subqueues or service is performed on one subqueue               */
150     /*****************************************************************************/
        while ((rounds<=MAX_EXT_MODULES+1)&&(service==0)) {
          if ((op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>0)&&(flag_HOQ_down[subqueue]==0)) {
            flag_HOQ_down[subqueue]=1;
            service=1;
155         }
          else if (subqueue<MAX_EXT_MODULES+1) {subqueue++; rounds++;}
          else if (subqueue==MAX_EXT_MODULES+1) {subqueue=2; rounds++;}
```

```
    }

160 /*****************************************************************************/
    /* only need to access the SUBQ in order to do op_subq_stat in next "if" statement*/
    /*****************************************************************************/
    if (op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>0) {
    pkptr = op_subq_pk_access(subqueue,OPC_QPOS_HEAD);
165 op_pk_nfd_get(pkptr,"MAX_NUM_P1355",&max_num_p1355);
    }

    /*****************************************************************************/
    /* service has been given and there are a commensurate number of P1355 to create the MAC */
170 /*****************************************************************************/
    if ((flag_HOQ_down[subqueue]==1)&&(op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>=max_num_p1355)) {
      pkptr1 = op_subq_pk_remove(subqueue,OPC_QPOS_HEAD);
      i=1;
      while (i<max_num_p1355) {
175     op_pk_destroy(pkptr1);
        pkptr1 = op_subq_pk_remove(subqueue,OPC_QPOS_HEAD);
        if (op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)==0) {i=max_num_p1355;}   /*an earlier condition ought to have requi
        i++;
        }
180   op_pk_nfd_strip(pkptr1,"P1355");
      op_pk_nfd_set(pkptr1,"ETH");
      op_pk_nfd_set(pkptr1,"NUM",max_num_p1355); /*ETH doesn't take a size the way Sandeep set it... so give indication of size t
      op_subq_pk_insert(10,pkptr1,OPC_QPOS_TAIL);
      flag_HOQ_down[subqueue]=0;
185   }
     if ((op_sim_time()>OUTPUT_START)&&(subqueue==3))  {op_stat_global_write(ete_gsh3,op_subq_stat(subqueue,OPC_Q
     else if ((op_sim_time()>OUTPUT_START)&&(subqueue==4)) {op_stat_global_write(ete_gsh4,op_subq_stat(subqueue,OPC
     else if ((op_sim_time()>OUTPUT_START)&&(subqueue==5)) {op_stat_global_write(ete_gsh5,op_subq_stat(subqueue,OPC

190  /*******************************/
     /* same situation as in CODE1 */
     /*******************************/
     conch=conch^1;
     time=op_sim_time();
195  op_intrpt_schedule_self(time,CODE0);
}

/*******************/
/*******************/
200 /** CODE3 occurs **/
/*******************/
/*******************/
else if (op_intrpt_code()==CODE3) {
if (op_stat_local_read(STAT_WIRE0)==0) {stat_flag=0;}
205 else if (op_stat_local_read(STAT_WIRE0)==1) {stat_flag=1;}
self_flag=0;
}


210
```

...
...

transition **self -> idle**			
attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **self -> stat**			
attribute	*value*	*type*	*default value*
name	tr_11	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **init**			
attribute	*value*	*type*	*default value*
name	init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	forced	toggle	unforced

enter execs **init**

```
/*************************/
/* initialize parameters*/
/*************************/
for (i=0;i<=MAX_EXT_MODULES;i++) {
   flag_HOQ_down[i+2]=0;                    /*represents whether the subqueue "i" has been given already*/
   }
conch=0;
queries=0;
subqueue=2;                              /*subqueue starts at 2 because this parameter indicates the first subqueue in the lower half of T
last_arrival_time=0;
stat_flag = 0;
self_flag = 0;

/*************************/
/*obtain promoted values*/
/*************************/
op_ima_obj_attr_get(op_id_self(),"INTERARRIVAL_ATM",&interarrival_atm);
op_ima_obj_attr_get(op_id_self(),"MAX_EXT_MODULES",&MAX_EXT_MODULES);
op_ima_obj_attr_get(op_id_self(),"T_DEL",&T_DEL);
op_ima_obj_attr_get(op_id_self(),"mult",&mult);
op_ima_obj_attr_get(op_id_self(),"OUTPUT_START",&OUTPUT_START);
/*************************************************************/
/*assume DELAYS dependent on promoted values just obtained*/
/*************************************************************/
bit_interarrival_atm = 1.0/28000000.0;          /*currently, rated at 12Mbps*/
T_LANE = mult*12570*bit_interarrival_atm;       /*service time parameter on top of T9000*/
SERV_TIME_DOWN = mult*12570*bit_interarrival_atm;  /*service time parameter on bottom of T9000*/

UPDATE = .01*interarrival_atm;                  /*rechecks to send P1355 after UPDATE...  is heuristic*/
```

...
...

```
SERV_DIFF = .5*interarrival_atm;                /*heuristic... for now*/
/*P1355_SEND_TIME = .00001*interarrival_atm;      heuristic... perhaps, should be in terms of ETH interarrival or something els
t_1=0;

ete_gsh1 = op_stat_global_reg("t9000_atm_side_subqueue1");
ete_gsh3 = op_stat_global_reg("t9000_atm_side_subqueue3");
ete_gsh4 = op_stat_global_reg("t9000_atm_side_subqueue4");
ete_gsh5 = op_stat_global_reg("t9000_atm_side_subqueue5");



for(i=0;i<10;i=i+1){
mean_q[i]=0;
prev_q[i]=0;
tot_pkt[i]=0;
atmhigh_q[i]=0;
atmvar_q[i]=0;
}

prev_time=0;
```

exit execs **init**

```
op_intrpt_schedule_self(op_sim_time()+interarrival_atm,CODE0);
```

transition **init -> idle**

attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **stat**

attribute	*value*	*type*	*default value*
name	stat	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **stat**

```
/*************************************************************/
/* there has been a change in the statistic wire backpressure*/
/* modify the stat_flag accordingly
/*************************************************************/
if (op_stat_local_read(STAT_WIRE0)==0) {stat_flag=0;}
else if (op_stat_local_read(STAT_WIRE0)==1) {stat_flag=1;}
```

...
...

transition **stat -> idle**			
attribute	*value*	*type*	*default value*
name	tr_9	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **st_5**			
attribute	*value*	*type*	*default value*
name	st_5	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **st_5**

```
 for(i=0;i<10;i=i+1){
printf("(atm side t9000)average_queue_lenght of queue[%d]=%lf\n",i,mean_q[i]/op_sim_time());
op_stat_scalar_write("ave_queue_length(atm side)",mean_q[i]/op_sim_time());
printf("(atm side T9000) highest queue size[%d]=%1f\n",i,atmhigh_q[i]);
printf("(atm side T9000)  variances in queue size[%d]=%1f\n",i,sqrt((atmvar_q[i]/op_sim_time())-(mean_q[i]/op_sim
}
```

transition **st_5 -> idle**			
attribute	*value*	*type*	*default value*
name	tr_15	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

B.1.2 Programming Code for the Ethernet Port T9000 (4-by-4 Case)

...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
INTERARRIVAL_ATM	promoted	double	3.533319E-05
MAX_EXT_MODULES	promoted	integer	4
T_DEL	promoted	double	1.28E-06
GUGU	promoted	integer	0

Header Block

```
#define STAT_WIRE0 0
#define THRESHOLD  1
#define STAT (op_intrpt_type() == OPC_INTRPT_STAT)
#define PK_ARRIVAL (op_intrpt_type() == OPC_INTRPT_STRM)
#define PK_SEND (op_intrpt_type() == OPC_INTRPT_SELF)
#define MAC_QUEUE 0
#define P1355_QUEUE 1
#define CODE0 0
#define CODE1 1
#define CODE2 2
#define CODE3 3
#define CODE4 4
#define END op_intrpt_type()==OPC_INTRPT_ENDSIM
```

State Variable Block

```
int      \conch;
int      \flag_HOQ_down[64];  /*we won't exceed 64 for MAX_EXT_MODULES*/
int      \subqueue;
double   \last_arrival_time;
int      \MAX_EXT_MODULES;
double   \T_DEL;
double   \T_LANE;
double   \UPDATE;
double   \SERV_DIFF;
int      \queries;
double   \SERV_TIME_DOWN;
double   \t_last
double   \t_1;
int      \stat_flag;
double   \P1355_SEND_TIME;
int      \self_flag;
int \gugu;
double \mean_q[10],\prev_q[10],\prev_time,\tot_pkt[10];
int  \sum11;
int \no_of_insertions;
double \ethhigh_q[10],\ethvar_q[10];
```

Temporary Variable Block

```
int        queue_condition;
int        i,j,k;
int        max_num_p1355;
double     time;
Packet*    pkptr;
Packet*    pkptr1;
Packet*    pkptr2;
int        service;
```

...
...

```
int       rounds;
int       origen;
int       condition;
int       condition1;
int       condition2;
int       eth_size;
double    interarrival_atm;
double    t_lane;
double    t_trans;
int       flag;
int       address;
int       N;
int       bit_interarrival_atm;
```

unforced state **idle**

attribute	*value*	*type*	*default value*
name	idle	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

enter execs **idle**

```
/*****************************************************************/
/*we'll send a P1355 to the C104 if (1.) the subqueue is not empty,   */
/*(2.) there is no backpressure, and (3.) the last P1355 has not been */
/*sent in less than P1355_SEND_TIME seconds                          */
/*****************************************************************/
if (((op_subq_stat(P1355_QUEUE,OPC_QSTAT_PKSIZE)>0)&&(stat_flag==0))&&((op_sim_time()-t_1)>P1355_SEND_TIME)
pkptr = op_subq_pk_access(P1355_QUEUE,OPC_QPOS_HEAD);

/*t_trans = t_SENT + t_TRANS   */
/*it will have been written at the preceding stage*/
op_pk_nfd_get(pkptr,"TRANS",&t_trans);

/*****************************************************************/
/* present op_sim_time() will take into account the delay for*/
/* t_LANE and t_DEL(first 16 bytes) by virtue of the self    */
/* interrupt initiated when the MAC gets serviced           */
/*****************************************************************/
t_lane = op_sim_time();

/*this sends a P1355 to the C104*/
if (t_lane>t_trans) {
   pkptr = op_subq_pk_remove(P1355_QUEUE,OPC_QPOS_HEAD);
   op_pk_send(pkptr,0);
   t_1 = op_sim_time();
   }
}

else if (self_flag==0) {
op_intrpt_schedule_self(op_sim_time() +  UPDATE,CODE3);
self_flag=1;
```

...
...

```
    }

    if (op_subq_stat(6,OPC_QSTAT_PKSIZE)>0) {
    pkptr = op_subq_pk_remove(6,OPC_QPOS_HEAD);
35  op_pk_send(pkptr,1);
    }
```

exit execs **idle**

```
    for(i=0;i<10;i=i+1)
    {
 5  mean_q[i]=mean_q[i]+prev_q[i]*(op_sim_time()-prev_time);

    if (ethhigh_q[i] < prev_q[i]) {ethhigh_q[i]=prev_q[i];}
    ethvar_q[i]=ethvar_q[i]+prev_q[i]*prev_q[i]*(op_sim_time()-prev_time);

10  prev_q[i]=op_subq_stat(i,OPC_QSTAT_PKSIZE);
    tot_pkt[i]=tot_pkt[i]+1;
    }

    /*if(prev_q[2]>0){printf("prev_q[2]=%lf\n",prev_q[2]);}; */
15  if((gugu==1)&(prev_q[3]>0)){printf("q3 caught red handed");};
    prev_time=op_sim_time();
```

transition **idle -> self**

attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	PK_SEND	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> rec**

attribute	*value*	*type*	*default value*
name	tr_3	string	tr
condition	PK_ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> stat**

attribute	*value*	*type*	*default value*
name	tr_8	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

transition **idle -> idle**			
attribute	*value*	*type*	*default value*
name	tr_10	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> st_5**			
attribute	*value*	*type*	*default value*
name	tr_13	string	tr
condition	END	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **rec**			
attribute	*value*	*type*	*default value*
name	rec	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **rec**

```
/* get packet from stream */
pkptr = op_pk_get(op_intrpt_strm());
op_pk_nfd_get (pkptr, "ORIGEN", &origen);
if(origen==0){sum11=sum11+1;}
op_pk_nfd_get (pkptr, "dest_address", &address);
/*printf ("origen=%d,address=%d \n", origen,address);*/

/*****************************************************************************/
/*  ETH packets from ATM_ETH-->T9000 or MAC-->T9000                          */
/*  the emptiness or non_emptiness of MAC_QUEUE subqueue implies whether     */
/*  T_DEL will be appended to the T_LANE when service is done                */
/*  if MAC_QUEUE is nonempty, then it is assumed that the 16 bytes that      */
/*  contribute to T_DEL will have already arrived by the time LANE starts    */
/*  total delay - suggested to be 80% interarrival time                      */
/*****************************************************************************/

/* (a.) MAC_QUEUE subqueue is empty */
if ((op_intrpt_strm()==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)==0)) {
   flag=0;
   op_pk_nfd_set(pkptr,"FLAG",flag);
   op_subq_pk_insert(MAC_QUEUE,pkptr,OPC_QPOS_TAIL);
   }

/*(b.) MAC_QUEUE subqueue is nonempty*/
else if ((op_intrpt_strm()==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)>0)) {
   flag=1;
   op_pk_nfd_set(pkptr,"FLAG",flag);
   op_subq_pk_insert(MAC_QUEUE,pkptr,OPC_QPOS_TAIL);
}
```

...
...

```
30
   /*********************************************************************/
   /*P1355 packets from C104 --> T9000 to be put into subqueues that */
   /*correspond to their origination                                   */
   /*********************************************************************/
35 else if (op_intrpt_strm()==1) {
     op_pk_nfd_get(pkptr,"ORIGEN",&origen);
     op_pk_nfd_get(pkptr,"dest_address",&address);
     op_subq_pk_insert(origen+2,pkptr,OPC_QPOS_TAIL);
    if(origen==0){  no_of_insertions=no_of_insertions+1;}
40   pkptr = op_subq_pk_access(origen+2,OPC_QPOS_HEAD);
     op_pk_nfd_get(pkptr,"MAX_NUM_P1355",&max_num_p1355);

   /*************************************************************************/
   /*  if service has been given to the HOQ P1355 and this new addition     */
45 /*  puts the subqueue occupancy over the edge(i.e. MAX_NUM_P1355 required) */
   /*  then send out an ETH packet and destroy MAX_NUM_P1355 from the HOQ   */
   /*************************************************************************/
     if ((flag_HOQ_down[origen+2]==1)&&(op_subq_stat(origen+2,OPC_QSTAT_PKSIZE)>=max_num_p1355)) {
       pkptr1 = op_subq_pk_remove(origen+2,OPC_QPOS_HEAD);
50     i=1;
       while (i<max_num_p1355) {
         op_pk_destroy(pkptr1);
         pkptr1 = op_subq_pk_remove(origen+2,OPC_QPOS_HEAD);
         if (op_subq_stat(origen+2,OPC_QSTAT_PKSIZE)==0) {i=max_num_p1355;}
55       i++;
         }
       op_pk_nfd_strip(pkptr1,"P1355");
       op_pk_nfd_set(pkptr1,"ETH");
       op_pk_nfd_set(pkptr1,"NUM",max_num_p1355); /*ETH doesn't have weight the way Sandeep set it... so indicate size this way*/
60     op_subq_pk_insert(10,pkptr1,OPC_QPOS_TAIL);
       flag_HOQ_down[origen+2]=0;
      }
   }
```

transition **rec -> idle**			
attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **rec -> stat**			
attribute	*value*	*type*	*default value*
name	tr_12	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

forced state **self**

attribute	*value*	*type*	*default value*
name	self	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **self**

```
/*****************************************************************************************/
/*****************************************************************************************/
/**  C104 is the center of the roving server.  When condition C104 is true then the    **/
/**  server has been released from serving the up-side(CODE1) or the                   **/
/**  down-side(CODE2) of the T9000.  Upon a CODE0 interrupt, the roving server will    **/
/**  alternate(i.e., up->down or down->up) and service the corresponding part of       **/
/**  the T9000.  If this part of the T9000 doesn't require service, the roving server  **/
/**  reverts back to the former part of the T9000.  If neither part requires service,  **/
/**  then a self interrupt CODE0 is originated for SERV_DIFF seconds after the present **/
/**  time.  For example, (1.) just serviced top, (2.) switch and try service at bottom,**/
/**  (3.) if service is not required at bottom, go to top and try service, (4.) if     **/
/**  service is not required at top or bottom, then originate self interrupt for       **/
/**  SERV_DIFF seconds into the future                                                  **/
/*****************************************************************************************/
/*****************************************************************************************/
if  (op_intrpt_code()==CODE0)  {
      queue_condition=0;
      for (i=0;i<MAX_EXT_MODULES;i++) {
      if ((op_subq_stat(i+2,OPC_QSTAT_PKSIZE)>0)&&(flag_HOQ_down[i+2]==0)) {
        queue_condition++;  /*(queue_condition>0)implies we can service a subqueue on bottom*/
        }
      }

/*****************************************************************************************/
/* The roving server is on top, and there is non-zero queue occupancy in SUBQ0          */
/*****************************************************************************************/
if ((conch==0)&&(op_subq_stat(MAC_QUEUE,OPC_QSTAT_PKSIZE)>0)) {
   pkptr = op_subq_pk_access(MAC_QUEUE,OPC_QPOS_HEAD);
   op_pk_nfd_get(pkptr,"FLAG",&flag);       /*flag queue occupancy empty (=0) or non-zero(=1)*/
   condition1 = (flag==1)&&(last_arrival_time<=op_sim_time());
   condition2 = (flag==1)&&(last_arrival_time>op_sim_time());
    /*************************************************************/
    /* MAC packet in top of T9000 has not been given service yet*/
    /*************************************************************/
     if (flag==0) {
     op_intrpt_schedule_self(op_sim_time()+T_LANE+T_DEL,CODE1);
       }
     /*****************************************************************************************/
     /*MAC packet has been given service, but "the end of the last packet has arrived"      */
     /*****************************************************************************************/
     else if (condition1) {
       op_intrpt_schedule_self(op_sim_time()+T_LANE,CODE1);
     }
     /*****************************************************************************************/
     /* MAC packet has been given service, and "the end of the last packet has not arrived" */
     /*****************************************************************************************/
     else if ((condition2)&&(last_arrival_time>=op_sim_time()+T_LANE)) {
       op_intrpt_schedule_self(last_arrival_time,CODE1);
     }
```

```
    else if ((condition2)&&(last_arrival_time<op_sim_time()+T_LANE)) {
      op_intrpt_schedule_self(op_sim_time()+T_LANE,CODE1);
     }
    queries=0;
   }

/*****************************************************************************************/
/* the roving server is at the bottom, and condition>0 ==> there exists some queue with */
/* non-zero queue occupancy in SUBQ2-SUBQ"MAX"                                           */
/*****************************************************************************************/
else if ((conch==1)&&(queue_condition>0)) {
    op_intrpt_schedule_self(op_sim_time()+SERV_TIME_DOWN,CODE2);
    queries=0;
   }

/*****************************************************************************************/
/* we have checked either top or bottom but not both                                     */
/*****************************************************************************************/
else if (queries<1) {
      conch=conch^1;
      op_intrpt_schedule_self(op_sim_time(),CODE0);
      queries++;
   }

/*****************************************************************************************/
/* we have checked both top and bottom to no avail... don't waste time or interrupts... */
/* come back and check again after a delayed self interrupt                              */
/*****************************************************************************************/
else if (queries==1) {
      conch=conch^1;
      time = op_sim_time()+SERV_DIFF;
      op_intrpt_schedule_self(time,CODE0);
      queries=0;
      }
}



/*****************************************************************************************/
/*****************************************************************************************/
/** A true condition for CODE1 signifies that the top half(i.e. T9000->C104 flow) **/
/**   is being given service.                                                         **/
/*****************************************************************************************/
/*****************************************************************************************/
else if (op_intrpt_code()==CODE1) {
    pkptr1 = op_subq_pk_remove(MAC_QUEUE,OPC_QPOS_HEAD);
    eth_size = op_pk_nfd_size(pkptr1,"ETH");
       max_num_p1355 = eth_size/(32*8);   /*change this to ceiling function(.) (?)*/
       /*************************************************************************/
       /* send first p1355 with flag=1, and field information about max_num_p1355 */
       /* NB: flag and MAX_NUM_P1355 fields ought not have a nonzero bit size     */
       /*************************************************************************/
       pkptr2 = op_pk_copy(pkptr1);
       op_pk_nfd_set(pkptr2,"P1355",1);
       op_pk_nfd_strip(pkptr2,"ETH");
       op_pk_nfd_set(pkptr2,"MAX_NUM_P1355",max_num_p1355);
       op_subq_pk_insert(P1355_QUEUE,pkptr2,OPC_QPOS_TAIL);
       /*************************************************************************/
       /* send max_num_p1355 p1355s with flag=0 for all P1355 packets except the first one*/
```

```
        /*****************************************************************************/
        for (i=1;i<max_num_p1355;i++) {
        pkptr2 = op_pk_copy(pkptr1);
        op_pk_nfd_set(pkptr2,"P1355",0);
        op_pk_nfd_set(pkptr2,"MAX_NUM_P1355",max_num_p1355);
        op_pk_nfd_strip(pkptr2,"ETH");
        op_pk_nfd_get(pkptr2,"TRANS",&last_arrival_time);
        op_subq_pk_insert(P1355_QUEUE,pkptr2,OPC_QPOS_TAIL);
        }
        op_pk_destroy(pkptr1);

        /************************************************/
        /* XOR the roving server: (0->1) or (1->0)      */
        /* go immediately back to CODE0... control center */
        /************************************************/
        conch=conch^1;
        time = op_sim_time();
        op_intrpt_schedule_self(time,CODE0);
    }



/*****************************************************************************/
/*****************************************************************************/
/** A true condition for CODE2 signifies that the bottom half(i.e. C104->T9000 flow) **/
/**   is being given service                                                  **/
/*****************************************************************************/
/*****************************************************************************/
else if (op_intrpt_code()==CODE2) {
    /*initialize: ~locally*/
    service=0;
    rounds=0;
    /*********************************************************************/
    /*give the next subqueue a chance for the service and send back to */
    /*the beginning if the cardinality of destinations has been reached*/
    /*********************************************************************/
    if (subqueue<MAX_EXT_MODULES+1) {subqueue++;}
    else {subqueue=2;}
    /*****************************************************************************/
    /*jump out of loop if no service after checking "cardinality of destinations"*/
    /*number of subqueues or service is performed on one subqueue              */
    /*****************************************************************************/
    while ((rounds<=MAX_EXT_MODULES+1)&&(service==0)) {
      if ((op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>0)&&(flag_HOQ_down[subqueue]==0)) {
        flag_HOQ_down[subqueue]=1;
        service=1;
        }
      else if (subqueue<MAX_EXT_MODULES+1) {subqueue++; rounds++;}
      else if (subqueue==MAX_EXT_MODULES+1) {subqueue=2; rounds++;}
    }

    /*****************************************************************************/
    /* only need to access the SUBQ in order to do op_subq_stat in next "if" statement*/
    /*****************************************************************************/
    if (op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>0) {
    pkptr = op_subq_pk_access(subqueue,OPC_QPOS_HEAD);
    op_pk_nfd_get(pkptr,"MAX_NUM_P1355",&max_num_p1355);
    }
```

```
    /*****************************************************************************************/
    /* service has been given and there are a commensurate number of P1355 to create the MAC */
    /*****************************************************************************************/
    if ((flag_HOQ_down[subqueue]==1)&&(op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)>=max_num_p1355)) {
      pkptr1 = op_subq_pk_remove(subqueue,OPC_QPOS_HEAD);
      i=1;
      while (i<max_num_p1355) {
        op_pk_destroy(pkptr1);
        pkptr1 = op_subq_pk_remove(subqueue,OPC_QPOS_HEAD);
        if (op_subq_stat(subqueue,OPC_QSTAT_PKSIZE)==0) {i=max_num_p1355;}    /*an earlier condition ought to have requi
        i++;
        }
      op_pk_nfd_strip(pkptr1,"P1355");
      op_pk_nfd_set(pkptr1,"ETH");
      op_pk_nfd_set(pkptr1,"NUM",max_num_p1355); /*ETH doesn't take a size the way Sandeep set it... so give indication of size t
      op_subq_pk_insert(10,pkptr1,OPC_QPOS_TAIL);
      flag_HOQ_down[subqueue]=0;
      }

     /*******************************/
     /* same situation as in CODE1 */
     /*******************************/
     conch=conch^1;
     time=op_sim_time();
     op_intrpt_schedule_self(time,CODE0);
}

/*******************/
/*******************/
/** CODE3 occurs **/
/*******************/
/*******************/
else if (op_intrpt_code()==CODE3) {
if (op_stat_local_read(STAT_WIRE0)==0) {stat_flag=0;}
else if (op_stat_local_read(STAT_WIRE0)==1) {stat_flag=1;}
self_flag=0;
}

/*************************************************************************/
/*************************************************************************/
/**  condition CODE3 signifies that a P1355 will be sent to the C104**/
/*************************************************************************/
/*************************************************************************/
/*else if ((op_intrpt_code()==CODE3)&&(op_subq_stat(P1355_QUEUE,OPC_QSTAT_PKSIZE)>0)) { */
/*the second condition above - op_subq_stat(.)>0 - ought not be there*/
/*this condition should be checked in state IDLE; however, it seemed */
/*that there might have been a race condition, so for now check the  */
/*condition again here... this should be ok... just redundant        */
/*if (op_stat_local_read(STAT_WIRE0)<THRESHOLD) {  */
/*pkptr = op_subq_pk_remove(P1355_QUEUE,OPC_QPOS_HEAD); */
/*send to appropriate stream*/
/*op_pk_send_forced(pkptr,0);*/
/*}*/
/*}*/
/*------------------we're doing this in the IDLE state, now------------------*/
```

...
...

transition **self -> idle**			
attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **self -> stat**			
attribute	*value*	*type*	*default value*
name	tr_11	string	tr
condition	STAT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **init**			
attribute	*value*	*type*	*default value*
name	init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	forced	toggle	unforced

enter execs **init**

```
/*************************/
/* initialize parameters*/
/*************************/
for (i=0;i<=MAX_EXT_MODULES;i++) {
   flag_HOQ_down[i+2]=0;
   }
conch=0;
queries=0;
subqueue=2;   /*subqueue starts at 2 because this parameter indicates the lower half of T9000*/
last_arrival_time=0;
stat_flag = 0;
self_flag = 0;



/*************************/
/*obtain promoted values*/
/*************************/
op_ima_obj_attr_get(op_id_self(),"INTERARRIVAL_ATM",&interarrival_atm);
op_ima_obj_attr_get(op_id_self(),"MAX_EXT_MODULES",&MAX_EXT_MODULES);
op_ima_obj_attr_get(op_id_self(),"T_DEL",&T_DEL);
op_ima_obj_attr_get(op_id_self(),"GUGU",&gugu);
```

...
...

```
/*************************************************************/
/*assume DELAYS dependent on promoted values just obtained*/
/*************************************************************/
bit_interarrival_atm = 1/28000000;     /*promote later*/
T_LANE = 0;                            /*there is no LAN emulation... LAN emulation is on ATM side*/
SERV_TIME_DOWN = .1*24*32*8*bit_interarrival_atm;   /*10 percent interarrival... need time to reconstruct ETH packet*/
UPDATE = .01*interarrival_atm;         /*rechecks to send P1355 after UPDATE...  is heuristic*/

                                       /*if UPDATE too large... then too many interrupts*/
                                       /*if UPDATE too small... then it's not doing its job of sending to the C104 as soon as THRESHOLD conditi
SERV_DIFF = .5*interarrival_atm;           /*heuristic... for now*/
P1355_SEND_TIME = .00001*interarrival_atm;    /*heuristic... perhaps, should be in terms of ETH interarrival or something else...
t_1=0;

for(i=0;i<10;i=i+1){
mean_q[i]=0;
prev_q[i]=0;
tot_pkt[i]=0;
ethhigh_q[i]=0;
ethvar_q[i]=0;
}

prev_time=op_sim_time();


/*DEBUGGING*/
sum11=0;
no_of_insertions=0;
/*END OF DEBUGGING*/
```

exit execs **init**

```
op_intrpt_schedule_self(op_sim_time(),CODE0);

 /*op_sim_time()+inter_arrival_atm_time has been changed to as shown above*/
```

transition **init -> idle**

attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **stat**

attribute	*value*	*type*	*default value*
name	stat	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

...
...

enter execs **stat**

```
/*************************************************************/
/* there has been a change in the statistic wire backpressure*/
/* modify the stat_flag accordingly
/*************************************************************/
if (op_stat_local_read(STAT_WIRE0)==0) {stat_flag=0;}
else if (op_stat_local_read(STAT_WIRE0)==1) {stat_flag=1;}
```

transition **stat -> idle**

attribute	*value*	*type*	*default value*
name	tr_9	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **st_5**

attribute	*value*	*type*	*default value*
name	st_5	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **st_5**

```
for(i=0;i<10;i=i+1){
printf("(eth side t9000) mean_q[%d]=%lf ,ethset=%d\n",i,mean_q[i]/op_sim_time(),gugu);
printf("(ETH SIDE T9000) highest qsize[%d]=%1f,ethset=%d\n",i,ethhigh_q[i],gugu);
printf("(ETH SIDE T9000) variance of qsize[%d]=%1f,ethset=%d\n",i,(sqrt(ethvar_q[i]/op_sim_time())-(mean_q[i]/op_
}


printf("no of p1355 packets received from atm side by t9000 on eth side=%d\n",sum11);
printf("no of insertions=%d\n",no_of_insertions);

```

transition **st_5 -> idle**

attribute	*value*	*type*	*default value*
name	tr_15	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

B.1.3 Programming Code for the C104 (4-by-4 Case)

...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
DELAY	promoted	double	1E-06

Header Block

```
/*---------------------------------------------------------------------------------------
Name of Software        : c104_sw
Version                 : 1.0
Description/Purpose     : To manage the tasks and operations within the C104 module performed by the
                          Packet Layer Protocol according to the IEEE P1355 Heterogeneous Interconnect
                          Standard.
                          ____________________________________________________________

Name of Developer(s): Jangkyung Kim
Released to             : Institute for Systems Research (ISR)
                          University of Maryland at College Park
Date of Creation        : 07/19/95
Date of Revision        : N/A
By                      : N/A
Comments on Revision: N/A
Warnings                : None.
---------------------------------------------------------------------------------------*/
#define   MAX_NO_LINK   8 /* Maximum number of DS Link */
/*#define       DELAY   .000001*/ /* DELAY = 1 usec */
/*now got from user at runtime or from ef file*/
/* Define the types of Events that can occur on the States of this Switch Process */
#define PK_ARRVD     (op_intrpt_type() == OPC_INTRPT_STRM)     /* packet arrived */
#define   PK_SENT     (op_intrpt_type() == OPC_INTRPT_SELF)     /* packet is now ready for transmit */
#define END (op_intrpt_type()==OPC_INTRPT_ENDSIM)


/* End of Types of Events that can occur on the States of the Switch Process */
```

State Variable Block

```
/* Received Packets Counter */
int \rcvd_pkts;

/* Transmitted Packets Counter */
int \xmitd_pkts;

/* Destroyed Packets */
int \destr_pkts;

int \out;

/* input subq number for the round robin access */
int   \curr_q[MAX_NO_LINK];
double  \tot_packets;
double \DELAY;
int \origen2;
int \sum1;
```

…
…

Temporary Variable Block

```
   Packet *tmp_pkptr;                    /* the temporary packet pointer */
   Packet *pkptr;                        /* the received packet pointer  */
   Packet *out_pkptr;                    /* the packet to be transmitted */

 5 int count;                            /* To be used to count maximum number of links */
   int out_ds_link;                      /* output DS Link number */
   int out_subq;                         /* Output sub_queue number (from 0 to MAX_NO_LINK) */
   int dest_address;                     /* the destination output data link address where packet is to be routed */
   int curr_input_lnk;                   /* the current input link where packet came from */
10 double stamp_time1, stamp_time2;
   double     pk_transfer_time;
   int    pk_len;
```

Function Block

```
   /* This function writes the end_of_simulation traffic on the */
   /* output links and the ouput links throughput            */
   record_stats()
        {
 5          /* Record Final Statistics            */
            op_stat_scalar_write("Channel Traffic G",
                                 (double) xmitd_pkts/op_sim_time());
            op_stat_scalar_write("Channel Throughput S",
                                 (double) rcvd_pkts/op_sim_time());
10          printf("the final statistics are : %d, %d, %d\n", rcvd_pkts, xmitd_pkts, destr_pkts);

        }
```

forced state **Init**

attribute	*value*	*type*	*default value*
name	Init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Init**

```
   /* Initialize Accumulators */
   rcvd_pkts = 0;
   xmitd_pkts = 0;
   destr_pkts = 0;
 5
   stamp_time1 = 0.0;
   stamp_time2 = 0.0;

   for(count=0; count < MAX_NO_LINK; count++)
10 curr_q[count] = MAX_NO_LINK; /* First input queue index */
   tot_packets=0;


   op_ima_obj_attr_get(op_id_self(),"DELAY",&DELAY);
15

   sum1=0; /*debugging*/
```

...
...

transition **Init -> Idle**

attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **Idle**

attribute	*value*	*type*	*default value*
name	Idle	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

enter execs **Idle**

```
/*****************************************************************************/
/* Wait interrupt                                                            */
/* If packet arrived then go to receive state                                */
/* If packet is sent then go to the transmit state                           */
/*****************************************************************************/
```

exit execs **Idle**

transition **Idle -> Recv**

attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	PK_ARRVD	string	
executive		string	
color	RGB312	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> Xmit**

attribute	*value*	*type*	*default value*
name	tr_3	string	tr
condition	PK_SENT	string	
executive		string	
color	RGB330	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> Idle**

attribute	*value*	*type*	*default value*
name	tr_18	string	tr

condition	default	string	
executive		string	
color	RGB320	color	RGB333
drawing style	spline	toggle	spline

transition **Idle -> st_4**			
attribute	*value*	*type*	*default value*
name	tr_21	string	tr
condition	END	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **Recv**			
attribute	*value*	*type*	*default value*
name	Recv	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Recv**

```
    /* increment the count of received packets for the switch */
    ++rcvd_pkts;
    /* get the current input stream index where the packet is arriving from */
    curr_input_lnk = op_intrpt_strm();
 5  /*if(curr_input_lnk==4){printf("C104 GETS A P1355=%lf\n",op_sim_time());};
    */
    /* retrieve the packet from the input stream where the arrival occured */
    pkptr = op_pk_get(curr_input_lnk);
    /*FOLOOWING STATEMENT STAMPS THE ARRIVAL TO SEE IF T9000 IS A BOTTLE NECK*/
10  op_pk_nfd_set(pkptr,"POST_T9000_TIME",op_sim_time());
    /* Stamp the packet so that later we can extract the packet delay inside the switch */
    op_pk_stamp(pkptr);


15  /* get the output link destination address of the packet */
    op_pk_nfd_get(pkptr,"dest_address",&dest_address);
    /*THE NEXT STATEMENT HAS BEEN ADDED FOR DEBUGGING -sandeep*/
    /*if((curr_input_lnk>4)&&(curr_input_lnk<=7)&&(dest_address!=0)){
    printf("**********************************************************\n");
20  printf("PANIC PANIC PANIC PANIC PANIC PANIC PANIC\n");
    printf("**********************************************************\n");
    } */
    op_pk_nfd_get(pkptr,"ORIGEN",&origen2);
    if(origen2==0){sum1=sum1+1;}
25
    /*END OF DEBUGGING STUFF*/

    /* Determine if this output data link is busy or not */
    if (!op_subq_empty(dest_address))
30  { /* output link is busy(output subq is not empty), so save the packet in the input subq */
        op_subq_pk_insert(curr_input_lnk,pkptr,OPC_QPOS_TAIL);
        op_stat_local_write(0,1);
    }
```

```
      else
35    { /* output link is not busy (Output subq is empty)*/
      pk_len = op_pk_total_size_get (pkptr);
      pk_transfer_time = pk_len / 1000000000.0;
      op_intrpt_schedule_self(op_sim_time() + pk_transfer_time,dest_address);
      op_subq_pk_insert(dest_address,pkptr,OPC_QPOS_TAIL);
40    }
```

transition **Recv -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_12	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **Xmit**			
attribute	*value*	*type*	*default value*
name	Xmit	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **Xmit**

```
      /* If the P1355 packets are transmitted to the outlink of the C104 (i.e. if transmit timer expired), then  */
      /* transmit one p1355 packet from input subq to the output subq in round robin fashion.                   */

      out_subq = op_intrpt_code();
 5    out_ds_link = out_subq;
      out_pkptr = op_subq_pk_remove(out_subq,OPC_QPOS_HEAD);

      if (out_pkptr == OPC_NIL) {
            printf("recoverable error \n");
10    }
      else {
         if(out_ds_link==0)
        { tot_packets=tot_packets+1;}
         op_pk_send_delayed(out_pkptr,out_ds_link,DELAY);
15          /* The packet in the out_subq are sent to the output ds-link */
            xmitd_pkts++;
      }

      for(count=0; count<MAX_NO_LINK; count++, curr_q[out_ds_link]++){
20          if (curr_q[out_ds_link] >= (2 * MAX_NO_LINK))
                  curr_q[out_ds_link] = curr_q[out_ds_link] - MAX_NO_LINK;
                  /* Input queue index are from MAX_NO_LINK to 2 * MAX_NO_LINK-1 */

            if (!op_subq_empty(curr_q[out_ds_link])) /* If input subq has packet to send */
25          {
                  /* get the packet from the corresponding input queue index */
                  tmp_pkptr = op_subq_pk_access(curr_q[out_ds_link],OPC_QPOS_HEAD);
                  if (tmp_pkptr == OPC_NIL)
```

...
...

```
            {
30              printf("recoverable error \n");
            }
            else
            {
                /* get the output link destination address */
35              op_pk_nfd_get(tmp_pkptr, "dest_address", &dest_address);
                if ((dest_address == out_ds_link) && op_subq_empty (dest_address))
                { /* if packet is headed to the just emptied out_subq then, */
                  /* move p1355 packet from input_subq to out_subq */
                    out_pkptr = op_subq_pk_remove (curr_q[out_ds_link], OPC_QPOS_HEAD);
40                  op_stat_local_write(0,0);

                    /* send the packet through the output link */
                    pk_len = op_pk_total_size_get (out_pkptr);
                    pk_transfer_time = pk_len / 1000000000.0;
45                  op_intrpt_schedule_self(op_sim_time() + pk_transfer_time, dest_address);
                    op_subq_pk_insert(dest_address, out_pkptr, OPC_QPOS_TAIL);
                    break;
                } /* end of if */
            } /* end of else */
50      } /* end of if */
}/* end of for */

curr_q[out_ds_link] = curr_q[out_ds_link] + 1;              /* Increase the current queue index by 1 for the round robin */

55 /*---------------------------------------------------------------------------------------------*/
```

transition **Xmit -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **st_4**			
attribute	*value*	*type*	*default value*
name	st_4	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **st_4**

```
printf("total no of packets=%lf \n",tot_packets);
printf("sum1=%d\n",sum1);
```

...
...

transition **st_4 -> Idle**			
attribute	*value*	*type*	*default value*
name	tr_22	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

B.2 Programming Code for the LRD-based Traffic Generator at the ATM Port

...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
source_time_step	promoted	double	0.0
alpha	promoted	double	0.0
lambda	promoted	double	0.0
mul_fact	promoted	integer	0

Header Block

```
#include <stdio.h>
#include <math.h>
#define  NEXT op_intrpt_code()==0
#define SIZE 1000000
#define END (op_intrpt_type()==OPC_INTRPT_ENDSIM)

/*#define ALPHA 2.5
#define PP 2000*/
long parito();
```

State Variable Block

```
long \server[SIZE];
int \new_arrival;
int \max_len;
int \dec;
int \no;
Distribution* \pois;
double \time_step;
double \PP;
double \ALPHA;
int  \mf;
int \burst_fact;
double \source_bits;
```

Temporary Variable Block

```
long  i,j;
```

Function Block

```
long parito( )
{
  float y,z;
  long res;
  y=op_dist_uniform(1);
 z= pow(1-y,1/(1-ALPHA));
  res=(long)z;
  return(res);}
```

forced state **st_0**

attribute	*value*	*type*	*default value*
name	st_0	string	st

enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **st_0**

```
printf("YEH BUDDY I'M IN LRD\n");

 max_len=0;
 no=0;
5
for(i=0;i<SIZE;i=i+1){
server[i]=0;}
op_ima_obj_attr_get(op_id_self(),"source_time_step",&time_step);
op_ima_obj_attr_get(op_id_self(),"alpha",&ALPHA);
10 op_ima_obj_attr_get(op_id_self(),"lambda",&PP);
op_ima_obj_attr_get(op_id_self(),"mul_fact",&mf);
/*op_ima_obj_attr_get(op_id_self(),"burst_fact",&burst_fact);*/
pois=op_dist_load("poisson",PP,1);
/*printf("alpha=%lf\n",ALPHA);*/
15 /*printf("lambda=%lf\n",PP);*/
source_bits=0;
burst_fact=1;
mf=mf/(burst_fact);
time_step=time_step/(burst_fact);
20 /* printf("time_Step=%lf\n",time_step);*/
```

transition **st_0 -> st_1**

attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **st_1**

attribute	*value*	*type*	*default value*
name	st_1	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

enter execs **st_1**

```
new_arrival=op_dist_outcome(pois);
no=no+new_arrival;
/*printf("time=%lf\n",op_sim_time());   */
dec=0;
5 for(i=0;i<max_len;i=i+1){
if(server[i]!=0){
server[i]=server[i]-1;
if(server[i]==0){dec=dec+1;}
```

...
...

```
 }
}
i=0;
while((new_arrival!=0)&(i<SIZE)){
if(server[i]==0){
new_arrival=new_arrival-1;
server[i]=parito();}
i=i+1;
}

if(i>max_len) max_len=i;
no=no-dec;
/*printf("lrd sending %d\n", no*mf);*/
op_pk_send(op_pk_create(no*mf),0);
source_bits=source_bits+no*mf;
/*printf("ONE PACKET SENT no*mf= %d\n",no*mf);*/
op_intrpt_schedule_self(op_sim_time()+time_step,0);

op_stat_local_write(1,no*mf);
op_stat_local_write(2,max_len);
op_stat_local_write(3,dec);
```

transition **st_1 -> st_1**			
attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	NEXT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

B.3 Programming Code for the SRD-based Traffic Generator at the ATM Port

...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
no_of_states	promoted	integer	0
packet_mult	promoted	integer	0
discrete_unit	promoted	double	0.0
frame_time	promoted	double	0.0
atm_low	promoted	integer	0
atm_high	promoted	integer	0
origen	promoted	integer	0

Header Block

```
#define NEXT_ARRIVAL op_intrpt_code()==0
#define CHANGE_STATE op_intrpt_code()==1
#define TRANSITION  "transition"
/*#define N 3 no. of states in markov*/
#define OUTSTREAM 0     /*where to send packets*/
/* #define pack_size 10 */
#define RATES "rates"

/* commented out stuff has been promoted */
```

State Variable Block

```
/* List*  /list_ptr, /list_ptr2;   */

double \da[10][10];
int \st;
double \markov_time, \arr_time;
int \df;
double \rn;
double \tm[10][10];
double \rate_poi[10];
int  \ch_flag ;
Distribution *\exp_markov[10];
Distribution *\pois[10];
Distribution *\addr_ptr;
int \N;
int \pack_size;
int \pack_mult;
int \origen,\low,\high;
FILE *\fp1;
FILE *\fp2;
FILE *\fp3;
int \no_of_times;
double \residual_time, \time_step,\time_stamp;
```

```
int \no_sent;
double \frame_time;
```

Temporary Variable Block

```
List* list_ptr;
List* list_ptr2;
double sum;
double temp;
int j;
int i;
 int sw;
double testw,no_of_arrival;
Packet* pkt;
Packet* cp_pkt;
int address;
```

forced state **init**

attribute	*value*	*type*	*default value*
name	init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **init**

```
/*HERE ALL THE INPUT PARAMETERS ARE TAKEN*/
/*packet_mult=multiplication factor to (effectively)increase size of packets*/
/*discrete_unit = lenght of discretiastion*/

/*no_of_states=no_of_states of the markov chain*/
op_ima_obj_attr_get(op_id_self(),"no_of_states",&N);
op_ima_obj_attr_get(op_id_self(),"packet_mult",&pack_mult);
op_ima_obj_attr_get(op_id_self(),"discrete_unit",&time_step);
op_ima_obj_attr_get(op_id_self(),"atm_low",&low);
op_ima_obj_attr_get(op_id_self(),"atm_high",&high);
/*low and high define the range of the addresses generated*/
op_ima_obj_attr_get(op_id_self(),"origen",&origen);
addr_ptr=op_dist_load("uniform integer",low,high+1);
/*op_ima_obj_attr_get(op_id_self(),"frame_time",&frame_time); */
/*transition.gdf and rates.gdf store the (modified transition matrix) and the rates of the
poison processess in each state*/
fp1=fopen("transition_atm.gdf","r");
fp2=fopen("rates_atm.gdf","r");
fp3=fopen("checkwala","w");
 printf("SIMULATION BEGINS42=%d,%d \n",N,pack_size);
/*list_ptr=op_prg_gdf_read(TRANSITION); */
for(i=0;i<N;++i){
for(j=0;j<N;++j){
fscanf(fp1,"%lf\n",&temp);tm[i][j]=temp;
printf("tm[%d][%d]=%lf,%d\n",i,j,tm[i][j],N*i+j);}
exp_markov[i]=op_dist_load("exponential",(1/tm[i][i]),5.0);}

for(sw=0;sw<100;sw=sw+1){
testw=op_dist_outcome(exp_markov[1]);
```

...
...

```
/*printf("testw=%lf\n",testw);*/
}

/*now load parameters for distribution functions*/
/*load rates of state transition */

/*load rates in each state*/
/*list_ptr2=op_prg_gdf_read(RATES);*/
for(i=0;i<N;++i){
fscanf(fp2,"%lf\n",&rate_poi[i]);
/*printf("rate_poi=%lf\n" , rate_poi[i]);*/
pois[i]=op_dist_load("poisson",(rate_poi[i]*time_step),5.0);
}

for(sw=0;sw<100;sw=sw+1){
testw=op_dist_outcome(pois[0]);
/*printf("testw2=%lf\n",testw);*/
testw=op_dist_outcome(pois[1]);
/*printf("testw3=%lf\n",testw);*/ }




for(i=0;i<N;i=i+1 ){
da[i][i]=0;
}


for(i=0;i<N;i=i+1){
sum=0;
for(j=0;j<N;j=j+1){
if(j!=i){ sum=da[i][j]=tm[i][j]+sum;}  }
for(j=0;j<N;j=j+1){da[i][j]=da[i][j]/sum;}
}

for(i=0;i<N;i=i+1){
for(j=0;j<N;j=j+1){
printf("%lf\n", da[i][j]);}}

 st=0;
 /*initialise*/
markov_time=op_dist_outcome(exp_markov[st]); /*(to be filled)*/
no_of_times=(int)(markov_time/time_step);
no_sent=0;

pack_size=1;
time_stamp=0;

printf("im in initialisation\n");
```

transition **init -> arrival**			
attribute	*value*	*type*	*default value*
name	tr_3	string	tr
condition		string	

executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **arrival**			
attribute	*value*	*type*	*default value*
name	arrival	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(See below.)	textlist	(See below.)
status	unforced	toggle	unforced

enter execs **arrival**

```
/*IN THIS PART THE POISSSON PROCESS GIVES OUT BITS CORRESPONDING TO THE RATE OF THE STATE THE MARKO

/* printf("hey i'm in state1\n");*/

5

no_sent=no_sent+1; /*number of times this routine has been entered for this particular soujourn time)*/
no_of_arrival=op_dist_outcome(pois[st]); /*calculates # of arrivals for current time interval*/

/*printf("st,no of arrivals=%d ,%lf,%lf\n",st,op_dist_outcome(pois[st]),no_of_arrival);*/
10 /*HERE WE CREATE AN ATM PACKET COPIES OF WHICH ARE SENT */
pkt=op_pk_create_fmt("ATM_token");
op_pk_nfd_set(pkt,"CREATE_TIME",op_sim_time());
address=op_dist_outcome(addr_ptr);
op_pk_nfd_set(pkt,"dest_address",address);
15 op_pk_nfd_set(pkt,"ORIGEN",origen);


/*END OF PACKET FORMAT CREATION*/
/* This section was modified by Giles Charleston due to errors encountered */
20 for(j=0;j<no_of_arrival;j=j+1)
{
cp_pkt = op_pk_copy(pkt);
op_pk_send(cp_pkt,0);
}               /* forgot to destroy packet in orginal version */
25 op_pk_destroy(pkt); /* so added this line to assure packet is destroyed */
/*********************************************************/
/*printf("aithappa 1\n");*/

if(no_sent<no_of_times){   /*still remain in this  state*/
30 /*printf("NO switching,no_sent=%d\n",no_sent);*/
op_intrpt_schedule_self(op_sim_time()+time_step,0);
}
else
{
35 /*printf("YES switching\n");*/
/*switch states*/
op_intrpt_schedule_self(op_sim_time()+time_step,1);
}
/*printf("3\n");*/
40

op_stat_local_write(1,no_of_arrival);
```

...
...

exit execs **arrival**

transition **arrival -> markov_set**

attribute	*value*	*type*	*default value*
name	tr	string	tr
condition	CHANGE_STATE	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arrival -> arrival**

attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition	NEXT_ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **markov_set**

attribute	*value*	*type*	*default value*
name	markov_set	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **markov_set**

```
/*THIS ROUTINE IS ENTERED WHENEVER THERE IS A CHANGE OF STATE
THIS ROUTINE DETERMINES WHAT THE NEXT STATE SHOULD BE AND HOW LONG IT SHOULD STAY THERE*/
/*printf("i'm in this state(changing state)\n");*/
df=0;
i=0;
rn=op_dist_uniform(1);
while((df==0)&&(i<N)){
if(rn<da[st][i]){df=1;st=i;}
i=i+1;}

markov_time=op_dist_outcome(exp_markov[st]); /*generate sojourn time*/
/*printf("markov_time,st=%lf,%d\n",markov_time,st);*/
no_of_times=(int)(markov_time/time_step);
/*printf("no_of_times=%d\n",no_of_times);*/
no_sent=0;
/*residual_time=markov_time-(no_of_times*timestep);*/
op_stat_local_write(0,st);

/*fprintf(fp3,"%d %lf %d\n",st,markov_time,no_of_times);*/
```

...
...

transition **markov_set -> arrival**			
attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

Appendix C

Acronyms/Abbreviations

This appendix provides a compilation of several abbreviations/acronyms used throughout the thesis and also some common ones generally employed in the industry and academia.

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ALAX	ATM LAN Access Switch based on the IEEE1355 (P1355) Serial Bus Standard
ANSI	American National Standards Institute
ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency NETwork
ATM	Asynchronous Transfer Mode
AT&T	American Telephone & Telegraph
AUI	Attachment Unit Interface
B-ISDN	Broadband Integrated Services Digital Network
BER	Bit Error Rate
BIOS	Basic Input/Output System
BRI	Basic Rate Interface
BUS	Broadcast and Unknown Server
CBR	Constant Bit Rate
CCITT	Consultative Committee on International Telephony/Telegraphy
CES	Circuit Emulation Service
CIF	Cells In Frame
CODEC	CODer-DECoder
CNET	Centre National des Etudes de Telecommunications (French) National Center for Studies in Telecommunications
COTS	Commercial Off The Shelf
CPCS	Common Part Convergence Sublayer
CPE	Customer Premises Equipment
CPU	Central Processing Unit

CS	Convergence Sublayer
CSMA	Carrier Sense Multiple Access
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSU	Channel Service Unit
DCE	Data Communication Equipment
DQDB	Distributed Queue Dual Bus
DS	Data Strobe
DSL	Digital Subscriber Line
DSU	Data Service Unit
DTE	Data Terminal Equipment
EIA	Electronics Industry Association
ELAN	Emulated LAN
EOF	End Of File
EOM	End Of Message
ESPRIT	European Strategic Program for Research and Development in Information Technology
ETRI	Electronics and Telecommunications Research Institute (a Korean research entity)
FCFS	First Come First Served
FCT	Flow Control Token
FDDI	Fiber Distributed Data Interface
FIFO	First-In-First-Out
FO	Fiber Optic
FPU	Front Processor Unit
FSM	Finite State Machine
FTP	File Transfer Protocol
FORTRAN	FORmula TRANslation

GUI	Graphical User Interface
HDLC	High Level Data Link Control
HIC	Heterogeneous Inter-Connect
HOL	Head Of the Line
HOQ	Head of the Queue
HS	High Speed
IAB	Internet Architecture Board
IC	Input Controller
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Standards Organization
ISDN	Integrated Services Digital Network (Narrowband ISDN)
ISR	Institute for Systems Research
ITU	International Telecommunications Union (formerly known as CCITT)
LALUT	Local Addressing Look Up Table
LAN	Local Area Network
LANCE	LAN Circuit Emulation
LANE	Local Area Network Emulation
LAST	Laboratory for Advanced Switching Technology
LEC	LAN Emulation Client
LECS	LAN Emulation Configuration Server
LES	LAN Emulation Service (or Server)
LLC	Logical Link Control
LPFS	Longest Packet First Served
LRD	Long Range Dependent

MAC Media Access Control

MARS Multicast Address Resolution

MAU Media Attachment Unit

MML MAC Mapping Layer

MMPP Markov Modulated Poisson Process

MPOA Multiple Protocol Over ATM

MPEG Motion Picture Experts Group

NDIS Network Driver Interface Specification

NHRP Next Hop Resolution Protocol

NIC Network Interface Card

NNI Network-to-Network Interface

OC Output Controller

ODL Optical Data Link

OMI Open Microprocessor Systems Initiative

OPNET OPtimized Network Engineering Tools

OSI Open Systems Interconnect

PBX Private Branch Exchange

PC Personal Computer

PCI Peripheral Component Interconnect

PDU Protocol Data Unit

PEC Protocol Engineering Center (within ETRI)

POTS Plain Old Telephone Service

PPP Point to Point Protocol

PRI Primary Rate Interface

PSTN Public Switched Telephone Network

PVC Permanent Virtual Circuit/Connection

QoS Quality of Service

RACE Research for Advanced Communication in Europe

RFC Request For Comment

RR Round Robin

RISC Reduced Instruction Set Computing

SAR Segmentation And Reassembly

SDLC Synchronous Data Link Control

SIMAN SIMulation ANalysis

SLIP Serial Line Internet Protocol (IP)

SMDS Switched Multimegabit Data Service

SNA Systems Network Architecture

SNMP Simple Network Management Protocol

SONET Synchronous Optical Network

SRD Short Range Dependent

SS7 Signaling System # 7

STP Shielded Twisted Pair

SVC Switched Virtual Circuit/Connection

TAXI Transparent Asynchronous Transmitter/Receiver Interface

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol/Internet Protocol

TCS Transmission Convergence Sublayer

TDM Time Division Multiplexing

UBR Unspecified Bit Rate

UNI User-to-Network Interface

UTP Unshielded Twisted Pair

VBR Variable Bit Rate

VC Virtual Channel

VCC Virtual Channel Connection

VCI Virtual Channel Identifier

VCP Virtual Channel Processor

VLAN Virtual LAN

VLSI Very Large Scale Integration

VP Virtual Path

VPC Virtual Path Connection

VPI Virtual Path Identifier

WAN Wide Area Network

WWW World Wide Web

Appendix D

Glossary of Some Selected Technical Terms

Address Resolution Protocol (ARP) - A protocol used to dynamically discover the low-level physical network hardware address that corresponds to the higher level protocol address for a given host. ARP is limited to physical network systems that support broadcast packets that can be heard by all hosts on the network.

ARPANET - A pioneering long haul network funded by ARPA (Advanced Research Projects Agency) later named DARPA (Defense Advanced & Research Projects Agency) and built by BBN. it served from 1969 through 1990 as the basis for early networking research and as a central backbone during the development of the Internet. The ARPANET consisted of individual packet switching nodes interconnected by leased lines.

Attachment Unit Interface (AUI) - An IEEE 802.3 cable connecting the media attachment unit (MAU) to the network device. The term AUI also can be used to refer to the host back-panel connector to which an AUI transceiver might attach.

Backbone Network - Any network that forms the central interconnect for an internetwork. A national backbone is a WAN; a corporate backbone can be a LAN.

Backplane - The main bus that carries data within a device.

Benchmark - A point of reference from which measurements can be made; involves the use of typical problems for comparing performance and is often used in determining which system can best serve a particular application.

Bridge - An intermediate system (i.e. computer) that connects two or more LANs which use identical LAN protocols. The bridge acts as an address filter, picking up packets from one LAN that are intended for a destination on an other LAN and passing those packets on. The bridge does not modify the contents of the packets and does not add anything to the packet. The bridge operates at layer 2 of the OSI protocol stack.

Bridging - Techniques for interconnecting two LAN segments that utilize the same LLC procedures but may use the same or different MAC procedures.

Bus - A set of signals defined by an interface system that connect to devices for the purpose of transferring data.

Bus Specification - The definition of a bus. In many cases, the bus specification also includes the definition of bus module and enclosure mechanical requirements.

Cell - A small fixed-size frame used on ATM networks for instance. Each ATM cell contains 48 octets of data and 5 octets of header.

Collapsed Backbone - A non-distributed backbone where all all network segments are interconnected via an interconnecting device. A collapsed backbone may be a virtual network segment existing in a device such as hub, a router or a switch.

Concentrator - Device that serves a wiring hub in a star-topology network. Sometimes it refers to a device containing multiple modules of network equipment.

CSMA/CD - A characteristic of multiple access protocols that operates by allowing multiple stations to contend for access to a transmission medium by listening to see if the medium is idle, and a mechanism that allows the hardware to detect when two stations simultaneously attempt transmission (e.g. Ethernet uses CSMA/CD).

DS3 - A telephony classification of speed for leased lines equivalent to approximately 45 Mbps.

Emulation - The imitation of a computer system, performed by a combination of hardware and software, that allows programs to run on otherwise incompatible systems.

Enterprise Network - A geographically dispersed network under the auspices of one organization.

Ethernet - A baseband LAN specification invented by Xerox Corporation and developed jointly by Xerox, Intel, and Digital Equipment Corporation. Ethernet networks operate at 10 Mbps using CSMA/CD to run over coaxial cable. Ethernet is similar to a series of standards produced by IEEE referred to as IEEE 802.3 .

Finite State Machine (FSM) - A way of describing a network module as a set of states that, based on input and configuration, perform transitions to other states.

Firmware - Permanent or semi-permanent micro-instruction control for a user-oriented function.

Gateway - Originally, researchers used the term gateway for dedicated computers that route packets; vendors have adopted the term router. Gateway now refers to an application program that interconnects two services (e.g. an e-mail gateway).

Gbps - (Giga Bits Per Second) A measure of the rate of data transmission.

Hot Swap - Meaning live insertion. Installation and removal of boards without turning off the system power.

Hub - A combination of concentrators, repeaters and possibly other functions (such as bridging) combined into a single chassis is a hub.

In-band - A method of sending management and/or control signals on the same frequency (in the same band) as data signals.

Internet - Physically, a collection of packet switching networks interconnected by bridges and/or routers along with inter networking protocols that allow them to function logically as a single, large, virtual

network. When written in upper-case, Internet refers specifically to the global Internet.

Internetworking - The connection or interconnection of local and remote networks through intermediate systems such as bridges and routers.

Interoperability - The ability of software and hardware on multiple machines from multiple vendors to communicate meaningfully. This term best describes the goal of inter networking, namely to define an abstract, hardware independent networking environment that makes it possible to build distributed computations that interact at the network transport level without knowing the details of underlying technologies.

Kbps - (Kilo Bits Per Second) A measure of the rate of data transmission.

Kernel - A term used in operating systems. The kernel shields the low-level functioning of the operating system from high-level interfaces, such as user shells.

Latency - The delay between the time a device receives a packet and the packet is forwarded out of the destination port.

Mbps - (Millions of Bits Per Second) A measure of the rate of data transmission.

Multicast - A technique that allows copies of a single packet to be passed to a selected subset of all possible destinations.

Network Interface Card (NIC) - The circuit board or other hardware that provides the interface between a communicating DTE and the network.

OC3 - A bit rate of approximately 155 million bits per second used over fiber optic connections.

OSI - (Open Systems Interconnect) A reference to protocols, specifically ISO standards, for the interconnection of cooperative computer systems.

Out-of-band - Any frequency separate from the band being used for data, voice or video traffic. This typically requires a completely separate signal path or wire.

Packet - Used loosely to refer to any small block of data sent across a packet switching network.

Protocol - A formal description of message formats and the rules two or more machines must follow to exchange those messages. These rules or procedures are commonly agreed upon by committees such as IEEE and ANSI.

Repeater - A hardware device that extends a LAN. A repeater regenerates and propagates electrical signals from one physical network to another. A repeater rebroadcasts a signal to prevent its degradation.

Router - A special purpose intermediate system, specifically dedicated computer, used to connect two or more networks that may or may not be similar. The router employs an internet protocol present in each router and each host (station) of the network. The router operates at layer 3 of the OSI model. The router usually has more function for traffic management of large networks. It performs functions beyond those of a bridge including the determination of optimal network routes to minimize network traffic, provide congestion control and filter broadcast frames.

Simple Network Management Protocol (SNMP) - The Internet standard protocol for managing nodes on an IP network.

Topology - The physical arrangement or shape of a network.

Tunneling - Carrying protocol A within protocol B packets such that B treats A as though it were a higher level protocol. It is used to get data between network segments that use a protocol that is not supported by the internet.

Twisted Pair - Two insulated wires twisted together so that each wire faces the same amount of electromagnetic and radio-frequency interference from the environment. It consists of two 18 to 24 American Wire Gauge (AWG) solid copper strands.

REFERENCES

[1] Ahmadi, H., et al., "A High-Performance Switch Fabric for Integrated Circuit and Packet Switching," *Proc. IEEE Infocom,* pp. Page 9-18, New Orleans, March 1988.

[2] Alles, Anthony, "ATM Internetworking," Research Report, Cisco Systems Inc., May 1995.

[3] Anido, G. J. and Seeto, A. W., "Multipath Interconnection: A Technique for Reducing Congestion Within Fast Packet Switching Fabrics," *IEEE J. Select. Areas in Communication,* Vol. 6, No. 9, pp. 1480-88, December 1988.

[4] Barri, P. and Goubert, J. A. O., "Implementation of a 16 x 16 Switching Element for ATM Exchanges," *IEEE J. Select. Areas in Communication,* Vol. 9, No. 5, pp. 751-57, June 1991.

[5] Berri, Noemi, "Asynchronous Transfer Mode (ATM) Switch Technology and Vendor Survey," Research Report NAS-95-001, NAS Systems Division, NASA Ames Research Center, January 3, 1995.

[6] Bertsekas, Dimitri and Gallager, Robert, *Data Networks* . Upper Saddle River, New Jersey: Prentice Hall Inc., 1992.

[7] Cohen, M. E. and Abensour, D. S., "An ATM Strategy for IBM Networking Systems," *IBM Systems Journal,* Vol. 34, pp. 554-563, 1995.

[8] Christensen, K. J. et al., "Local Area Networks- Evolving from shared to switched access," *IBM Systems Journal,* Vol. 34, pp. 347-374, 1995.

[9] De Prycker, Martin, *Asynchronous Transfer Mode: Solution for Broadband ISDN.* New Jersey: Prentice Hall, 1995.

[10] De Prycker, M. and De Somer, M., "Performance of an Independent Switching Network with Distributed Control," *IEEE J. Select. Areas in Communication,* SAC-5, No. 8, pp. 1293-1301, October 1987.

[11] Devault, M. et al., "The 'Prelude' ATD Experiment: Assessments and Future Prospects," *IEEE J. Select. Areas in Communication,* Vol. 6, No. 9, pp. 1528-36, December 1988.

[12] Goke, L. R. and Lipovski, G. J., "Banyan Networks for Partitioning Multiprocessor Systems," *Proc. First Annual Symp. Computer Architecture,* pp. Page 21-28, December 1973.

[13] IBM Technical White Paper, "Local Area Network Directions: IBM Solutions,", 1994-1995.

[14] IEEE, IEEE1355, "Standard for Heterogeneous Interconnect (HIC) (Low Cost Low Latency Scalable Serial Interconnect for Parallel System Construction)," January 1995.

[15] IEEE Communications Society, *Performance Evaluation of High Speed Switching Fabrics and Networks.* New York, NY: IEEE Press, 1993.

[16] Inmos SGS-Thomson Microelectronics, *ST C104 Asynchronous Packet Switch, Preliminary Data Sheet.* United Kingdom: June 1994.

[17] Jain, Raj, *The Art of Computer Systems Performance Analysis.* New York, New York: John Wiley and Sons Inc., 1989.

[18] Karol, M. J. and Hluchyj, M. G., "Queuing in High-Performance Packet Switching," *IEEE J. Select. Areas in Communication,* Vol. 6, No. 9, pp. 1587-97, December 1988.

[19] Karol, M. J. et al., "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. Communication,* COM-35, No. 12, pp. 1347-56, December 1987.

[20] Keiser, Gerd E., *Local Area Networks.* New York, New York: Mc Graw-Hill Book Company, 1989.

[21] Kesselring, W. David, "An Overview of Interconnecting LANs over ATM," IBM Unclassified Document Number TR29.1931, Networking Systems Hardware Division, 1994-1995.

[22] Kim, J. and Ryu, M. G., "Design of an ATM LAN Access Switch Based on IEEE P1355," Technical Research Report ISR TR95-114, Laboratory for Advanced Switching Technologies - Institute for Systems Research, University of Maryland, College Park, MD, 1995.

[23] Kim, J. and Ryu, M. G., "ATM LAN Access Switch (ALAX): Protocol Architecture (LAN Emulation Version)," Technical Research Report ISR TR95-112, Laboratory for Advanced Switching Technologies - Institute for Systems Research, University of Maryland, College Park,MD, 1995.

[24] Kim, J. et al., "OPNET Simulation Model of the ALAX" Technical Research Report ISR TR 95-113, Laboratory for Advanced Switching Technologies - Institute for Systems Research, University of Maryland, College Park,MD, 1995.

[25] Kim, J. et al., "IEEE 1355-Based Architecture for an ATM Switch- A Case for Onboard Switching and Processing," *American Institute of Aeronautics and Astronautics,* Vol. 25, pp. 451-457, April 1996.

[26] Kim, J. et al., "ALAX- A P1355-Based Architecture for an An ATM LAN Access Switch, with Application to ATM Onboard Switching," Technical Research Report ISR TR 95-115, Laboratory for Advanced Switching Technologies - Institute for Systems Research, University of Maryland, College Park,MD, 1995.

[27] Luderer, W. R. and Knauer, S. C., "The Evolution of Space Division Packet Switches," *Proc. Int'l. Switching Symp.*, Vol. 5, pp. 211-16, Stockholm, May 1990.

[28] May, M. D. et al.- Inmos SGS-Thomson Microelectronics, *Networks, Routers and Transputers: Function, Performance and Application.*UK: IOS Press, 1993.

[29] Mc Quillan, J., "The Present and Future of LAN-in-a-box," *Business Communications Review,* Vol. 21, No. 8, pp. 12-14, August 1991.

[30] Newman, Peter , "ATM Local Area Networks" Technical Research Report- Adaptive Corp., Redwood City, California. March 1993.

[31] Newman, Peter, "ATM Technology for Corporate Networks," *IEEE Communications Magazine,* Vol. 5, pp. 90-101, April 1992.

[32] Newman, Peter, "A Fast Packet Switch for the Integrated Services Backbone Network," *IEEE J. Select. Areas in Communication,* Vol. 6, No.9, pp. 1468-79, December 1988.

[33] Oie, Y. et al., "Survey of the Performance of Non-Blocking Switches with FIFO Input Buffers," *Proc. IEEE Int'l Conf. Communication,* (ICC '90) Vol. 2, pp. 737-41, April 1990.

[34] *OPNET Manuals.* Washington, D.C.: MIL3, 1994.

[35] Rao, Sandeep , "Traffic Models for the OPNET Simulator of the ALAX" Technical Research Report ISR TR 95-117, Laboratory for Advanced Switching Technolo-

gies - Institute for Systems Research, University of Maryland, College Park,MD, 1995.

[36] Rao, Sandeep , "OPNET Simulator of the ALAX: Preliminary Results" Technical Research Report ISR TR 95-118, Laboratory for Advanced Switching Technologies - Institute for Systems Research, University of Maryland, College Park,MD, 1995.

[37] Sanders, Mark and Mc Cormick, Ernest, *Human Factors in Engineering and Design.* New York, New York: Mc Graw-Hill Inc, 1993.

[38] Shneiderman, Ben, *Designing the User Interface- Strategies for Effective Human Computer Interaction.* New York, New York: Addison-Wesley Publishing Company, 1992.

[39] Simoneau, Paul, *Guide to Networking & Internetworking Terms.* Cary, North Carolina: American Research Group Inc., 1993.

[40] Stallings, William, *Data and Computer Communications.* Englewood Cliffs, New Jersey: Prentice Hall, 1994.

[41] Tanenbaum, Andrew, *Computer Networks.* Englewood Cliffs, New Jersey: Prentice Hall, 1989.

[42] Tobagi, F. A., "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," *Proc. IEEE,* Vol. 78, No. 1, pp. 133-67, January 1990.

[43] Tolly, Kevin et al., "Hardcore ATM Switching," *CommunicationsWeek,* pp. 59-73, August 26 1996.

[44] The ATM Report, "ATM Industry Survey," Vol. 3, No. 10, December 30, 1995. Rockville, MD: Broadband Publishing Corporation.

[45] The ATM Report, "An ATM Forum Progress Report," Vol. 3, No. 12, March 8, 1996. Rockville, MD: Broadband Publishing Corporation.

[46] Wang, W. and Tobagi, F. A., "The Cristmas Tree: An Output Queuing Space-Division Fast Packet Switch,' *Proc. IEEE Infocom,* Vol. 1, pp. 163-70, April 1991.