INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Performance Evaluation of Multimedia Satellite Communications Systems Using On-board Packet Switches

Tien Hy Bui

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montreal, Quebec, Canada

February 1998

©Tien Hy Bui, 1998

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisitions et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-39978-8

Canada

ABSTRACT

Performance Evaluation of Multimedia Satellite Communications Systems Using On-board Packet Switches

Tien Hy Bui

Satellite communications systems are ideally equipped to provide future multimedia services on a global level and at a distance-independent cost. In order to meet the ever-growing bandwidth demands of these new applications, satellites employing multiple spot beam antennas are required. An on-board packet switch is an essential element in such a system as it offers full connectivity among users and offers efficient utilization of the space segment. This thesis evaluates the performance of a satellite-switched system in a multimedia environment mainly composed of voice, video, and data sources. Aggregate voice or video traffic is modeled as a 2-state Markov Modulated Poisson Process (MMPP) while two models for aggregate data traffic, MMPP and Pareto Modulated Poisson Process (PMPP) are used to examine the effects of traffic burstiness and long-range dependent behaviour. Multiple Frequency Time Division Multiple Access (MF-TDMA) is utilized on the uplink in conjunction with a dynamic capacity allocation scheme. Higher priority is given to voice and video real-time traffic to avoid delay variation. On-board downlink queue is provided for data jitter-tolerant traffic to achieve high statistical multiplexing gain.

Simulation results show that the system can support traffic predominently composed of real-time applications. As jitter-tolerant data becomes the dominant traffic component and becomes highly correlated, the size of the uplink and downlink queues need to be increased to maintain an acceptable quality-of-service (QoS). The packet loss due to the Knockout contention scheme is much lower than that due to the limited capacity on the uplink and downlink. This makes the Knockout switch fabric attractive for on-board switching since it achieves low complexity. The need for a congestion control scheme that can shape the traffic is required, especially when the traffic has long-range dependent behaviour.

Keywords: Packet Switching, Satellite Communications, Multimedia Traffic

To the memory of my grand-parents

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my thesis supervisor, Prof. Tho Le-Ngoc for his invaluable guidance and support without which it would have been impossible to complete this thesis.

I gratefully acknowledge the role of the Canadian Institute for Telecommunications Research (CITR) and Spar Aerospace Limited in funding this project.

I would like to thank my colleagues and friends at the National Centre of Excellence in Telecommunications (Concordia University), especially Dr. P. Tsingotjidis, R. DiGirolamo, T.V.J. Ganesh Babu, S. Shah-Heydari, H.R. Mehrvar for their continuous encouragement and support.

My special thanks to my parents, brother and extended family members who have always been by my side through good times and bad.

I am also in debt to my fiancee and future wife, Thanh Tam, who had to endure all the possible range of emotions I have gone through.

Finally, I would also like to extend my gratitude to faculty and staff members of the department of Electrical and Computer Engineering, Concordia University.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Background

Multimedia applications will revolutionize every aspect of our daily lives. Be it at home, school or business, it will be an integral part of our everyday activities. In the near future, we will be able to reach out to the world directly from the comfort of our home. In fact, with the simple click of a button, a multitude of services such as videophony, video-on-demand, high definition television (HDTV), high-resolution imaging, and home-banking will be available at our fingertips. Moreover, thanks to the breakthroughs in electronics and digital communications, these services will be available and in an integrated manner. However, the amalgam of new services (multimedia traffic) will require larger bandwidths and higher bit rates to support the resulting traffic through broadband networks [1, 2].

In order to rapidly deploy these broadband services over vast regions, especially in remote and rural areas, a wireless communications network is needed. A wireless network (e.g., cellular-based or satellite-based) not only provides wide area coverage, but also liberates the user from the tether. This in turn enables the possibility of mobile communications and a universal personal telecommunication (UPT) approach [3].

When we consider the available wireless options [4], a satellite-based system is the only one capable of truly providing global services at a distance-insensitive cost, in addition to supplying all the advantages of a wireless system.

A satellite system is composed of two parts: (i) a space segment, comprising the satellite spacecraft in orbit, and (ii) a ground segment that includes all the earth-stations. The end-user can either be connected directly to the earth-station or indirectly through the auspice of an existing terrestrial network (e.g., telephone network, cable system, dedicated lines, etc.). End-user traffic is sent to the earth-station, where it is processed and transmitted to the satellite by modulating a radio frequency carrier. Traditionally, the satellite received all the earth-station radio frequency carriers in its uplink (earth-to-satellite) frequency spectrum, amplified these carriers, and re-transmitted them back to earth in a different downlink (satellite-to-earth) frequency spectrum in order to avoid interference. In essence, the function of the satellite was simply that of a large repeater in space, providing mainly overseas telephone trunks and broadcasts of television programs [5, 6, 7].

One of the major drawbacks of early generation satellites was the need for large and expensive earth-stations which limited the penetration of satellite-based services mainly to the business sector. As satellite and space technology improved, more powerful satellites with larger antennas and longer life spans were deployed. This reduced the channel cost, as well as the size and cost of earth-stations. The satellite system steadily became an economical means of communications, readily available to a greater number of users.

With the growing satellite user population, a need to increase total system capacity was felt. An efficient way to sustain such a capacity increase is to use multiple spot beams in conjunction with a reuse of their frequency allotment. Theoretically, if one large beam covering a given area can provide capacity C, N non-overlapping spot beams covering the same area will provide capacity NC. However, in practice, the actual capacity provided is always lower than NC, in order to avoid interference between adjacent beams. Another beneficial effect of using narrow antenna beams is the introduction of high antenna gains, permitting power savings in both the uplink and downlink channels [5].

1.2 On-board Switching

While the use of multiple spot beams improves transmission aspects, it introduces the requirement for full connectivity between earth-stations. The satellite system then acts as a provider of dynamic links connecting any pair of earth-stations in the network whenever the need arises. From the network management standpoint, the usage of satellite resources has to be optimized when supporting a large population of earth-stations. On the other hand, the quality-of-service (QoS) required by users should be maintained. To achieve both the efficient utilization of space segment resources and acceptable user QoS, an element of switching is required in the satellite network [8].

The first issue to address, when a switch is being introduced into the satellite system, is its location; that is whether it should be placed in a ground terminal or on board the satellite. In the case of an on-board switch, two earth-stations communicate with each other in one hop (up and down transmission) through the satellite. On the other hand, the same earth-stations would need two hops to be interconnected when the multiple spot beam satellite is used as a simple repeater in the sky; one hop from the source station to an intermediate switching node on-ground, and a second hop from the intermediate switching station to the destination station. Each hop through the satellite incurs some delay and the value of the total delay contracted is strongly related to the distance separating the satellite from the earth.

In the case of a geosynchronous or geostationary earth orbit (GEO) satellite system with an on-ground switch, communication between two different earth-stations requires approximately 500 ms. This will certainly be unacceptable for delay-sensitive applications. In the case of low earth orbit (LEO) satellite systems, two-round trip delays correspond to 20 ms, a delay value that is within the limits of most service requirements. Although the propagation delay is not a limiting factor in an on-ground switch implementation for LEO systems, tracking of the on-ground switching node by the satellites becomes an issue because of the non-geostationary orbit. This tracking mechanism added to the already complex nature of a LEO satellite system makes an on-ground switch configuration less favourable in the case of LEO as well.

Additional advantages of the on-board switch configuration include increased flexibility to allocate system resources, such as uplink and downlink bandwidth, and independent uplink and downlink optimization. This last advantage comes from the fact that the uplink-transmitted digital signal is completely regenerated on board the satellite, by performing demodulation down to baseband. After switching and amplification, the signal is remodulated onto the downlink carriers. Thus, the uplink and the downlink can be designed as separate independent links.

1.2.1 On-board Circuit Switching

The most popular and easy to implement switching technique is circuit switching. Used in terrestrial networks, it offers delay-sensitive (real-time) services such as telephony. In circuit switching, a complete path is set up from the origin to the destination when a call is made. The path remains dedicated to that call until one of the two communicating parties releases it.

At least two proposed satellite systems employ circuit switching on board the satellite; the European Space Agency (ESA) [9] and the National Aeronautics and Space Administration (NASA) Advanced Communications Technology Satellite [10]. Although circuit switching is attractive for stream type traffic, e.g., telephony, it is not very suitable in an environment where traffic demands are not constant throughout time, but vary in an abrupt and unpredictable way. This is exactly the kind of environment that future multimedia traffic creates. We describe this environment as bursty. Burstiness is characterized by the variability in the message-generation process, in terms of both the length of the message and the message-interarrival time.

1.2.2 On-board Packet Switching

Packet switching is a switching technique designed to handle bursty data traffic. In packet switching, the communications bandwidth is dynamically assigned to users on an as-needed basis. This is achieved through the segmentation of user information messages into a series of packets and independently routing them to their destination. Hence, during the inactivity period of a user, when no packets are created, the remaining user population can use the free bandwidth to transmit their packets, thus achieving a certain sharing level of the communications capacity. The level of sharing can be expressed through the measure of statistical multiplexing gain, where the term multiplexing refers to any technique which permits a number of independent users to share one physical facility. As we can see, the level of sharing is increased as the users information messages become more bursty. Finally, an additional advantage of the packet switching technology is that it can integrate stream and bursty traffic more easily than circuit switching can.

To support the larger bandwidth and higher bit rate demands of multimedia services, a refined packet switching technique called fast packet switching will have to be employed. Basically, this fast packet switching technique is very similar to the one proposed for Asynchronous Transfer Mode (ATM) networks in the sense that both employ a constant length packet format and have self-routing capability. The difference lies in the size of the constant packet format and the nature of the links between the users and the on-board packet switch. In the case of an ATM switch, the size of the packet (cell) is specified to be 48 bytes payload plus 5 bytes header, and there are permanent links between the users and the on-board switch. Meanwhile, in the satellite context, the packet format is unspecified, and the links between the users and the on-board switch are dynamically allocated on demand. In this thesis, we will only address fast packet switches and hence, for simplicity, we will refer to them as packet switches. The term packets will be used to designate fixed-length packets in general. Nevertheless, we will use the term cell when ATM switches are involved.

On-board packet switching has been researched by companies and space agencies in many countries including Canada, the United States, and European countries. For example, the ESA program [11, 12], the NASA ACTS program [13, 14], Spar Aerospace Limited and the Canadian Department of Communications [15, 16], Teledesic's network of 288 LEO satellites [17], and Motorola's Celestri System [18]. A more detailed list of all the broadband satellite projects can be found in [19].

All these proposed projects are aimed at providing a broadband satellite communications system to support future multimedia applications. Accordingly, their common primal concern is the type of on-board switch they will employ and how it will behave in a multimedia environment. This is not only an issue linked with space communications, but it is also at the heart of research in terrestrial communications [20, 21, 22, 23, 24, 25]. The reason is that an ideal switch for every imaginable application does not exist, and it is both inefficient and inconceivable to design a new switch for every new application that appears. In that sense, it is essential that a performance study of the existing packet switches, under the light of the new application, be undertaken before any attempt to design a new switch. This will provide the designer with information on the capabilities and limitations of this system. As multimedia becomes the service of the future, satellite-switched systems have to be designed to support the increased traffic generated by these applications.

1.3 Thesis Outline

In this thesis, we will address the performance evaluation of an on-board packet switch in a multimedia environment. Due to the complex nature of multimedia traffic, which does not lend itself easily to analysis, system simulations were considered to be the best alternative.

In Chapter 2, we present a detailed description of the different components of a satellite system, i.e., uplink beam components, on-board packet switch and downlink beam components. We introduce the traffic types that our system supports, namely, real-time (voice and video) and jitter-tolerant (data). Moreover, we elaborate briefly on the multiple access schemes used at the uplink. Finally, we will review the fast packet switch architectures, focusing particularly on two switching fabrics, the shared-memory switch and the Knockout switch.

Chapter 3 presents the simulation model of our system. We will describe the models of the different traffic types. For voice and video traffic, we will use the well-known Markov Modulated Poisson Process (MMPP) model [26]. For data traffic, we will use two different models; the first one is the above-mentioned MMPP model and the second one the recently proposed model in [27], the Pareto Modulated Poisson Process (PMPP) model. Note that this last model resembles the MMPP model in all aspects but the sojourn times of the controlling Markov process and has been introduced to capture long-range dependencies in data traffic [28]. Furthermore, we will delineate the simulation models of the uplink, the Knockout and the shared-memory switches, and the downlink.

Chapter 4 presents the results of our simulative analysis. Specifically, we are concerned with the packet loss probability for real-time traffic at the different stages of the system. The packet loss probability as well as several performance indices relating to the queueing of the jitter-tolerant traffic will also be discussed. Moreover, we present these results for the two different types of switches that we used in our system, that is the Knockout and the shared-memory switches. Parametric analysis pertaining to the switch size and the packet destination distribution is also performed.

Chapter 5 concludes the thesis discussing the impact of our simulation results on the system design and suggests several future work.

CHAPTER 2
SATELLITE SYSTEM DESCRIPTION

In this chapter, we describe the satellite system under study. Starting with a brief system overview, we will look in detail at the functions of the different components of the system. Complexity as well as performance of on-board switching are of great concern since the satellite is limited in mass, power and bandwidth. We will review several available packet switches and we will select those which are more attractive based on their features, both promising and suitable for our applications.

Figure 2.1 shows a typical multiple spot beam satellite communications system using an on-board packet switch to provide multimedia services to earth-stations.

In order to transmit information via the satellite network, the end-user must first issue a call admission request to a scheduler on board the satellite. The on-board scheduler will decide whether or not to grant admittance to the call, based on a set of parameters which include the type of QoS the system has to provide to the end-user and the current or predicted status of the network.

Once the call is admitted, the end-user begins to generate its bursty multimedia traffic in a packetized format. These packets are either forwarded directly to the earth-station or indirectly, through the help of existing terrestrial network, e.g., the Internet. In general, more than one end-user can be connected to an earth-station,

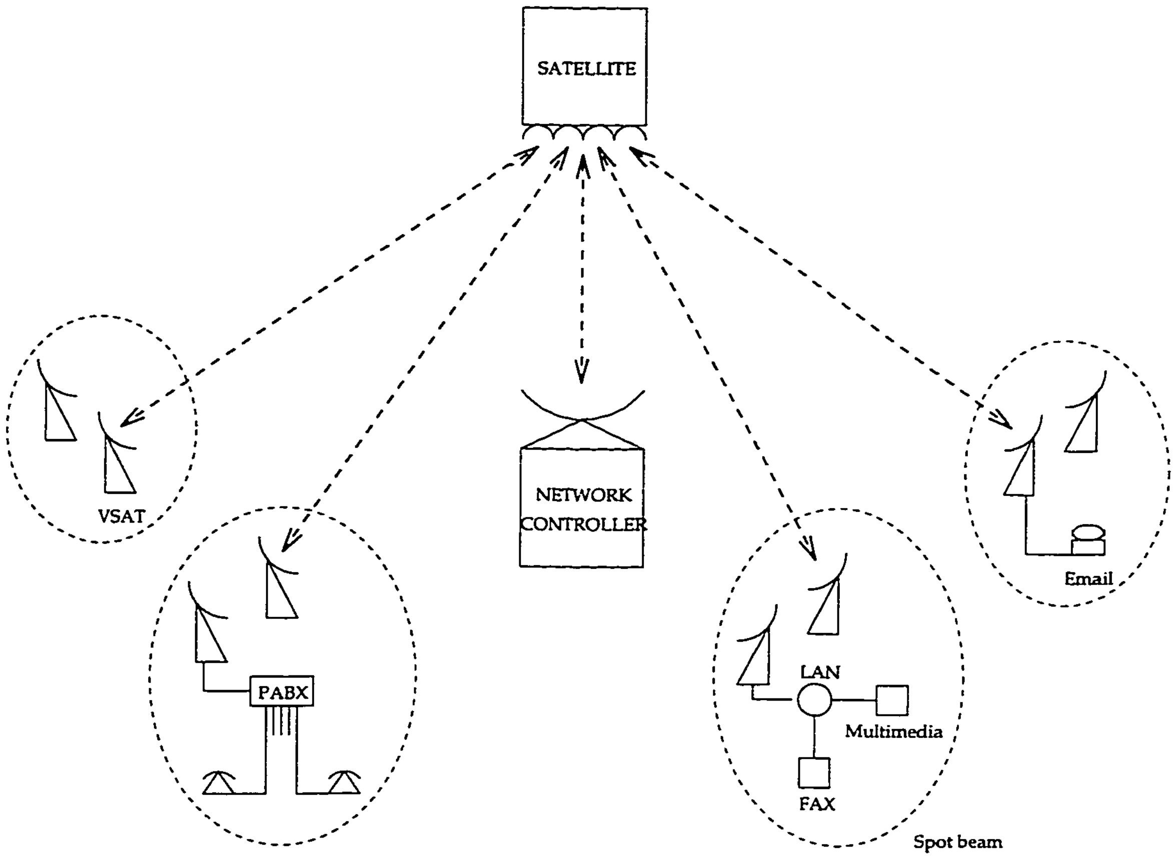

Figure 2.1: A typical multibeam satellite communications system

creating a confluence of all earth-station end-users traffic.

In order to access the satellite, the aggregate earth-station traffic has to contend for uplink capacity with the aggregate traffic of all the other earth-stations in the footprint of the uplink spot beam. As a result, a multiple access scheme in conjunction with queueing at the earth-stations is required to efficiently divide the limited uplink bandwidth into channels and to dynamically allocate these channels to fit the demands of the traffic.

At the satellite, the packets carried by the different uplink beams are processed and switched according to their destination downlink beam request. There is a possibility that two or more packets have requested the same destination, causing a destination contention problem that is resolved accordingly by the switching fabric used.

Switched packets to be resent to earth in the same downlink beam will have to contend among themselves for the limited downlink capacity. As such, queueing is required at the downlink.

Finally, the unique broadcasting capability of the satellite enables the downlink beam traffic to be received at all the earth-stations in the coverage area of that beam. The destination earth-stations will then forward only those packets intended to their end-users.

2.1 End-user Applications

We assume that the admitted end-users employ bursty multimedia applications. Thus, at any given time, each end-user can generate voice (e.g., telephone call), video (e.g., videophone call) or data (e.g., file transfer) traffic. Voice and video traffic have very strict requirements in terms of delay and delay jitter (variation), but can sustain some packet loss. On the other hand, data traffic is loss-sensitive (e.g., a bank transaction), but can absorb variable delays. Due to their nature, we will refer to voice and video traffic as real-time traffic, and data traffic as jitter-tolerant traffic.

This general method of classifying services can also be used in other service-providing systems. For example, in the context of ATM, there are five proposed classes of services, namely, constant bit rate (CBR), real-time variable bit rate (rt-VBR), non-real-time variable bit rate (nrt-VBR), available bit rate (ABR), and unspecified bit rate (UBR) [2]. CBR (e.g., uncompressed voice) and rt-VBR (e.g., compressed video) traffic, having tight constraints on end-to-end transfer delay and delay jitter, are real-time traffic. Meanwhile, nrt-VBR (e.g., image), ABR and UBR (both specified to accommodate the transfer of data, e.g., HTTP and FTP) traffic, having no constraints on delay jitter and being tolerant to delay, are jitter-tolerant traffic.

2.2 Uplink Beam Components

As multimedia information is generated by the end-user, it is packetized and forwarded to the end-user earth-station. The earth-station serves as a gateway to the space segment, and it is possible that more than one end-user can be connected to it. Since the uplink beam has a fairly wide coverage area, more than one earth-station is within the footprint of the beam. This results in a situation where there is a large population of users requiring access to a bandwidth-limited link. For this reason, an efficient uplink multiple access scheme is required.

In order to support a wide range of traffic types and provide a high level of statistical multiplexing, Multiple Frequency Time Division Multiple Access (MF-TDMA) is used at the uplink [16, 29, 30]. Moreover, MF-TDMA employs much lower transmission bit rates when compared to TDMA, permitting the use of smaller and cheaper earth-stations. Under MF-TDMA, the bandwidth of each uplink spot beam is organized as a group of n equal-size frequency slots, each is shared by the earth-stations in a TDMA manner on a demand-assignment basis. Each TDMA carrier is further divided into c equal-size, non-overlapping time slots per frame. Thus, an uplink beam has a total capacity of $C = nc$ frequency-time slots per frame, each of which can accomodate a fixed-length packet (e.g., an ATM cell). The frame structure is shown in Figure 2.2. A small portion of the uplink MF-TDMA frame is reserved for signal-

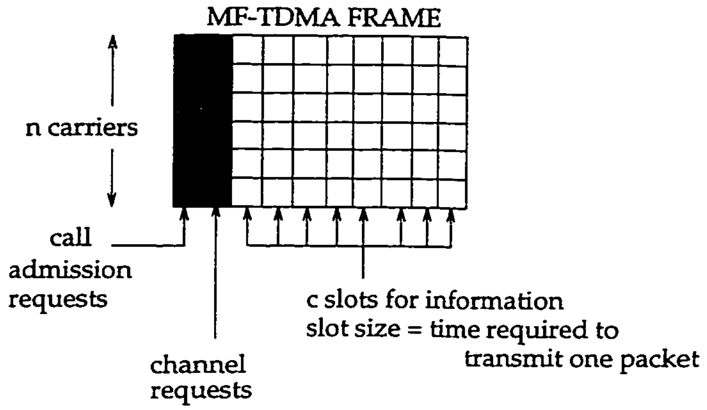

Figure 2.2: An MF-TDMA frame ($C = nc$ slots)

ing, e.g. channel requests. Each earth-station can transmit up to c non-overlapping time-frequency slots. In other words, c is selected to accomodate the maximum peak rate of the earth-station, e.g. 2.048 Mbps. The aggregate peak rate when all earth-stations are active can be larger than the allowable peak rate in the MF-TDMA frame structure, but the probability that all earth-stations are active at the same time is very low, as we will see in Chapter 4.

Since end-users employ bursty multimedia applications, the number of packets arriving at an earth-station in a frame can vary from one frame to another. For this, an on-board scheduler is used to dynamically allocate non-overlapping frequency-time slots to demanding earth-stations in each frame. However, due to the nature of the traffic and the large amount of end-users, the number of packets arriving at the earth-station during a frame can still exceed the uplink capacity. In anticipation of such a case, a priority scheme in conjunction with queueing at each earth-station is implemented.

Since jitter-tolerant packets can be queued and forwarded at a later time while real-time traffic cannot tolerate delay variation, the scheduler gives a higher priority to real-time traffic in dynamic capacity allocation, i.e., the scheduler first allocates frequency-time slots to real-time traffic, and then uses the remaining capacity to accommodate jitter-tolerant traffic. In this way, if more than C real-time packets require transmission in a given TDMA frame, the excessive ones are lost. If the number of real-time packets demanding uplink transmission is smaller than C, but the total of arrival real-time and jitter-tolerant packets in a TDMA frame exceeds C, then the excessive jitter-tolerant packets will be stored in the on-ground queues of the corresponding earth-stations.

Packets received from different carriers of the same uplink beam are multiplexed into one stream at the rate of C packets per frame and passed to the input port of the $N \times N$ on-board packet switch.

2.3 On-board Packet Switching

We chose to investigate on-board packet switching over on-board circuit switching for its dynamic and efficient allocation of bandwidth, permitting high statistical multiplexing gains in a bursty multimedia environment. However, when considering on-board packet switching, output port contention and queueing are of major concern.

Output port contention refers to the situation where more than one packet arriving in the same slot are destined to the same output port. This problem occurs in packet switching due to the absence of coordination among arriving packets as far as their destination requests are concerned. As a result of the contention problem, queueing is needed to store the contending packets.

A plethora of packet switching fabrics have been proposed to solve the contention and queueing problems [21, 22, 24, 23, 25]. We will now review these switching fabrics and highlight the candidates for satellite applications.

Packet switching fabrics can be classified based on different attributes [21, 22, 25]. We choose to classify them according to the type of physical connection implemented between the input and output ports. Based on this attribute, we have two families of switching fabrics: time-division switch and space-division switch [25]. Furthermore, time-division switches are themselves divided into two classes: shared-memory and shared-medium switches. In turn, space-division switches are composed of crossbar-based switches, Banyan-based switches, and switches with N^2 disjoint paths [22].

2.3.1 Time-division Switch

A switch can be regarded as a communications resource which is shared by all input and output ports. A time-division switch allows access to this resource via a time division multiplexing (TDM) scheme, where all ports transmit according to a common time reference. An important feature of this family of switches is that their cost and complexity increase linearly with their size N.

Shared-memory switch [22]

Packet switching fabrics of the shared-memory type consist of a memory shared by all input and output lines (Figure 2.3). The packets arriving on all input lines are multiplexed onto a single stream which is fed to the common memory for storage. The common memory can be completely partitioned into N separate sections, one section for each output port, or shared among all output ports. The complete partitioning of

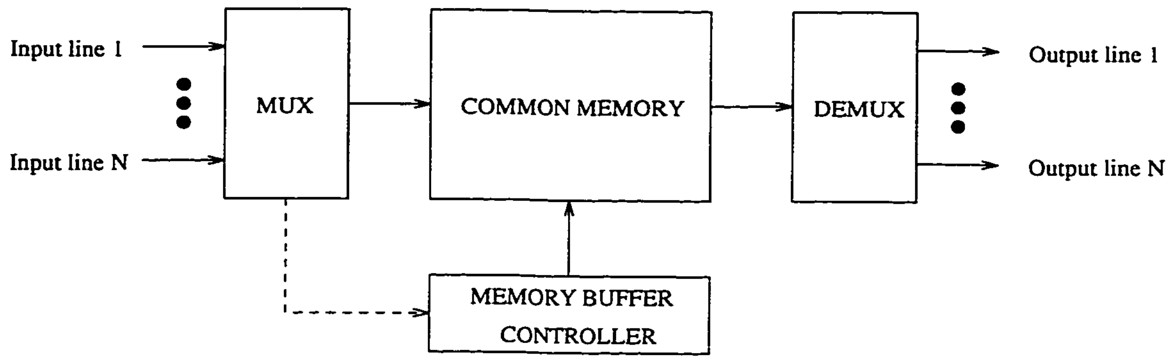

Figure 2.3: A shared-memory switch

the memory permits the implementation of a simple array of first-in first-out (FIFO) buffers, but can require a larger overall size of memory. On the other hand, fully sharing the memory can reduce the overall size, but requires a more complicated memory management.

The actual routing function in shared-memory switch is performed by a central controller. In order to route all incoming packets to their requested destination port, the controller must be capable of processing sequentially N incoming packets and selecting N outgoing packets every time slot. For an input and output port speed V, the required memory bandwidth of a shared-memory switch will be $2NV$. In order to alleviate the demand for high memory access speeds, a parallel memory (or W bit-slice) organization can be implemented.

One of the drawbacks of this switching fabric is a lack of modularity which limits its size expansion capability.

Shared-medium switch [22]

In the shared-medium switch (Figure 2.4), all arriving packets on the N input lines are synchronously multiplexed onto a common high-speed medium (which can be a bus or a ring) requiring a bandwidth of NV, for an input and output port speed V. Each output line is in turn connected to the shared-medium through an address filter, capable of receiving all packets transmitted on the medium, and an output buffer. It is the address filter that decides if a packet observed on the medium is destined to that output.

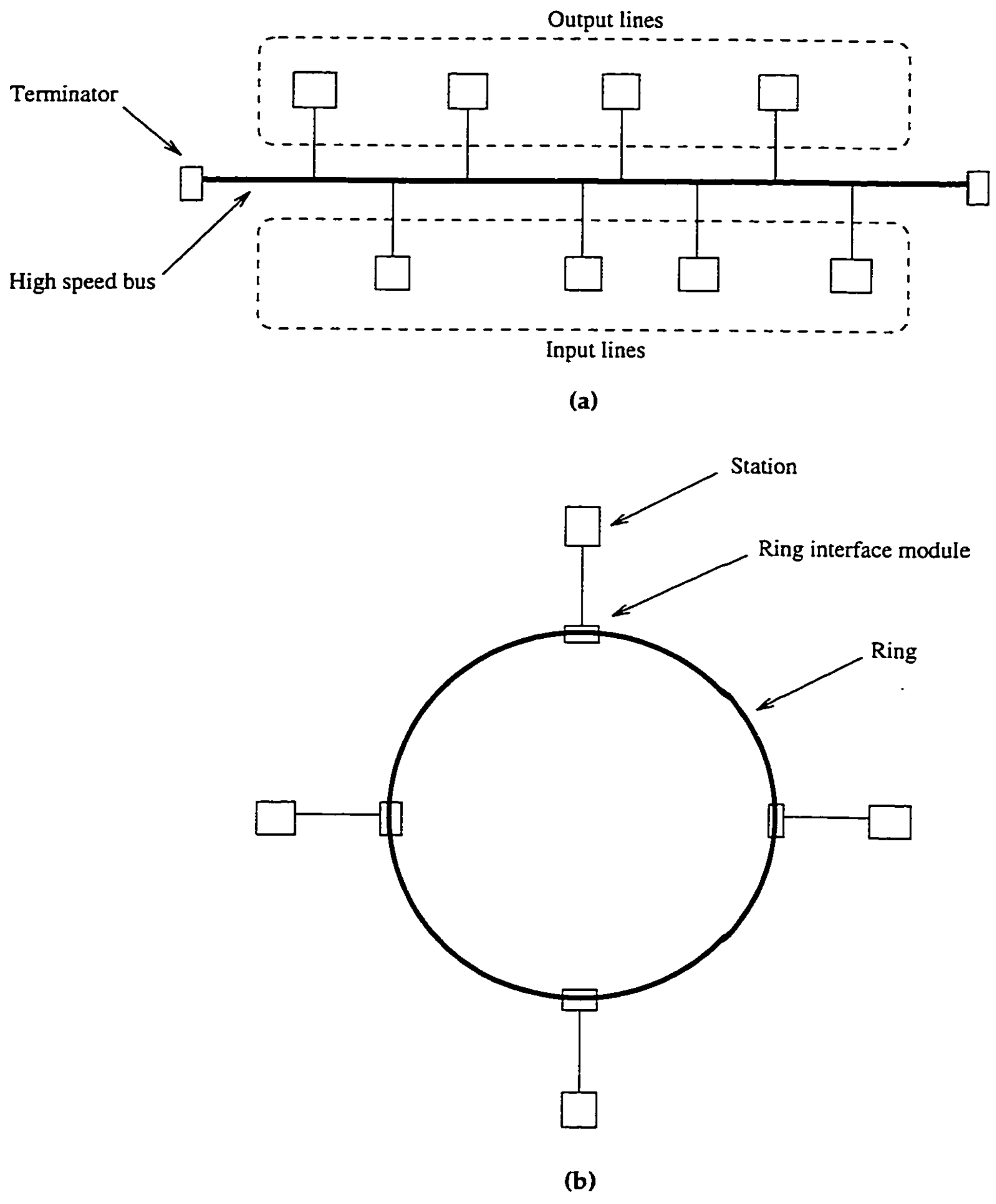

Figure 2.4: A shared-medium switch: (a) shared-bus, (b) shared-ring

Similar to the shared-memory switch, one of the main concerns in a shared-medium switch is how to implement the high-speed medium. Again, to meet the bandwidth requirement, we can use a parallel organization to reduce the circuit speed.

The structure of the shared-medium switch makes it easier to expand in size than the shared-memory switch, but there is still a limit on the achievable switch size due to the bandwidth limitation. In addition, shared-medium switch enjoys the advantages of a higher degree of fault-tolerance, and the ability to support multicasting/broadcasting functions with little modifications. However, a drawback of the shared-medium switch is the buffers cannot be pooled and shared among the output ports.

2.3.2 Space-division Switch

Contrary to time-division switch, where all the inputs are multiplexed onto a stream in order to share the switching resource, in space-division switch, multiple connections exist between the switch input and output ports. As a result, no memory component in the switch has to operate at a speed higher than the port speed. However, the complexity of space-division switches grows faster than the linear relation in time-division switches. In some cases, it can reach the order of N^2.

Crossbar-based switch [22]

In a crossbar-based switch, each pair of input and output ports is connected via a crosspoint switch, resulting in a square array of N^2 crosspoint switches (Figure 2.5). Crossbar-based switches are internally non-blocking; two packets originating from

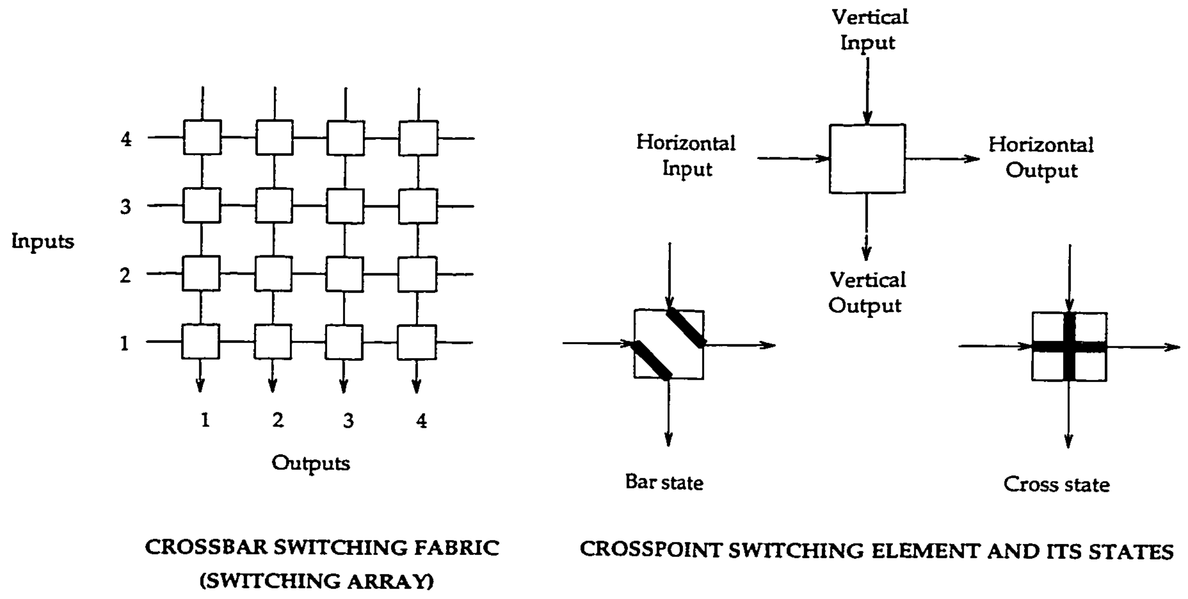

Figure 2.5: A crossbar-based switch

different sources and destined to different output ports can be switched without internal conflicts. However, if packets contend for the same output port, buffering is mandatory for the packets having conflicting destination port. The possible locations of the buffers are at the inputs, outputs or crosspoints of the switching array.

The advantages offered by this class of switches include the self-routing capability, simplicity and modularity of the switching array. However, the complexity of the switch is of the order of N^2.

Banyan-based switch [22]

Banyan-based switches are designed with the objective of reducing the number of switching elements to less than the N^2 required for the crossbar-based switch. As a result of the lower number of switching elements, input lines will have to share the use of some switching elements. Although sharing the use of switching elements helps reduce the switch complexity, it imposes the problem of internal blocking. There are various derivatives of the multi-stage interconnection networks that have been proposed and studied to resolve internal blocking [32, 33, 34, 35, 36, 37, 38].

On the positive side, Banyan-based switches are modular and self-routing.

Switches with N^2 disjoint paths [22]

Due to the performance setbacks encountered in the two previous space-division switch classes, efforts have been made to alleviate the problems of output contention and internal blocking. The switch with N^2 disjoint paths employs the maximum available hardware resource to allow each input port to be directly connected to every output port.

Within the switching fabrics with N^2 disjoint paths, the Knockout switch has distinguishing advantages, such as simplicity, modularity, low latency and self-routing [39]. In the sequel, we will focus our attention exclusively in its description.

The Knockout switch [39]

The Knockout switch operates on a slot basis. Broadcast buses are used to transport packets from N input ports to the corresponding output ports as shown in Figure 2.6. Each output port has a bus interface that selects the corresponding packets. In each time slot, the bus interface receives up to N packets from all input ports, and can switch up to L packets, where L is less than or equal to N. Excessive packets are discarded. As L increases, the loss probability is reduced but the switch complexity is increased.

The many advantages associated with the Knockout switch are a reduction in

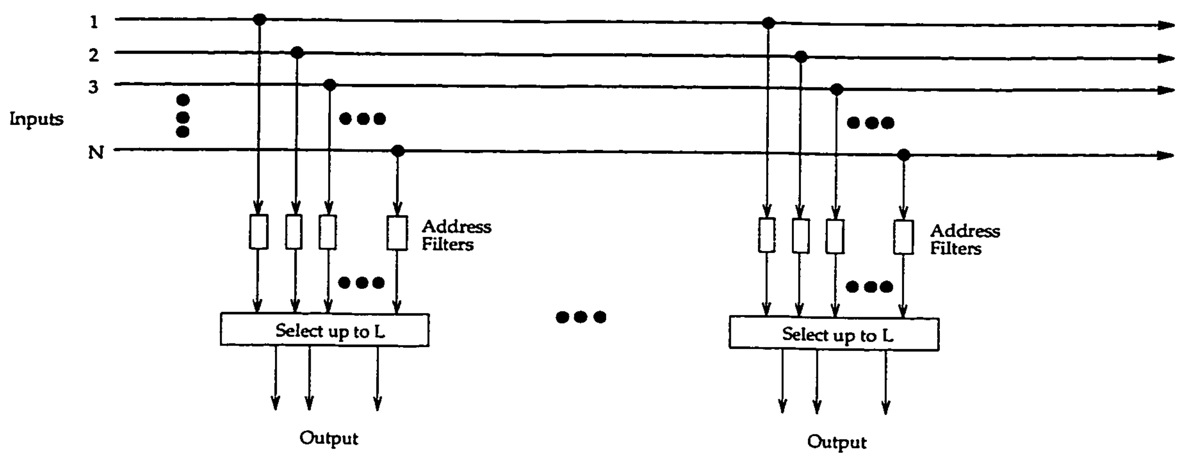

Figure 2.6: A Knockout switch

resource wastage, ease of implementing a priority scheme in the Knockout contention, ease of implementing multicasting/broadcasting function, cell order preservation, and modularity.

2.3.3 Candidates for Satellite Applications

When choosing a switching fabric for satellite purposes, special attention must be paid to its unique spatial environment. Due to the mass limitation and the long mission lifetime (approximately a decade) of the spacecraft, complexity and fault tolerance become important issues. In addition, the numerous and diversified applications necessitate tighter bounds on performance in terms of packet loss and delay. On the other hand, while modularity is of particular interest in terrestrial applications, it is of secondary importance in satellite applications [16]. With these criteria in mind, we will now select the candidates most suitable for an on-board switch system.

From our earlier discussions, we know that time-division switches have the lowest level of complexity. However, they require high memory access speeds. Since a shared-medium switch has to have dedicated output buffers for each output port, it will require more memory compared to a shared-memory switch employing a fully shared buffer. In that sense, the shared-memory switch presents a stronger case for satellite applications. Especially, when we consider the improved shared-memory switch employing parallel access proposed in [31]. The novel approach in switching reduces considerably the high memory access speed requirements. For these reasons, the shared-memory switch is a good candidate for satellite communications.

As for space-division switches, we observed that the cost of extra complexity to improve the performance in the case of the crossbar-based switch grows in the order of N^2. Hence, to achieve a respectable performance, the complexity of the switch will not be attractive for satellite communications anymore. We also argued that the major drawback of Banyan-based switches is internal blocking. The various derivatives of the Banyan-based switch that have been proposed to solve the internal blocking involve the use of multi-stage interconnection networks in a stand-alone, multi-plane or tandem organization [32, 33, 34, 35, 36, 38]. Thereby, the complexity level of the switch is significantly increased. Finally, we perceived that the Knockout switch (belonging to the N^2 disjoint paths class) offered an efficient scheme to resolve output port contention while maintaining a high level of performance. In addition, the ease of implementing a fault-tolerant system, the self-routing capability and the simple switch reconfiguration to support multicasting and broadcasting functions make the Knockout switch the leading candidate for satellite applications.

2.4 Downlink Beam Components

At the output of the on-board packet switch is the downlink. Unlike the uplink, no multiple accessing scheme is required, and hence, packets arriving at the downlink are simply TDM onto a carrier on a frame basis. Since switched packets are uncoordinated as far as their destination requests are concerned, the number of switched packets can exceed the downlink capacity. Again, a priority scheme in conjunction with downlink queueing is implemented.

The downlink queue has to be distinguished from the switch buffer due to their different raisons d'être. The switch buffer is needed when the available switching capacity cannot route all incoming packets to their appropriate output port. On the other hand, the downlink queue is required when the available downlink capacity cannot support all switched packets.

Assuming symmetry between input and output, the downlink TDM carrier also has a capacity of C packets per frame. Similar to the uplink allocation, the downlink server gives real-time traffic a higher priority. If the number of real-time packets arriving at an output port in a frame exceeds the capacity C, the excessive ones are discarded. If there are less than C real-time packets but the total number of arrival packets is larger than C, excessive jitter-tolerant data packets are stored in the on-board downlink queue and forwarded whenever capacity becomes available. Jitter-tolerant traffic loss occurs on the downlink only when the downlink queue is full. Downlink queues for jitter-tolerant traffic can be organized as separated memory blocks, one for each output port. Alternatively, the N downlink queues can be combined in one single common memory block. By using separated memory blocks, the downlink queue is simply a FIFO buffer and has a simple control. However, this approach can require a larger overall size of memory. On the other hand, the common memory approach can reduce the overall size, but requires a more complicated memory management.

2.5 Methodology of Performance Evaluation

The performance of a communications system is mainly dependent on the traffic it has to serve. In our case, we evaluated two classes of traffic, namely, real-time traffic and jitter-tolerant traffic. While jitter-tolerant traffic can be queued, real-time traffic has stricter requirements on delay and delay variation, and thus cannot. In order to evaluate their performance, we will first define the different performance measures, followed by the approach we will use to obtain these values.

We define average packet delay as the average transit delay of a packet from the time it arrives at its originating earth-station to the time it has reached its destination earth-station. Since the medium propagation delay is a constant, dependent entirely on the orbital location of the satellite, we choose to exclude it in our delay measurement. (As a matter of fact, we can always obtain a particular mean delay for any orbital system by simply adding its round-trip delay to the measured value.) In addition, we will also measure the standard deviation of the delay, and, when possible, the probability of queue exceeding any given threshold value.

Another performance measure of importance is the packet loss probability. The packet loss probability is the probability that a packet is lost in the system as a result of limited resources (memory and/or switching capacity). It is defined as the ratio of the total number of packets lost to the total number of packets actually sent.

The performance study of this system aims to provide us with the values of the parameters just described. In order to appreciate the results of such a study, we must clearly understand the issues and nature of multimedia traffic modeling. Multimedia systems require the traffic modeling of the many services they offer. There is a vast number of mathematical models proposed to generate the traffic of the individual multimedia services (voice, video, data) [40, 41], but their mathematical tractability when combined has yet to be proved.

Additionally, the degree of complexity of an exact queueing analysis of such a system grows very rapidly when we consider larger switch sizes. This is due to the network of queues at the input and output of the switch, where the application of mathematical traffic modeling is highly likely to be non-parsimonious.

System simulation is the most viable approach given the complexity entailed in the network of queues analysis involving multimedia traffic. Furthermore, simulations provide a precise model of the system, rapidly available results and a means of validating analytical results (when analysis is possible).

CHAPTER 3
SIMULATION MODELS

We described in the previous chapter the satellite system under study. Due to the complex nature of this system, simulations were chosen as the most viable approach for a performance study of the system. In the following, we will present the simulation models of the system, beginning with a discussion on multimedia traffic modeling. The modeling and simulations of this system will be performed using OPtimized Network Engineering Tools (OPNET). The OPNET code used can be found in Appendix A.

Figure 3.1 shows the simulation model of the satellite system. It is composed of

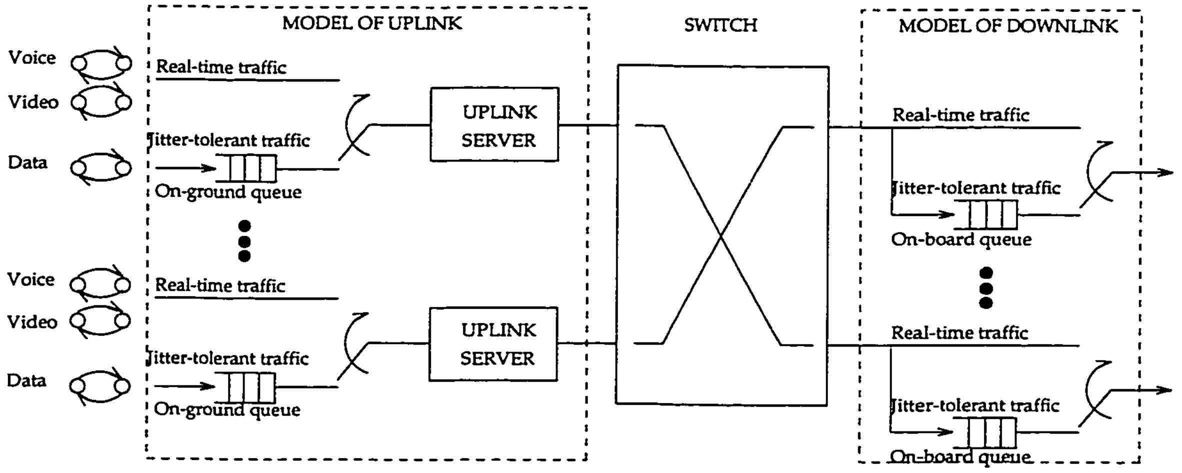

Figure 3.1: Simulation model of the satellite system

three parts, namely, the uplink model, the on-board switch model, and the downlink model. In the sequel, we will describe each of these parts in details, starting with the uplink model.

3.1 Uplink Model

As discussed in last chapter, the earth-stations are the end-users access points to the satellite. The earth-stations can be logically viewed as queueing facilities where the multiple accessing scheme to the satellite is implemented. In order to simplify the model at the uplink, we represent the ensemble of earth-stations in the coverage area of a spot beam by one mega-earth-station. This mega-earth-station is then modeled as a queue representing the sum of all earth-station queues (Figure 3.2).

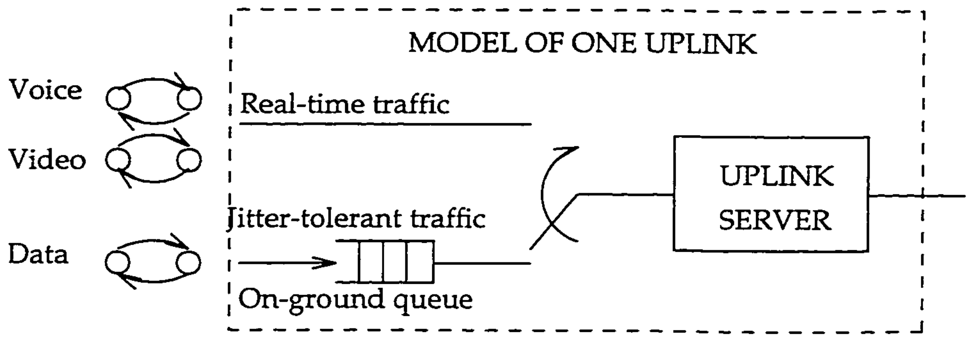

Figure 3.2: Simulation model of earth-station population of one spot beam

The mega-earth-station will then receive the aggregate traffic of the entire end-user population in a spot beam coverage area, and will forward the resulting packet traffic to the satellite using the uplink access scheme. Since our main goal is to study the performance of the on-board switch, we are only interested in the effect of the uplink access scheme on the input traffic to the switch. Thus, we will divide our discussion on the uplink model into two parts, namely, the traffic model and the model of the effect of uplink access.

3.1.1 Traffic Modeling

In this section, we will discuss the modeling of multimedia traffic [40]. We assume that the individual end-users have independent sources generating either voice, video or data packets. In the following, we will describe the modeling of the individual sources and their approximate aggregate model. We will also present an algorithm to capture the relevant characteristics of the traffic, such as the peak packet rate of an individual source, A, the peak-to-average rate ratio of an individual source, R, the number of end-users sources admitted for voice, N_{vo}, video, N_{vi}, and data, N_{d}, and the value of the index of dispersion of counts (IDC) at infinity, $\mathrm{I}(\infty)$. The index of dispersion of counts at time t, $\mathrm{I}(t)$, is equal to the ratio of the variance of the number of arrivals by time t to the mean number of arrivals by time t.

Voice source modeling

A packetized voice source, such as that of an end-user, can be modeled by an on-off source (Figure 3.3). During the silence period (off-state), no packets are transmitted. Meanwhile, during the talk period (on-state), packets are generated at a constant peak rate A packets/s. It is assumed that the sojourn time in a state is exponentially distributed with mean off-time $\frac{1}{\alpha}$, and mean on-time $\frac{1}{\beta}$.

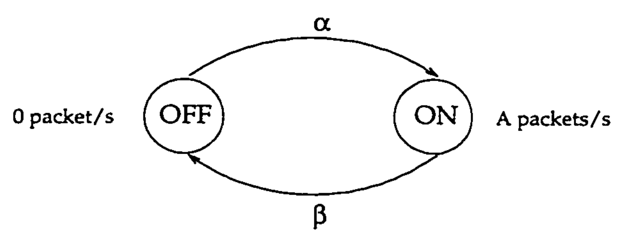

Figure 3.3: An on-off source; α and β, probability transition rates; A constant peak rate

In [26], it has been shown that a superposition of on-off sources can be modeled successfully by a 2-state Markov Modulated Poisson Process (MMPP). An MMPP is a Poisson process with rate modulated by a Markov Process. It can be better described by refering to Figure 3.4. The figure shows a 2-state Markov Process with states 1 and 2 and probability transition rates σ_1 and σ_2. When in state 1, the process generates packets with Poisson rate λ_1 and when in state 2, with λ_2.

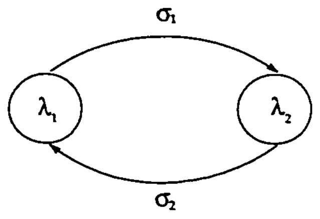

Figure 3.4: A 2-state MMPP source; σ_1 and σ_2, probability transmission rates; λ_1 and λ_2, Poisson packet rates

We follow the approach of [26] and model the aggregate of N_{vo} on-off voice sources by a 2-state MMPP source. Towards this end, we must choose the four parameters that characterize the MMPP process, namely λ_1, λ_2, σ_1, and σ_2. We perform this task by matching several statistical characteristics of the original superposition of on-off sources with the corresponding ones of the MMPP. At this point, we deviate from [26] and we use a different technique to perform the above matching.

Parameters matching technique [43, 44]

In this technique, we will match the I(∞), the mean arrival rate, and the duration of underload and overload periods, as explained below, of the original process with the same of the MMPP model. In the following, we will describe the technique assuming voice traffic only. However, the matching process can be applied as is for video and data traffic, as it will be discussed in the appropriate section.

In order to find $\lambda_{1,\mathrm{vo}}$ and $\lambda_{2,\mathrm{vo}}$, we will match the mean arrival rates in underload and overload states of the on-off model with the two rates of the MMPP. Underload and overload states in the on-off model are given with respect to a parameter $\omega \triangleq \lfloor \frac{N_{\mathrm{vo}}}{R_{\mathrm{vo}}} \rfloor$, dependent on the average capacity devoted to the voice traffic. When the number of active on-off sources, i, is less than or equal to ω, $i \leq \omega$, we say that we are in an underload state. In contrast, when $i > \omega$, we say that we are in an overload state. Denoting by π_i, $\pi_i = \begin{pmatrix} N_{\mathrm{vo}} \\ i \end{pmatrix} \left(\frac{1}{\mathrm{R_{vo}}}\right)^{\mathrm{i}} \left(\frac{\mathrm{R_{vo}}-1}{\mathrm{R_{vo}}}\right)^{\mathrm{N_{vo}}-\mathrm{i}}$, the probability that i sources are on, and performing the matching, we obtain

$$\lambda_{1,\mathrm{vo}} = \frac{\sum_{i=0}^{\omega}(iA_{\mathrm{vo}}\pi_i)}{\sum_{i=0}^{\omega}(\pi_i)} \tag{3.1}$$

and

$$\lambda_{2,\mathrm{vo}} = \frac{\sum_{i=\omega+1}^{N_{\mathrm{vo}}}(iA_{\mathrm{vo}}\pi_i)}{\sum_{i=\omega+1}^{N_{\mathrm{vo}}}(\pi_i)} \tag{3.2}$$

To find the two remaining parameters of the MMPP model, i.e., $\sigma_{1,\mathrm{vo}}$ and $\sigma_{2,\mathrm{vo}}$, we match the $\mathrm{I_{vo}}(\infty)$ of the two models, as well as their mean arrival rate. We know that for $\mathrm{N_{vo}}$ on-off voice sources the $\mathrm{I_{vo}}(\infty)$ and the average aggregate rate are given as [26]

$$\mathrm{I}_{\mathrm{on-off,vo}}(\infty) = \frac{1-(1-\frac{\alpha_{\mathrm{vo}}}{A_{\mathrm{vo}}})^2}{(\frac{\alpha_{\mathrm{vo}}}{A_{\mathrm{vo}}}+\frac{\beta_{\mathrm{vo}}}{A_{\mathrm{vo}}})^2} \tag{3.3}$$

and

$$\lambda_{\mathrm{on-off,vo}} = N_{\mathrm{vo}}\frac{A_{\mathrm{vo}}}{R_{\mathrm{vo}}} \tag{3.4}$$

Meanwhile, the same parameters for the 2-state MMPP are expressed as [26]

$$\mathrm{I}_{\mathrm{MMPP,vo}}(\infty) = 1 + \frac{2\sigma_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})^2}{(\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}})^2(\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}})} \tag{3.5}$$

$$\lambda_{\mathrm{MMPP,vo}} = \frac{\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}}}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}} \tag{3.6}$$

Matching Equations (3.4) and (3.6), as well as Equations (3.3) and (3.5), we obtain

$$\lambda_{\mathrm{on-off,vo}} = \frac{\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}}}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}}$$

and

$$\mathrm{I}_{\mathrm{on-off,vo}}(\infty) = 1 + \frac{2\sigma_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})^2}{(\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}})^2(\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}})} \tag{3.7}$$

Noting that

$$\lambda_{1,\mathrm{vo}} - \lambda_{\mathrm{on-off,vo}} = \lambda_{1,\mathrm{vo}} - \frac{\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}}}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}} = \frac{\sigma_{1,\mathrm{vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}}$$

and

$$\lambda_{\mathrm{on-off,vo}} - \lambda_{2,\mathrm{vo}} = \frac{\lambda_{1,\mathrm{vo}}\sigma_{2,\mathrm{vo}} + \lambda_{2,\mathrm{vo}}\sigma_{1,\mathrm{vo}}}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}} - \lambda_{2,\mathrm{vo}} = \frac{\sigma_{2,\mathrm{vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})}{\sigma_{1,\mathrm{vo}} + \sigma_{2,\mathrm{vo}}}$$

we can solve Equation (3.7), yielding the last two equations,

$$\sigma_{1,\mathrm{vo}} = \frac{2(\lambda_{1,\mathrm{vo}} - \lambda_{\mathrm{on-off,vo}})^2(\lambda_{\mathrm{on-off,vo}} - \lambda_{2,\mathrm{vo}})}{\lambda_{\mathrm{on-off,vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})(\mathrm{I}_{\mathrm{on-off,vo}}(\infty) - 1)} \tag{3.8}$$

and

$$\sigma_{2,\mathrm{vo}} = \frac{2(\lambda_{1,\mathrm{vo}} - \lambda_{\mathrm{on-off,vo}})(\lambda_{\mathrm{on-off,vo}} - \lambda_{2,\mathrm{vo}})^2}{\lambda_{\mathrm{on-off,vo}}(\lambda_{1,\mathrm{vo}} - \lambda_{2,\mathrm{vo}})(\mathrm{I}_{\mathrm{on-off,vo}}(\infty) - 1)} \tag{3.9}$$

We will use the measurements of the individual on-off sources provided by Spar Aerospace Limited, i.e., $\mathrm{I}_{\mathrm{on-off,vo}}(\infty)$, A_{vo}, and R_{vo}. However, $\lambda_{\mathrm{on-off,vo}}$ will depend on the number of on-off sources, N_{vo}, we will aggregate. These four parameters will be used in Equations (3.1), (3.2), (3.8) and (3.9) to obtain values of λ_1, λ_2, σ_1, and σ_2.

Video source modeling

There are various techniques to model video traffic [40]. From these, we choose the approach proposed in [45] where a video source or the superposition of video sources is modeled by an aggregate of a number of mini-on-off sources. Each mini-on-off source is a process similar to that represented by the model in Figure 3.3, used to characterize the traffic of a voice source. As a consequence, the superposition of the many video sources can be modeled by a 2-state MMPP. This approach permits us to apply the same matching technique used in voice modeling.

Data source modeling

At the time of this writing, the modeling of computer data traffic is still a subject of hot debate [40]. One way of modeling a data traffic is through the same technique described above which was used to model the video sources. Then, the superposition of many data sources can also be modeled by a 2-state MMPP, and the matching technique used in voice modeling can be applied. However, recent measurements [28] have revealed that data traffic has long-range dependence [1] and self-similar characteristics, which are not captured by Markov chain-based models.

We will use a 2-state MMPP with theoretical value of $IDC(\infty) = \infty$ and we will also consider the model in [40], where a 2-state Pareto Modulated Poisson Process (PMPP) model was proposed to capture the long-range dependent nature of data traffic. A PMPP resembles to an MMPP in all aspects but the sojourn times of the controlling Markov process. The sojourn times of a PMPP are independent and identically distributed with a Pareto distribution having parameter $\alpha_{\mathrm{H}} = 3 - 2H$, where H $(0.5 < H < 1)$ is used to measure the degree of self-similarity and is called the Hurst parameter.

We can use a similar matching technique as above to choose the parameters of the PMPP model, namely, $\lambda_{1,\mathrm{d}}$, $\lambda_{2,\mathrm{d}}$, and H. We will, thus, match H of the data traffic with the same of the PMPP model. In addition, we will use a similar approach to the underload and overload matching described for voice modeling to obtain $\lambda_{1,\mathrm{d}}$ and $\lambda_{2,\mathrm{d}}$.

3.1.2 Modeling of the Effect of Uplink Access

The sources models described in the previous section are used to generate the aggregate traffic at the uplink. Since source traffic has to contend for slots on the uplink MF-TDMA frame of capacity C, queueing is required on-ground (Figure 3.2).

We model the otherwise distributed queue, that is formed in every earth-station

[1]The autocorrelation function of the number of packets generated in each slot of long-range dependent traffic decays hyperbolically as the lag increases, while that of short-range dependent traffic decays exponentially as the lag increases

accessing the same uplink capacity C, as a unique mega-earth-station uplink queue, ignoring its distributed nature. The uplink server in Figure 3.2 represents the queue scheduler, which otherwise resides on board the satellite. The scheduler discriminates between the different traffic types trying to access the uplink capacity by giving priority to the real-time traffic. However, there is no priority between voice and video.

In a MF-TDMA frame, the voice, video, and data sources generate X, Y, and Z packets, respectively. The Z packets enter the on-ground queue. The scheduler first serves the voice and video traffic. If $(X + Y)$ exceeds C then there are $(X + Y - C)$ real-time packets discarded (lost). Otherwise, the scheduler will use the remaining $(C-(X + Y))$ slots to serve the jitter-tolerant packets in the front of the on-ground queue. Jitter-tolerant data traffic is lost when overflow occurs in the on-ground queue. In order to investigate the packet loss behaviour for different queue sizes, we assume an infinite queue in our simulation model, and examine the survivor function, i.e., probability that the queue length exceeds a threshold value.

Finally, we need to model the slot assignment of the frame performed by the scheduler. The actual scheduler assigns slots of the MF-TDMA frame in a semi-random fashion (Figure 3.5 a), insuring that no user can transmit at more than one frequency carrier during a time slot. The frame is then sent to the satellite where it is multiplexed into one stream at rate C packets per frame. Due to this step, the resulting stream at the input of the on-board switch seems to have their slots randomly allocated to each user (Figure 3.5 b). We will model this effect of the uplink access scheme by randomly assigning the served packets a slot in the uplink

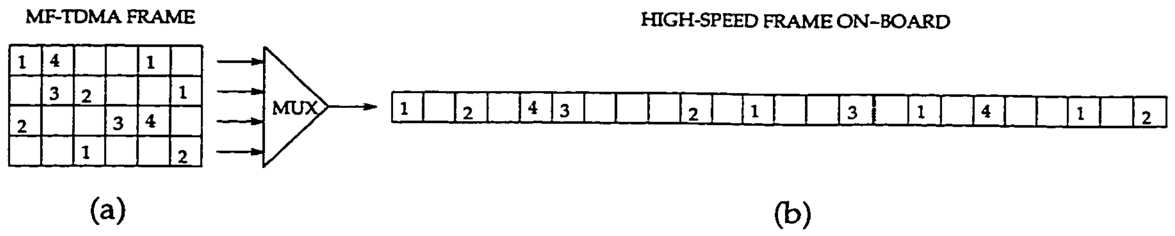

Figure 3.5: Scheduler slot assignment: (a) MF-TDMA frame sent at uplink, (b) multiplexed MF-TDMA frame into a high-speed stream on board the satellite

frame.

3.2 On-board Switching Model

At the satellite, the MF-TDMA frame of a spot beam is multiplexed into one stream at the rate of C packets per frame, and passed to the input port of the $N \times N$ on-board switch. The way these packets are switched depends on the type of switch. We will model two types of switches, namely, the shared-memory and the Knockout switches [39]. Note that in section 2.4 we have discussed the different purposes of the switch buffer and the downlink queues: the former is to store the packets requesting the same output ports while the latter contains switched packets contending for downlink capacity. Nevertheless, as the output of the switch buffer is actually the input to the downlink queues, instead of modeling two side-by-side memory areas, we select to model the memory component of the switch as an integral part of the downlink queues and will, consequently, address this in the downlink model section.

3.2.1 Shared-memory Switch Modeling

The model of the shared-memory assumes that there is no restrictions on memory access time. As a result, no packets will be lost at the switch level. The shared-memory switch can operate on a slot or a frame basis.

3.2.2 Knockout Switch Modeling

The Knockout switch operates on a slot basis. At every time slot, the switch can receive up to N packets from the N input ports. However, the switch can only route L ($L < N$) packets to a particular output port during the same time interval. As a consequence, if the number of packets requesting the same output port is greater than L, the exceeding packets are lost. There is no priority given at the switch, and hence, there is no discrimination in packet loss between real-time and jitter-tolerant traffic.

In the next section, we will discuss the downlink model.

3.3 Downlink Model

Switched packets are stored at the downlink queues, where they will be transmitted to the destination earth-stations in a TDM manner. We assume that the downlink has the same capacity as the uplink, which is C packets per frame. The downlink TDM server gives a higher priority to real-time traffic. In a given TDM frame, the numbers of arrival packets are x, y, and z for voice, video, and data, respectively. The z data packets enter the on-board FIFO downlink queue as shown in Figure 3.1. The TDM server first transmits real-time packets. If $(x + y)$ is larger than C, the excessive $(x + y - C)$ real-time packets are lost. Otherwise, the TDM server uses the remaining capacity of $(C-(x + y))$ time slots to send data packets in the front of the downlink queue. Jitter-tolerant traffic is lost when overflow occurs in the on-board queue. Again, in order to investigate the packet loss behaviour for different queue sizes, we assume an infinite queue in our simulation model, and examine the survivor function. As a means of improving the size of the on-board memory, we studied the cases where the memory is divided into separated blocks and where it is taken as a common block.

In the next chapter, we will establish the traffic scenarios used in the simulation of the system model. We will present the simulation results and discuss their impact on the system design.

CHAPTER 4
SIMULATION RESULTS

In this chapter, we will examine the effects of multimedia traffic mix and total traffic load on the satellite system performance. More specifically, we will be interested in the packet loss performance of the real-time traffic at three different points: (i) at the uplink, (ii) inside the on-board switching fabric, and (iii) at the downlink. As for the jitter-tolerant traffic, we present the survivor function of the uplink and downlink queue size as well as the packet loss performance inside the on-board switch. Moreover, we will study the effects of switch size and burstiness of traffic on both the performance of the switch and the performance of the downlink.

We will start our presentation by first describing the different traffic scenarios, and the particular values of the different parameters characterizing the voice, video, and data traffic models that were introduced in Chapter 3 and will be used in the simulation experiments.

4.1 Traffic Scenarios

The best way to explain our traffic scenarios is by first introducing the parameters α_{vo}, α_{vi}, and α_{d} (where $\alpha_{\mathrm{vo}} + \alpha_{\mathrm{vi}} + \alpha_{\mathrm{d}} = 1$). Denoting the aggregate average traffic arrival rate by λ, i.e.,

$$\lambda = \lambda_{\mathrm{vo}} + \lambda_{\mathrm{vi}} + \lambda_{\mathrm{d}} \tag{4.1}$$

we can define α_{vo}, α_{vi}, and α_{d} as

$$\alpha_{\mathrm{vo}} = \frac{\lambda_{\mathrm{vo}}}{\lambda},\ \alpha_{\mathrm{vi}} = \frac{\lambda_{\mathrm{vi}}}{\lambda},\ \alpha_{\mathrm{d}} = \frac{\lambda_{\mathrm{d}}}{\lambda} \tag{4.2}$$

Each one of these expresses the composition of voice, video, and data traffic in the overall traffic, respectively.

In reality the composition of admitted multimedia traffic varies depending on the applications requested by the end-users, resulting in an infinite number of combinations. In order to study the traffic mix, we chose to study four typical traffic composition scenarios, namely, (i) voice dominant, (ii) video dominant, (iii) data dominant, and (iv) equal load. These four traffic scenarios permit us to cover the cases where each type of service is predominently used, and a general case where all three services are equally utilized. Table 4.1 shows the corresponding values of α_{vo},

Scenario	voice (α_{vo})	video (α_{vi})	data (α_{d})
Data dominant	20%	10%	70%
Equal load	33.3%	33.3%	33.3%
Video dominant	10%	70%	20%
Voice dominant	70%	10%	20%

Table 4.1: Traffic scenarios

α_{vi}, and α_{d} for each traffic scenario. For example, the data dominant case implies that, on average, the aggregate arrival traffic to one uplink is composed of 20% voice traffic, 10% video traffic, and 70% data traffic [46].

To study the effect of traffic load, we define ρ (where the condition for system stability is $0 < \rho < 1$) as the normalized system utilization. We will consider three traffic utilization points for each traffic scenario: $\rho = 0.5$, 0.8, and 0.95.

Using Equations (4.1) and (4.2), as well as ρ and C (the beam capacity on the uplink or downlink as discussed in Chapter 3), we can alternatively express λ as

$$\lambda = \rho C = \frac{\lambda_{\mathrm{vo}}}{\alpha_{\mathrm{vo}}} = \frac{\lambda_{\mathrm{vi}}}{\alpha_{\mathrm{vi}}} = \frac{\lambda_{\mathrm{d}}}{\alpha_{\mathrm{d}}} \tag{4.3}$$

Unless otherwise specified, in the remainder of this thesis, we will assume that C is equal to 512 packets/frame. Furthermore, we will assume that the duration of a frame is 0.024 s. In other words, a frame will carry 512 packets every reoccuring 24 ms.

From Equation (4.3), we can obtain the average arrival rate of each type of traffic (i.e., λ_{vo}, λ_{vi}, and λ_{d}) in terms of their respective traffic composition (i.e., α_{vo}, α_{vi}, and α_{d}), ρ, and C, as follows,

$$\lambda_{\mathrm{vo}} = \alpha_{\mathrm{vo}}\rho C, \ \lambda_{\mathrm{vi}} = \alpha_{\mathrm{vi}}\rho C, \ \lambda_{\mathrm{d}} = \alpha_{\mathrm{d}}\rho C \tag{4.4}$$

We note that λ_{vo}, λ_{vi}, and λ_{d} correspond to the two values, given in Equation (3.4) and (3.6), we wanted to match in Chapter 3. Thus, we only need to know the value of IDC at infinity, peak packet rate, and peak-to-average ratio to obtain the four parameters of the 2-state MMPP. These values are measurements provided by Spar

Aerospace Limited, and are summarized in Table 4.2. Using ATM as an example (1 packet = 1 cell = 48 bytes), the peak rate for voice source is 64 kbps, and that for the video source is 384 kbps. The peak rate and peak-to-average ratio are hard to determine for data source, because of its bursty nature. As a consequence, they were not provided by Spar. Instead, for our matching purposes, we assumed a peak rate of 64 kbps (under ATM cell format) and peak-to-average ratio of 100. This will result in a large data user population with a high degree of variability in their transmission rates, a condition similar to the one we want to model.

Traffic Source	Peak rate (packets/s)	Peak-to-Average Ratio	IDC(∞)
Voice	$\frac{1000}{6}$	2.5	15.9
Video	1000	5	55.9
Data	$\frac{1000}{6}$	100	∞

Table 4.2: Input source parameters for various media

The aggregate peak rate in an uplink beam, A, can be obtained by summing the product of the peak-to-average ratio and the average rate of the three traffic sources, as follows,

$$A = \lambda_{\mathrm{vo}} R_{\mathrm{vo}} + \lambda_{\mathrm{vi}} R_{\mathrm{vi}} + \lambda_{\mathrm{d}} R_{\mathrm{d}} = 2.5\lambda_{\mathrm{vo}} + 5\lambda_{\mathrm{vi}} + 100\lambda_{\mathrm{d}} \tag{4.5}$$

We define the ratio $\mathrm{G_{SM}} = \frac{A}{C}$ as the statistical multiplexing gain. The term multiplexing refers, in general, to any technique which permits more than one independent user to share one physical facility. In our case, it is the number of multimedia users who can share the channels otherwise dedicated to each user in the case of circuit switching.

Using Equation (4.5), we can alternatively write

$$G_{SM} = \frac{2.5\lambda_1 + 5\lambda_2 + 100\lambda_3}{C} = \rho(2.5\alpha_1 + 5\alpha_2 + 100\alpha_3) \tag{4.6}$$

Table 4.3 shows the achievable G_{SM} for each traffic scenario under study.

Scenario	voice (α_1)	video (α_2)	data (α_3)	G_{SM}
Data dominance	20%	10%	70%	71.00ρ
Equal load	33.3%	33.3%	33.3%	35.83ρ
Video dominance	10%	70%	20%	23.75ρ
Voice dominance	70%	10%	20%	22.25ρ

Table 4.3: Achievable statistical multiplexing gain for each traffic scenario

However, G_{SM} introduces the problems of contention (resulting in packet loss) and queueing (resulting in delay, delay jitter, and queue size). We will report its effects on the performance of our system through simulation results when a 16×16 switch is used. The results of the system using a 4×4 switch will be shown to study the effects of switch size.

4.2 Uplink Performance

The uplink queue models the sum of all on-ground queues of users within the coverage area of a beam. With the assumption that each user is independent, we can consider one uplink queue to be representative of the ensemble of uplink queues. Since the server gives priority to the different types of traffic, the on-ground performance is determined by both real-time and jitter-tolerant traffic. For the uplink performance of real-time traffic, we look at the loss probability which is the probability that the arriving real-time packets exceed the uplink capacity (and thus are lost). Meanwhile, for the uplink performance of jitter-tolerant traffic, we investigate the average queue size and the survivor function.

4.2.1 Loss probability of real-time traffic

As previously discussed, real-time traffic loss occurs when the number of real-time packets arriving during a frame exceeds the uplink beam capacity, since no queueing is provided to real-time traffic. This translates into finding the probability that the number of arriving packets exceeds $C = 512$ packets/frame. We consider the models used for voice and video traffic. Consistently in all traffic scenarios, 2-state MMPP's were used to generate real-time traffic. We assume the average arrival rates for each 2-state MMPP to be λ_{ij} and the transition rates to be σ_{ij} (where $i = 1$ for voice source, $i = 2$ for video source, $j = 1$ for underload state, and $j = 2$ for overload state), the resulting real-time traffic yields a 4-state MMPP, with each state having the average arrival and transition rates as shown in Figure 4.1. In addition, we assume transitions between states to occur only at the frame edges (beginning or ending of a frame). This is a valid approximation since all mean sojourn times are larger than one time frame. Then, for the duration of a frame, the arrival rate is equal to the Poisson rate of the state the process is in. The probability of excess of a Poisson distribution is known and expressed by

$$P_i(a > 512) = 1 - \sum_{j=0}^{512} \left(\frac{\lambda_i^j e^{-\lambda_i}}{j!}\right) \tag{4.7}$$

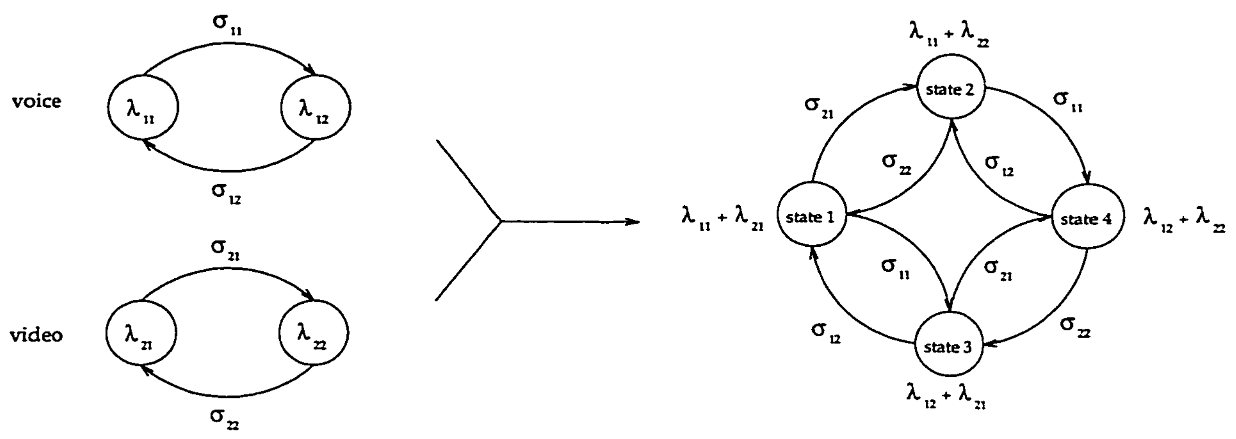

Figure 4.1: Model of aggregate real-time traffic

where P_i is the probability of excess in state i, a is the number of arrivals, and λ_i is the average arrival rate in state i.

Knowing the probability of excess in each state i, we only need to find the probability that the process is in state i to obtain the loss probability of real-time traffic, given by

$$P_{\text{loss}} = \sum_{i=1}^{4}(\gamma_i P_i) \tag{4.8}$$

where γ_i is the probability of state i, and P_i is defined in Equation (4.7).

We follow the procedures in [44] to obtain the probability γ_i of each of the four states of the aggregate MMPP. Table 4.4 shows the probability of loss in each state according to the average arrival rate in that state, and the state probability for the voice dominant traffic scenario at 95% utilization. From the values in Table 4.4 and Equation (4.8), P_{loss} for the voice dominant case at 95% load is 5.3169 $\times 10^{-6}$.

Similarly, we have calculated the loss probability for the other utilization points and traffic scenarios. Table 4.5 summarizes all the results for real-time traffic. As expected, the loss probability increases as the overall traffic becomes more dominated

State i	λ_i (packets/s)	P_i	γ_i
1	14,825	1.6488×10^{-12}	0.2922
2	17,024	3.6864×10^{-7}	0.2627
3	15,551	6.0705×10^{-12}	0.2388
4	17,750	2.3675×10^{-5}	0.2053

Table 4.4: Real-time traffic state average arrival rate, probability of loss in that state, and the state probability for voice dominant case at 95% utilization

Traffic Scenario	ρ		
	0.5	0.8	0.95
Data dominant	$< 10^{-14}$	$< 10^{-14}$	$< 10^{-14}$
Equal load	$< 10^{-14}$	9.2524×10^{-13}	1.6692×10^{-12}
Video dominant	$< 10^{-14}$	1.6514×10^{-12}	3.0806×10^{-6}
Voice dominant	$< 10^{-14}$	1.6808×10^{-12}	5.3169×10^{-6}

Table 4.5: Real-time traffic loss probabilities of all traffic scenarios and load cases

by real-time packets. Due to the CPU limitations of the computers (SPARC 5, 10, and 20) performing these simulation runs, we have recorded no packet loss in a sample population of 10^7 packets. This was true even for the voice dominant case at 95% utilization, in which we have calculated the loss probability to be 5.3169×10^{-6}. These results show that, at a confidence level of 95%, the uplink can support services requiring a loss probability of 10^{-5} for all traffic compositions.

In the next section, we will discuss the queue size requirement on-ground for jitter-tolerant traffic.

4.2.2 Queueing of jitter-tolerant traffic

Unlike real-time traffic, jitter-tolerant traffic can sustain delay, but it is loss sensitive. Therefore, queueing is required on-ground. In order to study the queue behaviour, we assumed an infinite queue and examined the survivor function (i.e., the probability that the queue length exceeds a threshold value). Mathematically, we can express the survivor function in the following way; let $q(x)$ be the probability density function of the number of packets in the queue, and $Q(x)$ be the cumulative probability function of $q(x)$ ($Q(x) = \int_{\infty}^{x} q(t)dt$), then the survivor function is $S(x) = 1 - Q(x)$.

Figure 4.2 shows the simulation results for the traffic scenarios, in which data packets were assumed to be generated by a 2-state MMPP (short-range dependent process). We observe that as the traffic becomes dominated by data applications, the queue size required has to be increased to maintain the same loss probability. This is shown through Figure 4.2 where the data dominant case required more queueing than the equal load, video dominant, and voice dominant cases (in descending order of

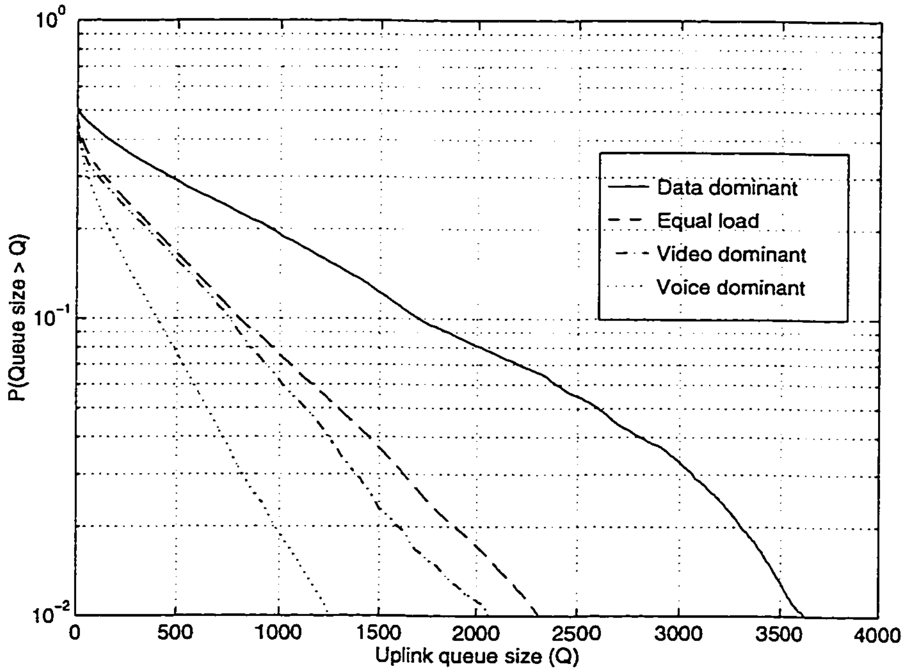

Figure 4.2: Survivor function of the on-ground uplink queue at 95% load when MMPP model is used for data traffic

queue size requirement). The two real-time dominant scenarios presented the lowest occupancy since they were both composed of 80% real-time packets and 20% jitter-tolerant packets. Despite the fact they had similar G_{SM} (23.75ρ for video dominant and 22.25ρ for voice dominant), the video dominant case has a longer queue size since video sources are burstier than voice sources.

We also took into account the burstiness of data sources by studying the survivor function when a 2-state PMPP (long-range dependent process) is used to model data traffic [40]. For PMPP, H is used as a measure of the level of burstiness (the higher the value of H, the burstier the traffic). Figure 4.3 depicts the survivor function of all traffic scenarios when data traffic is modeled by a 2-state PMPP. Comparing the queue values, at the same probability of excess, of Figure 4.3 with Figure 4.2, we

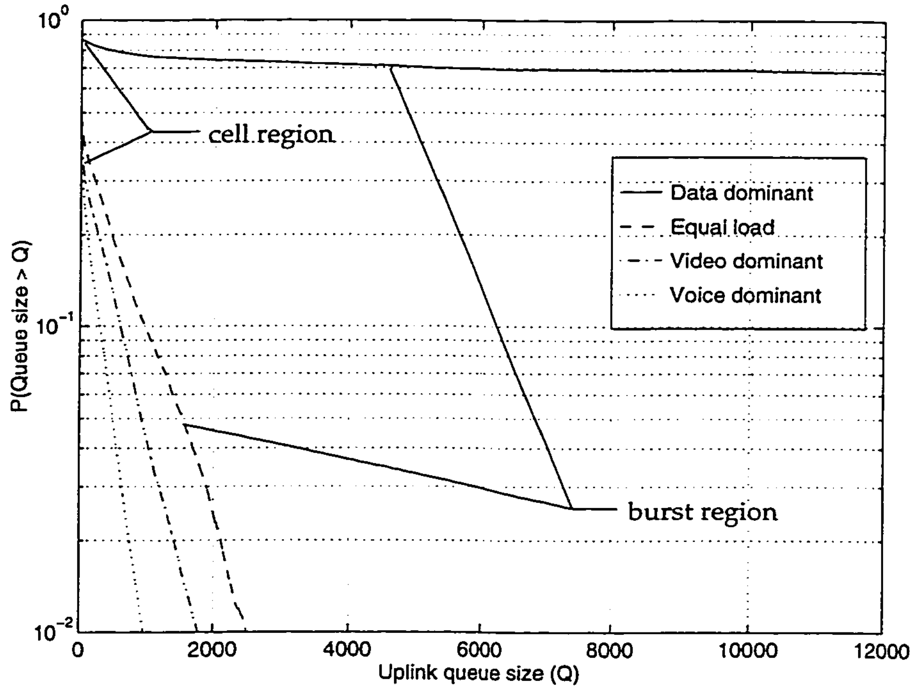

Figure 4.3: Survivor function of the on-ground uplink queue at 95% load when PMPP model is used for data traffic

observe that when data composition is low (as in the cases of voice dominant, video dominant, and equal load) the queue size required to support long-range dependent traffic is larger than that required to support short-range dependent traffic, but not by a significant number. However, when data composition is high (as in the case of data dominant) we can clearly see that the nature of jitter-tolerant traffic has a strong impact on the size of the uplink memory (Figure 4.4).

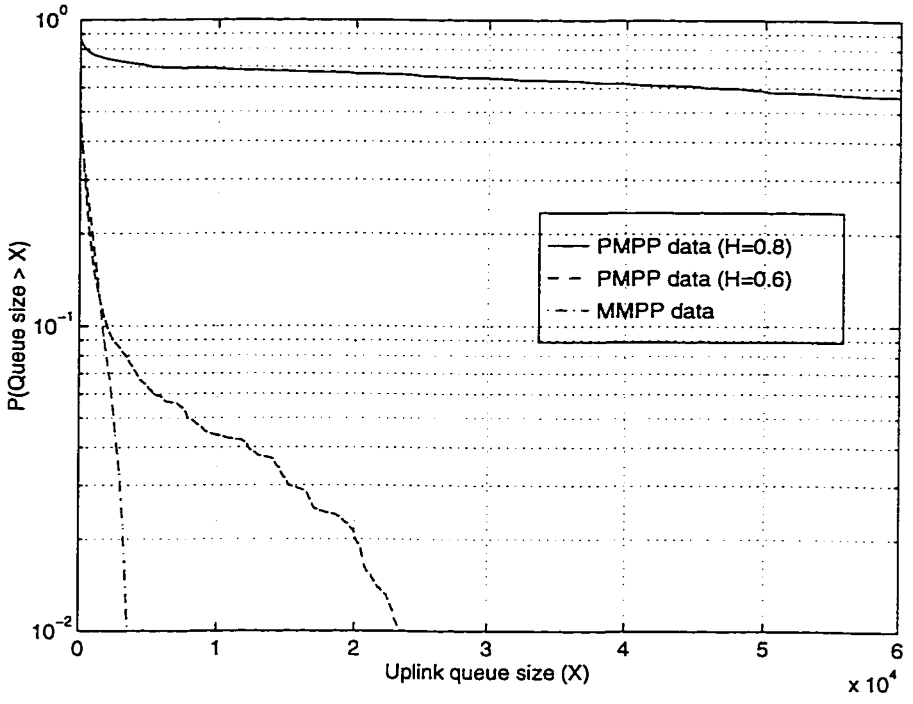

Figure 4.4: Survivor function of the on-ground uplink queue for data dominant at 95% load

A closer look at the PMPP curves (Figure 4.3) reveals two slopes or regions. These regions, as defined in [2], are the cell region and the burst region. They are a result of a 2-state process (MMPP or PMPP). It is worth noting that the burst region has a profound effect on the design of the queue, since it implies that after a certain threshold any further increase in the size of the queue will not improve the performance (i.e., reduce loss probability). From the designer perspective, this means that the queueing of jitter-tolerant traffic alone cannot always satisfy the QoS requirements of the end-user applications.

Uplink Capacity

One method to alleviate the queueing requirement is to increase the beam capacity [20]. Figure 4.5 shows the probability of excess as a function of the normalized queue size required to support data dominated traffic utilizing 95% of the capacity, for $C = 64$, 256, 512, and 4096. This is a good solution, but it has its constraints. To a certain extent, higher capacity can require the use of higher frequency bands which will always be limited by the available technological equipment.

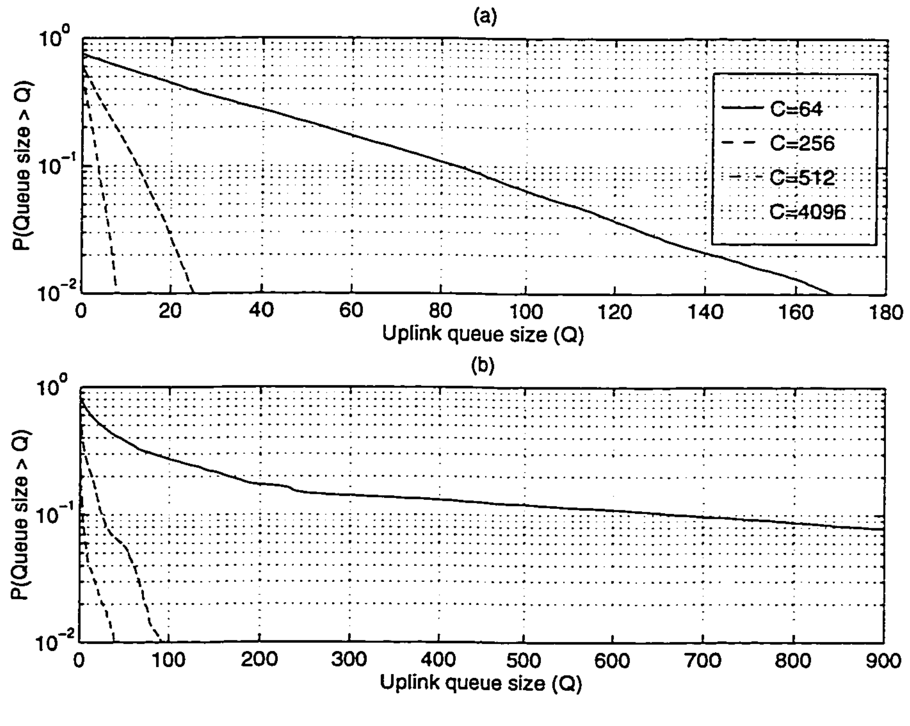

Figure 4.5: Survivor function of on-ground uplink queue for data dominant at 95% load as a function of capacity C: (a) MMPP data traffic and (b) PMPP data traffic

In the next two sections, we will move from the on-ground terminals to the on-board components and study the on-board performance of the switch (in Section 4.3) and the downlink (in Section 4.4).

4.3 On-board Packet Switch Performance

In this section, we discuss the performance of the on-board switch, which is measured in terms of packet loss probability. We considered two types of switch: the shared-memory and the Knockout switch.

4.3.1 Shared-memory switch

In the case of the shared-memory switch, we assumed that all packets arriving at the switch input ports can be written into the output memory (downlink). Hence, no contention needs to be resolved and the shared-memory switch is assumed to have no packet loss due to contention. This leads to larger on-board memory size requirements which will be discussed in section 4.4.

4.3.2 Knockout switch

In a Knockout switch, only L out of the possible N contending packets can be switched at every time slot and hence, the exceeding packets are lost [39].

We assumed that the on-board scheduler randomly assigns the slots of an MF-TDMA frame and, thus, there is an equal chance that a packet occupies any slot of the frame. In addition, we assume each spot beam population to be large enough that each packet is equally likely to choose any of the N available output ports (cell-level destination distribution).

Figure 4.6 shows the simulation results of the packet loss probability for various values of L and for different loads. Since no priority was given at the switch, Figure 4.6 is representative of both real-time and jitter-tolerant packets. We notice that the results are the same for both models of data traffic, MMPP and PMPP. Furthermore, they are equal to those computed by using the following analytical expression [39]

$$P[\text{packet loss}] = \frac{1}{\rho} \sum_{\text{k}=\text{L}+1}^{\text{N}} (\text{k} - \text{L}) \begin{pmatrix} N \\ k \end{pmatrix} \left(\frac{\rho}{\text{N}}\right)^{\text{k}} \left(1 - \frac{\rho}{\text{N}}\right)^{\text{N}-\text{k}} \tag{4.9}$$

where ρ is the probability that a packet occupies a time slot.

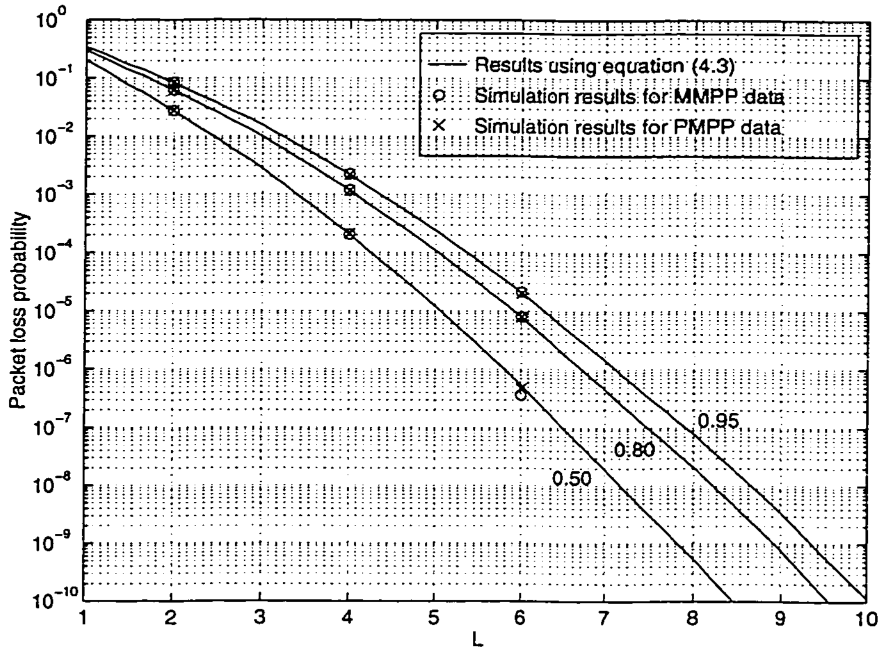

Figure 4.6: Packet loss probability vs. L of the on-board $16X16$ packet switch with load as a parameter

The fact that there was no difference in terms of performance when the different data traffic models were used shows that the burstiness of the traffic is greatly reduced once it has reached the on-board switch. This is explained in part by the framing structure which effectively limits the number of arrivals at one port to C in every frame. However, the significant factors which influence the traffic burstiness on the on-board switch are the size of the switch (assuming a fair use of the switch output, the larger the size of the switch, the fewer packets are likely to contend) and the packet destination distribution (when one output is in more demand than the others).

Effects of switch size

We observe in [39] that the size of the switch affects the packet loss probability and hence the on-board performance. In fact, since the probability that a large number (close to N) of arriving packets during a slot requesting the same output port is very small, and since packet loss in the system is inevitable, we can fix the number of accepted packets L ($L < N$) at an output port in such a way that the quality-of-service is not affected too much. This value was found to be around $L = 8$ for large values of N [39]. Our problem in obtaining simulation results for large systems is due to CPU limitations of the computer. Thus, instead of showing the benefits of increasing N, we have decided to show the drawbacks of reducing N. For this, we have obtained results for the same satellite system at $N = 4$. Figure 4.7 depicts the simulation results. Again, we observed the same packet loss probability as found in [39]. Given our simulation results for $N = 4$ and $N = 16$, we are inclined

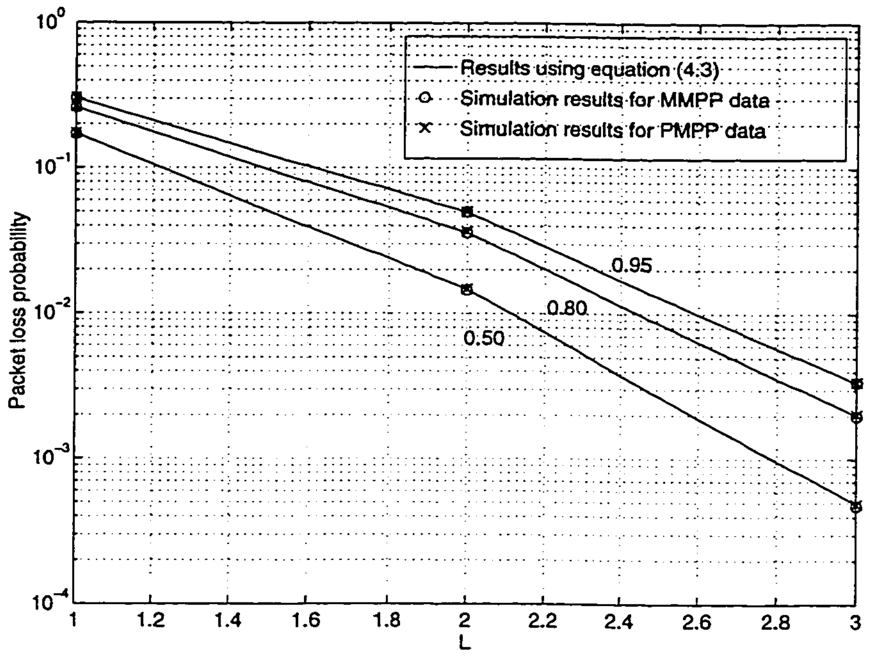

Figure 4.7: Packet loss probability vs. L of the on-board 4×4 packet switch with load as a parameter

to believe that the results in [39] are applicable in determining the on-board switch performance for any N. Consequently, increasing the size of the switch will improve the performance.

Effects of packet destination distribution

In previous discussions, we have assumed a large user population sharing the system in such a way that packets arriving at the switch seem to have an equal chance of selecting any output port (cell-level distribution). We will now consider a case where the resulting input traffic, seen at the switch, is not uniformly distributed (i.e., a burst-level destination distribution). The 4×4 system, previously used to study the size of the switch, will be utilized to assess the effects of packet destination distribution. This will also give us an idea as to what extent it affects the performance in [39].

The burst-level destination distribution presents the highest correlation in packet destination since all packets from the uplink choose the same destination for the length of the burst. Of course, it is assumed that each burst has an equal chance of choosing any of the N output ports. It is applicable when the user population in a spot beam is small or when a destination becomes a "hot spot" for the spot beam user population. This kind of dynamic hot spot traffic should be differentiated from the static one: more attention should be paid to burst-level destination distribution, since the destination area changes with the beginning of a new burst (hence the name dynamic hot spot) making it hard to predict where the highly demanded port will be next. Meanwhile, in a static hot spot (not covered in this thesis), the highly demanded port is fixed and known. Figure 4.8 shows the packet loss probability at each output, for the data dominant scenario at 50% utilization and using the PMPP model for data traffic. Unlike the cell-level packet destination distribution cases, we see that there is a noticeable difference in the packet loss probability at each output port. In general, if the model used is short-range dependent, the resulting packet loss probability is not affected by the packet destination distribution (as shown by the real-time packet loss behaviour). However, if the model used is long-range dependent, the hot spot will show differences in packet loss probability depending on whether the output port is under-used or over-used (as shown by the jitter-tolerant packet loss behaviour). Overall, the sum of all losses at all the ports can still be determined by Equation (4.9). This is because while the hot spot has changed location, the overall traffic load remains the same. From these results, we see that as the correlation of

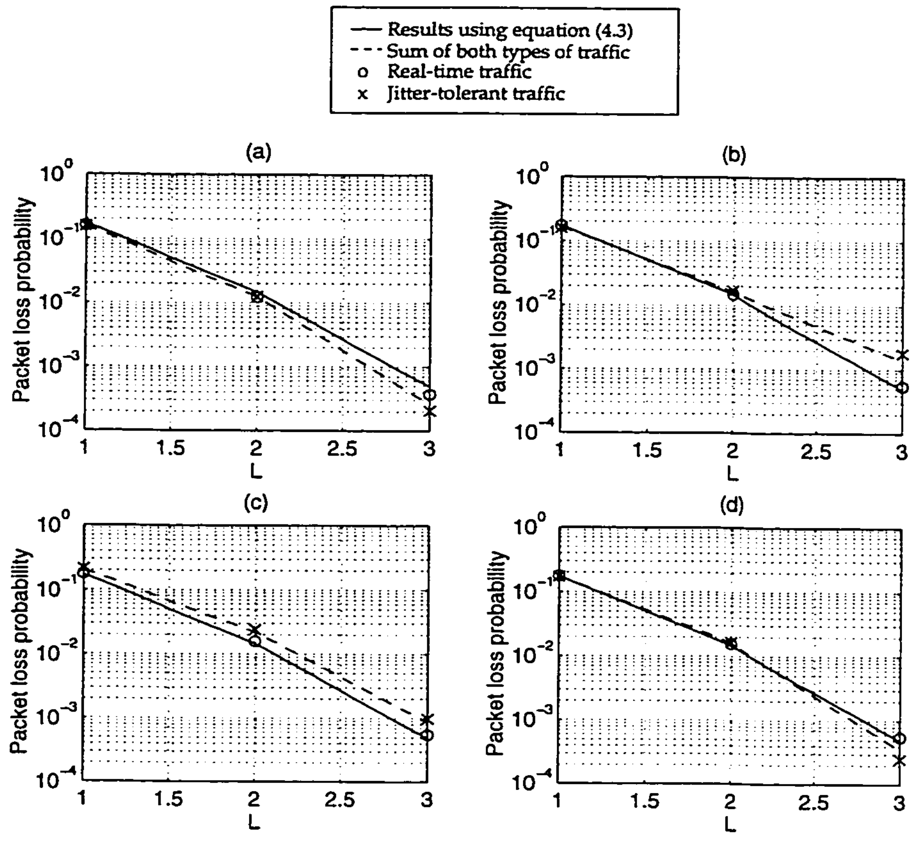

Figure 4.8: Packet loss probability vs. L of the on-board packet switch in a hot spot environment: (a) at port 1, (b) at port 2, (c) at port 3, and (d) at port 4

packet destination address increases, we have to pay more attention to the individual output ports if the input traffic shows strong correlation. The effects of the burst-level destination distribution are much greater at the downlink as we shall see in the next section.

4.4 Downlink Performance

Switched packets will require queueing prior to being transmitted to the downlink beam in a TDM manner. Similar to the uplink, the downlink has a maximum capacity of $C = 512$ packets/frame, and gives priority to real-time packets over jitter-tolerant packets. However, unlike the on-ground queues, on-board queues can be organized as separated FIFO memory blocks (one for each output port) or can be combined in one single common memory block. As discussed in the previous section, the downlink queues of the shared-memory switch will have the most packets to store since none are lost due to contention. We will discuss the memory requirement of this type of switch first. Then, we will study the effects of the Knockout scheme on the queue size. All the survivor function figures in this section show the total queue size required for all N downlink ports.

It is worth noting that unlike the uplink, we cannot derive an analytical expression for the loss probability at the downlink. While the real-time traffic at the uplink is known to be generated by a 4-state MMPP, the real-time traffic at the downlink is a combination of the split traffic of all the uplinks, of which the process is not known. We have plotted the probability density function of the real-time traffic arriving at the downlink queue in Figure 4.9 for the video dominant scenario at 80% utilization when the PMPP model is used for data traffic [1]. In addition, we have provided the Poisson curve with average arrival rate equal to that obtained from simulation for the arriving real-time traffic at the downlink. We note that the probability density function of the arrivals at the downlink resembles that of a Poisson process or an MMPP with λ's very close to each other in value. Further research will need to be done to assess the validity of this approximation model and to match the significant parameters of the traffic. When a model of the downlink traffic is available, the performance of the downlink queue will be at our immediate disposal without the

[1]Note that the probability density function of the arriving real-time traffic at the downlink for the same scenario using, instead, an MMPP model for data generation, will be the same as that given in Figure 4.9 since real-time traffic service is unaffected by jitter-tolerant traffic

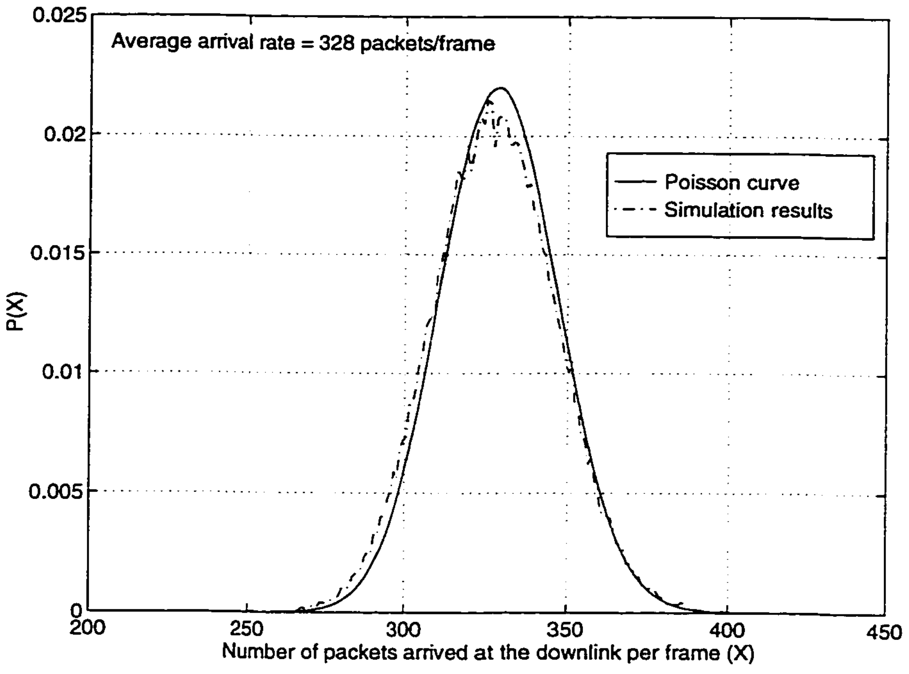

Figure 4.9: Probability density function of the number of packets arrived at the downlink every frame

need of simulating the satellite system. Until then, we will rely on simulation results to determine the downlink performance. From simulations, we see no real-time packet loss at the downlink in a sample population of 10^7 packets. Then, at a confidence interval of 95%, we can say that the loss probability is less than 10^{-5}.

Figure 4.10 shows the queue size required for each traffic scenario when data traffic is modeled by a 2-state MMPP and when the shared-memory switch is used. We note that the bursty nature of traffic has been greatly reduced on-board since all four curves are very similar. There is a saving factor of about 3 when a common memory block is used instead of separated queues. As was the case in the uplink, when data composition is low (as in the voice dominant, video dominant, and equal load scenarios), the downlink queue size is not much affected by the correlation of traffic. Hence, the queue size found in Figure 4.10 to support a given probability of excess

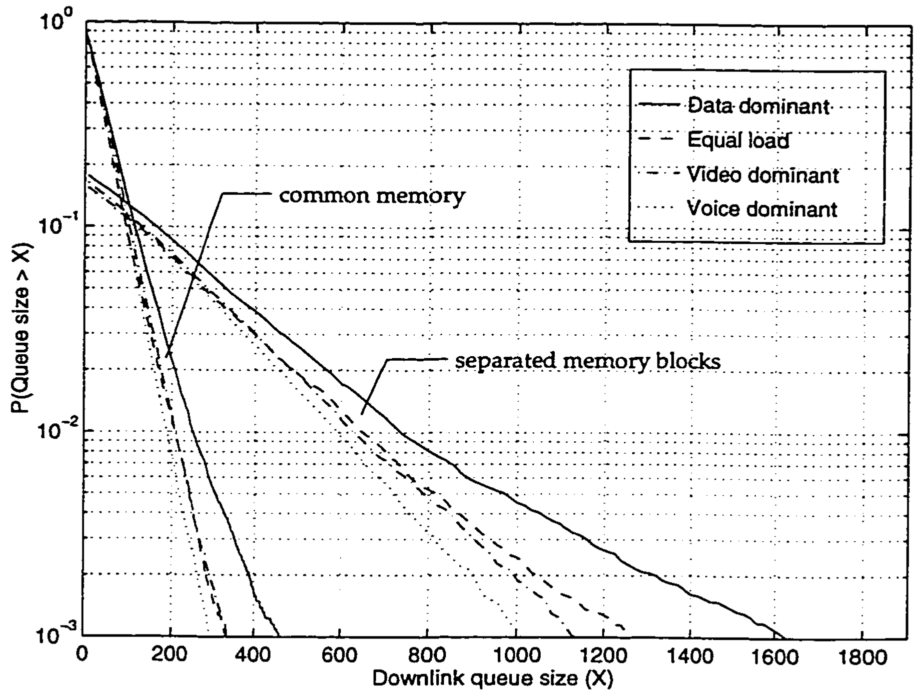

Figure 4.10: Survivor function of the on-board downlink queue at 95% load when MMPP model is used for data traffic

when data traffic is generated by MMPP, can also support the same probability of excess when PMPP is used for all scenarios except the data dominant one. Figure 4.11 depicts the shared-memory queue size required when traffic is dominated by data at varying degree of traffic correlation. We observe a similar behaviour as that found at the uplink (discussed in section 4.2), but the effect is less prominent. We can thus use the same reasoning as elaborated in section 4.2 which is, in the long run, queueing of long-range dependent traffic is not a good solution. We will discuss this subject in the next section.

Finally, we see the effect of the Knockout scheme on the amount of queueing required on-board in Figure 4.12. Obviously, as L approaches N, the queue size approaches that found for the shared-memory switch where there is no contention loss (which is $L = N$ in the Knockout scheme). This value of L is found to be $L = 6$

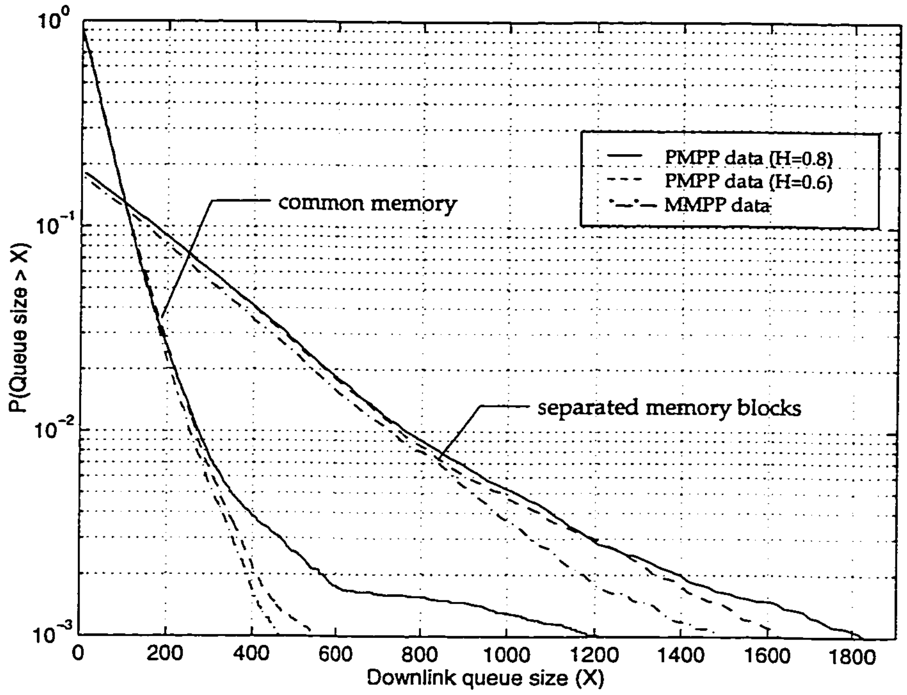

Figure 4.11: Survivor function of the on-board downlink queue for data dominant at 95% load

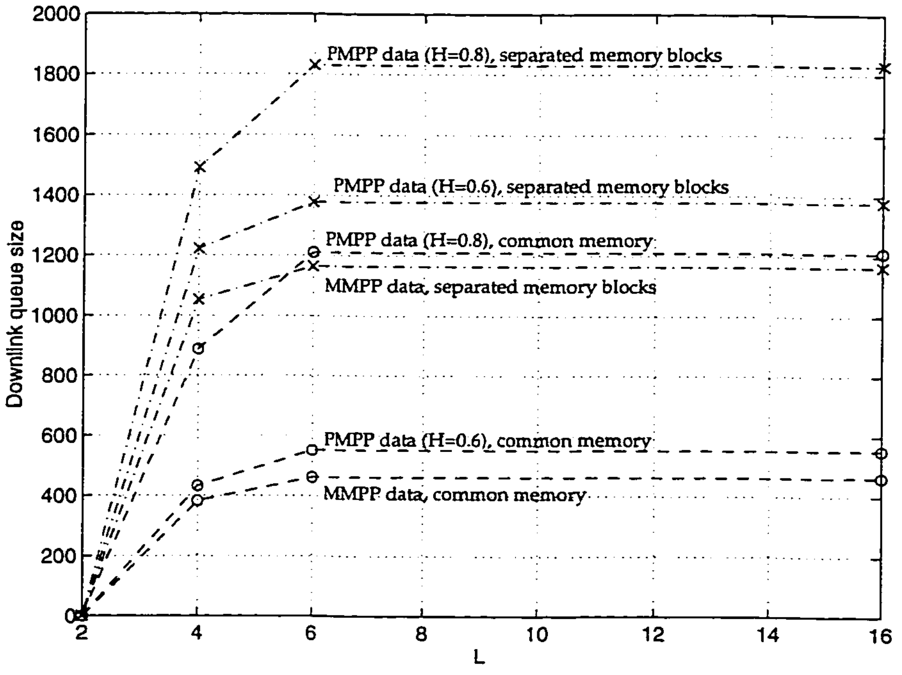

Figure 4.12: Downlink queue size vs. L for data dominant at 95% load (for a probability of excess of 10^{-3})

from Figure 4.12. We will see the effects of the switch size and the packet destination distribution on the downlink queue in the next two subsections.

Effects of switch size

Figure 4.13 shows the average individual on-board queue size required in a 4×4

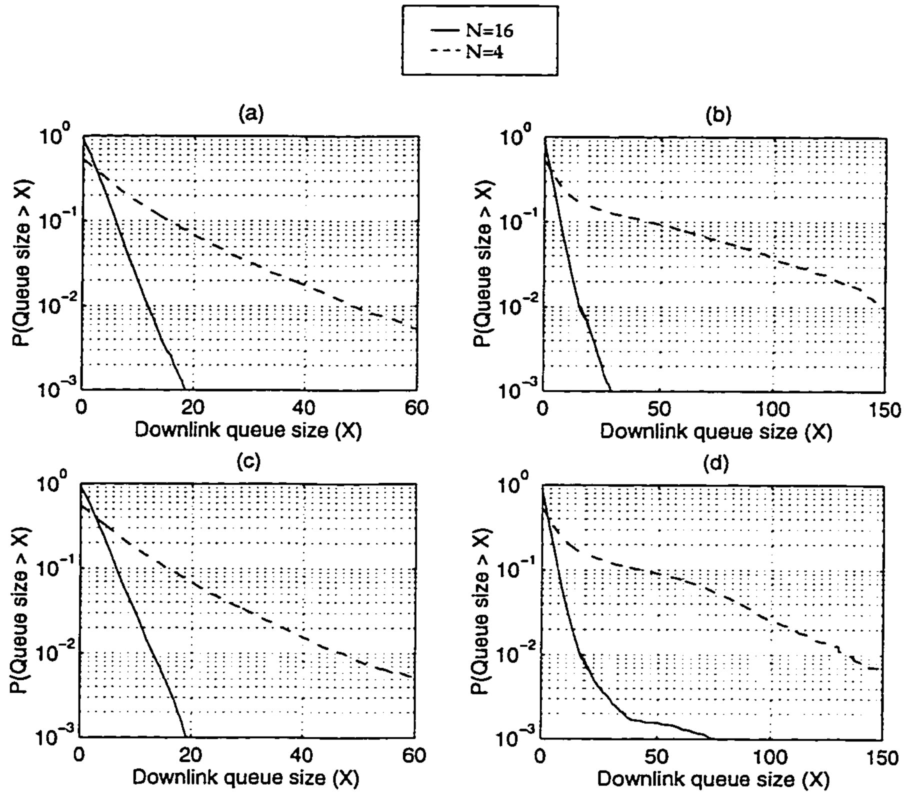

Figure 4.13: Survivor function of the average individual on-board downlink queue at 95% load as a function of the switch size: (a) voice dominant with MMPP data, (b) data dominant with MMPP data, (c) voice dominant with PMPP data (H=0.8), and (d) data dominant with PMPP data (H=0.8)

system as compared to a 16×16 system for the two extreme cases of data composition, namely, the voice dominant and the data dominant. Since we assumed the destination address is chosen following a cell-level distribution, the correlated uplink traffic is randomly split to each downlink, resulting in a reduction of the burstiness of the downlink traffic. As the size is increased, we should expect to see further reduction in the correlation of the downlink traffic, improving the queue size requirement on-board. This is depicted through Figure 4.13, as we see the individual memory size required to maintain the same level of loss probability actually decreases when we change the size of the switch from $N = 4$ to $N = 16$. In the next subsection, we will see the effects of burst-level destination distribution on the downlink queue size.

Effects of packet destination distribution

As in section 4.2, we use the results of the 4×4 system under burst-level destination distribution. Figure 4.14 plots the survivor function of the individual on-board downlink queue for the voice and data dominant scenarios at 50% utilization when data is generated by MMPP. When compared to the queue size required for the same cases under cell-level packet destination distribution (where we observed no queueing for all scenarios with 80% or lower load), there is a significant increase in the memory size due to the correlated nature of the packet destination distribution. This increase in queue size is large, and, even though we expect a smoothing effect in a larger switch system, the presence of the burst region in the 4×4 switch at a low probability of excess in the survivor function indicates that queueing will not help to improve the performance. In addition, we observe significant real-time packet losses that we did not see in a cell-level destination distribution.

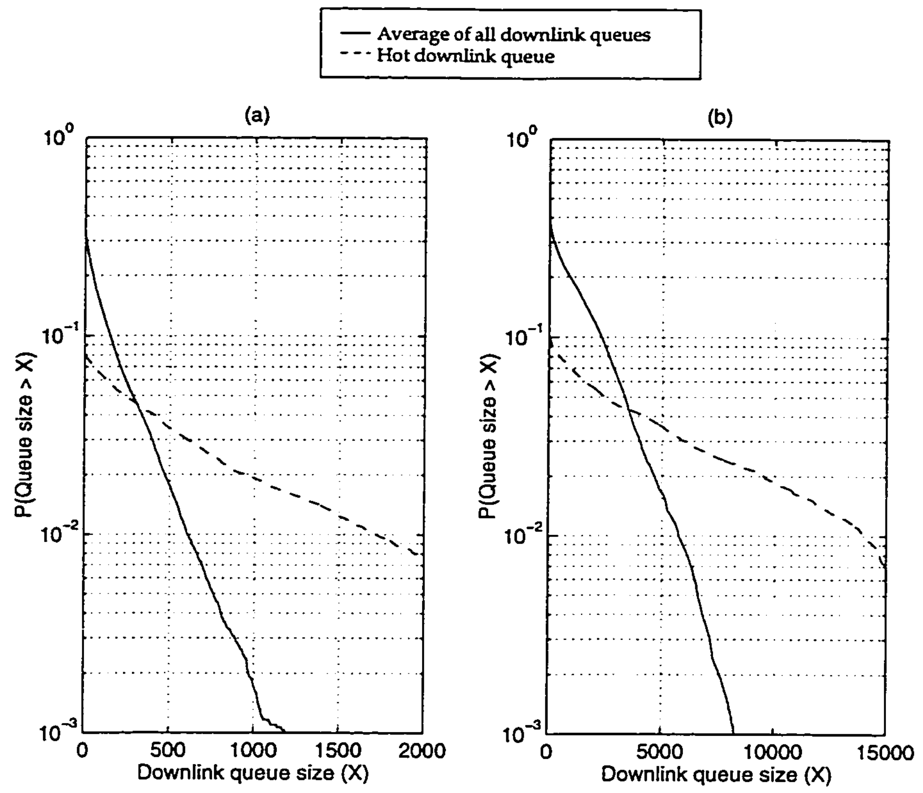

Figure 4.14: Survivor function of the individual on-board downlink queue at 50% load when data traffic is modeled by MMPP under a burst-level packet destination distribution: (a) voice dominant and (b) data dominant

4.5 Discussions

The simulation results presented in sections 4.2, 4.3, and 4.4 can be divided into the on-ground part and the on-board part. The concerns related to each part are the subjects of the following two subsections.

4.5.1 On-ground Concerns

It can be observed that the significant factors on-ground affecting the system performance are the traffic composition and the nature of the traffic. Evidently, the uplink can support real-time traffic without compromising the quality-of-service (maximum observed loss probability of 10^{-5}). However, jitter-tolerant traffic can cause overflow in the uplink queue, and hence results in data loss. On the other hand, since jitter-tolerant data sources are delay insensitive, the network manager is always inclined to accept more data users in the system to maximize the resource utilization in order to achieve high statistical multiplexing gains. As the composition of the traffic becomes more data dominated, and the data traffic becomes more correlated, the queue performance, as seen through the survivor function, seems to level off very quickly. Within this region, which is defined in [2] as the burst region, a very small improvement in the loss probability will require a large increase in the queue size. Consequently, particular attention must be paid to the nature of data traffic.

In the long run, the solution to the queueing problem can be found through observations made in section 4.2.2 . We saw that when traffic is short-range dependent, the burst region of the survivor function occurs at a lower loss probability, and the cell region, where an increase in the memory will effectively improve the queue performance, is considerably extended. Hence, when highly correlated traffic needs to be supported, an intelligent congestion scheme that shapes the traffic can be used to break the long-range dependency nature, and scatter long bursts throughout the frames, yielding results similar to, or even better than, those of a short-range dependent traffic.

4.5.2 On-board Concerns

The performance on-board is dependent on the size of the switch, the packet destination distribution, traffic composition and nature of the traffic. We saw that once packets have reached the on-board switch, the burstiness of the traffic is greatly reduced. This can be explained in part by the framing structure incorporated at the uplink and the random splitting of traffic at the switch, which imposes an upper limit to the size of the burst in every frame and scatters the burst to different outputs, respectively. This results in the performance of the on-board switch being comparable to the one reported in [39]. At the downlink, we have noted that the reduced burstiness has alleviated the memory size requirement. The on-board memory can be further reduced by using a common memory block organization, forcing the downlink ports to share the storage area. The resulting memory size would be reduced by up to 3 times.

Combining the switch and downlink results, we notice that there is no advantage to increasing the switching capacity L. In fact, from Figures 4.6 and 4.12, we can guarantee a packet loss probability of 10^{-6} with $L = 8$, and with the same required queue size as when using $L = N$ (shared-memory switch). This advantage becomes more apparent as we increase the size N. However, if the uplink traffic is highly correlated, on-board queueing may not be a good solution. In order to study the impact of traffic burstiness on-board, we studied the effects of switch size and packet destination distribution.

Assuming a fair use of each of the switch output, a larger switch will smooth the traffic and improve the performance at both the switch and the downlink. However, in the case where a port is in higher demand, particular attention should be paid to individual ports as the hot spot will present far worse performance than what we observe on average. This can be clearly seen at the downlink, which demonstrates the true need for a good congestion control scheme.

4.6 Summary

From the results presented in this chapter, it can be observed that the prominent point is a need for a good traffic control scheme to reduce the burstiness of the traffic arriving at the satellite system. This will help reduce both the memory size and the delay at the uplink and downlink. Tables 4.6 and 4.7 summarize the simulation results of our 16×16 satellite system. For simplicity, we only present two scenarios, namely, data dominant and video dominant. Since the results for the voice dominant and the equal load scenarios are not significantly different from the video dominant one, we have only presented the video dominant and data dominant cases in Tables 4.6

Traffic Scenario and Load	Total Average Delay (frames)	Standard Deviation of Delay (frames)	Queue Size Required		
			Uplink	Downlink	
				Separated	Shared
Video dominant at 50%	2.50	0.2887	0	0	0
Video dominant at 80%	2.50	0.2887	0	0	0
Video dominant at 95%	4.42	3.0447	2080	1054	339
Data dominant at 50%	2.50	0.2887	0	0	0
Data dominant at 80%	2.50	0.2893	0	0	0
Data dominant at 95%	5.21	5.0466	3635	1492	464

Table 4.6: Summary of the simulation results obtained for the different performance measures when MMPP model was used for data

Traffic Scenario and Load	Total Average Delay (frames)	Standard Deviation of Delay (frames)	Queue Size Required		
			Uplink	Downlink	
				Separated	Shared
Video dominant at 50%	2.50	0.2889	0	0	0
Video dominant at 80%	2.50	0.2894	0	0	0
Video dominant at 95%	5.14	4.9065	1836	1495	394
Data dominant at 50%	2.50	0.2889	0	0	0
Data dominant at 80%	2.51	0.2897	0	0	0
Data dominant at 95%	26.88	95.5778	$> 280,000$	1831	1210

Table 4.7: Summary of the simulation results obtained for the different performance measures when PMPP model was used for data

and 4.7.

In Tables 4.6 and 4.7, the total average delay and its standard deviation are normalized and given in terms of frames. These values do not take into account the medium propagation delay, and hence they are applicable to any environment. Furthermore, they assume that there is no loss due to contention at the switch and that all queues are of infinite size. Hence, for a Knockout switch with (or without) finite queues, these values will be lower. We note that the minimum average delay through the system is 2.50 frames, which is the constant amount of delay seen by the traffic with higher priority (i.e., real-time traffic). The minimum standard deviation of delay is 0.2887 frame and, again, is a constant for real-time traffic.

The queue size shows the memory requirement, in terms of packets, of one port in order to maintain a loss probability of 10^{-2}, in the case of the uplink, or 10^{-3}, in the case of the downlink. Thus, given the packet size in bits and the transmission speed in bits/s, we can find the memory size required in bits. For example, an ATM cell (48 bytes = 384 bits) in a $C = 512$ packets/frame = 196,608 bits/frame (alternatively, $C = 8.192$ Mbps) environment will require the designer to multiply the normalized value in the table by 196,608 to get the memory size in terms of bits per frame. We note that for the data dominant case at 95% load using PMPP data traffic, 10^7 samples were not enough to obtain a queue size at the uplink.

Tables 4.6 and 4.7 did not show the packet loss probability at the switch since it is readily available through [39] (Figure 4.6). When we consider the overall packet loss probability of the system, we see that $L = 8$ is enough to guarantee a packet loss probability of 10^{-6} at the switch. By increasing L to 12, the packet loss probability is reduced to less than 10^{-10}. This loss is less than that due to the uplink and downlink. This implies that the loss probability is not limited by the switching capacity L, but instead it is determined by the choice of the memory size on-ground and on-board. The memory size, as we have discussed, can be further reduced without affecting the packet loss probability through a good congestion control scheme. As a result, we do not need to provide a large switching capacity to meet the required QoS (L around 8 is enough) which makes the Knockout architecture attractive for on-board switching. In addition to the level of performance attainable by the Knockout switch, it also has key features such as a modular growth, easy maintainability, fault tolerance, and lower complexity [39], that make it an excellent candidate for on-board switching.

Finally, the numbers under G_{SM} in Tables 4.6 and 4.7 represent the number of users we can potentially multiplex (given their characteristics at the input) into a line required to support one user in a circuit switched environment. We observe that high G_{SM} is related to large data user population, large queue size and delay, and it can get out of control when the data traffic is highly correlated. Therefore, the on-board packet switch system is ideally suited for a traffic dominated by real-time applications. As we move towards a traffic dominated by jitter-tolerant applications, the satellite system becomes more complex and expensive to realize (in terms of memory and mass), unless a congestion control that can shape the traffic is used. It is only then that the on-board packet switch can fulfill its promise of high statistical multiplexing gain.

CHAPTER 5
CONCLUSIONS

Satellite communications systems have the potential of effectively and economically providing multimedia services to a world-wide user population. In fact, they are the only true wireless solution capable of global coverage at a distance-insensitive cost. With the rapid advances in the field of telecommunications, broadband integrated applications are becoming a reality, and broadband networks are being deployed to accommodate the ensuing traffic.

Future generation satellite communications systems will have to be designed with broadband multimedia services in mind. In order to increase the bandwidth to meet the growing user demand, a multiple spot beam configuration is required for the satellite. Although a multiple spot beam satellite system has a much larger overall capacity and improved transmission aspects, it necessitates full connectivity between earth-stations. An on-board packet switch in conjunction with an efficient uplink access scheme can fulfill this function, while insuring an efficient utilization of the space segment and maintaining an acceptable QoS.

In this thesis, we have studied the performance of a satellite-switched system in a multimedia environment to support future trends in broadband services. The driving force behind the choice of packet switching over circuit switching is the high achievable statistical multiplexing gain (G_{SM}). However, G_{SM} comes at a cost in terms of packet loss, average delay, queue size requirement and, in the case of satellite systems, complexity. Our obstacle in obtaining the performance measures is the nature of multimedia traffic. We have considered 2-state MMPP's to model the aggregate traffic of voice and video, and two models for the aggregate data traffic, namely a 2-state MMPP and a 2-state PMPP. We used two data models in order to study the correlation of data traffic, a topic raised by recent findings [28]. In order to examine the effects of multimedia traffic mix and total traffic load on our satellite system performance, we have considered four traffic scenarios in conjunction with three load points.

Simulations of the system revealed that the satellite-switched system can easily accommodate traffic dominated by real-time applications requiring loss probability of 10^{-5}. As the traffic becomes dominated by jitter-tolerant applications, and data traffic are more correlated, the average delay grows significantly, and the size of the uplink and downlink queue needs to be increased if acceptable QoS is to be maintained. In fact, we observed that both on-ground and on-board queues can require a large amount of memory. The problem aggravates when output ports become hot spots, and these hot spots change with time. We have considered varying the beam capacity in order to alleviate the queueing problem at the uplink, but this solution has its constraints. The restrictions are set by the bandwidth limitations on satellite systems.

The following three factors play a large role in the on-board performance of the system: (i) the framing structure imposed by the uplink access, (ii) the size of the switch, and (iii) the packet destination distribution. The framing structure and the increase in the size of the switch have beneficial effects on the on-board performance as they reduce the burstiness of the incoming traffic. Meanwhile, a correlated packet destination distribution (burst-level or dynamic hot spot) requires a larger on-board queue size.

The observations made for the on-ground and on-board components lead us to believe that a traffic control scheme that shapes the traffic is required to reduce the burstiness. This is especially true when jitter-tolerant traffic has long-range dependent behaviour and has a strong correlation in the choice of packet destination. In addition, our results can be used in traffic admittance. For example, if the satellite traffic is dominated by long-range dependent data traffic at 95% load, the controller should not accept any more users requesting data services.

Finally, we see that since the packet loss probability of the system is ultimately determined by the loss probability due to queue overflow at the uplink and downlink, and since the queueing requirement does not improve much after L has increased beyond the value of 8, the Knockout switch is an excellent fabric for satellite applications. In addition, the Knockout switch offers modular growth, easy maintainability, fault tolerance, and lower complexity [39], that reinforce its candidacy.

Our study can be further extended in the following areas:

1. *Incorporation of a traffic control scheme:* a traffic control scheme can break the sequence of packet arrivals during long burst periods of a long-range dependent traffic and shape it into a short-range dependent one. The scope of our study did not permit us to investigate further the benefits of congestion control in a

satellite environment, but it is a possible subject to consider in future works. Connection admission control, which is a form of congestion control, can be of three types, namely, reactive, preventive, and proactive. Reactive connection admission control will not be useful in a long propagation delay environment such as a satellite system, since information on the status of the system will be obsolete by the time it reaches the network controller. Preventive connection admission control can be used in satellite systems, but can result in an under-utilization of the network. Of the three aforementioned schemes, proactive connection admission control is the most promising for satellite applications since it attempts to predict the status of the network based on past history of the system and reacts before the congestion actually occurs.

2. *Mathematical modeling and analysis:* in our study, we cannot investigate switches with size larger than $N = 16$, and loss probabilities of low order of magnitude (in some cases, lower than 10^{-4}) due to the extremely long simulation run-time required and the CPU limitations of the computers (SPARC 5, 10, and 20). This shows that although system simulation is a powerful tool in a system performance study, there are limitations to this method. The solution to overcome this obstacle is to obtain a mathematical model of the system, and from analysis, we can provide equations for the performance measures for all values of N and all orders of magnitude. However, as we have discussed, analysis of this system is very complex (this is the reason for our choice for simulation approach), but possible, if certain generalization rules or assumptions are made. At which

point, even if an approximate mathematical model is established, its validation will be based on a comparison of the analytical results and the simulative results.

3. *Implementation of broadcasting and multicasting functions:* although the satellite provides inherent broadcasting capability to the user population within the coverage area of the same downlink beam, multicasting to different downlink spot beams will affect the performance of the system as it adds more packets inside the network, specifically in the space segment. However, as the size of the switch increases, the number of permutations increases exponentially and care must be taken to make the study a feasible one.

Bibliography

[1] D. Delisle and L. Pelamourgues, "B-ISDN and How it Works", *IEEE Spectrum*, Aug. 1991, pp. 39-42.

[2] M. Schwartz, *Broadband Integrated Networks*, Prentice Hall, 1996.

[3] F. Ananasso, F.D. Priscoli, "The Role of Satellites in Personal Communication Services", *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 2, Feb. 1995, pp. 180-196.

[4] F. Leite, et al., "Regulatory Considerations Relating to IMT-2000", *IEEE Personal Communications*, Aug. 1997, pp. 14-19.

[5] T.T. Ha, *Digital Satellite Communications*, McGraw-Hill, 1990.

[6] J. Martin, *Communications Satellite Systems*, Prentice-Hall, 1978.

[7] T.N. Saadawi and M.H. Ammar with A.E. Hakeem, *Fundamentals of Telecommunication Networks*, John Wiley & Sons, 1994.

[8] T. Le-Ngoc, T.H. Bui, and M. Hachicha, "Performance of A Knockout Swicth for Multimedia Satellite Communications", in *Proc. Globecom '97*, Phoenix, Arizona, Nov. 3-8, 1997.

[9] G.B. Alaria, et al, "On-Board Processor for a TST/SSTDMA Telecommunications System", *ESA Journal*, Vol. 9, 1985.

[10] G. Beck, W. M. Holmes, "The ACTS Flight System: Cost Effective Advanced Communications Technology", in *Proc. Communication Satellite Systems Conference AIAA*, Mar. 1984.

[11] S.J. Campanella, B.A. Pontano, and H. Chalmers, "Future Switching Satellite", in *Proc. AIAA 12th International Communication Satellite Systems Conference*, Virginia, Mar. 13-17, 1988, pp. 264-273.

[12] P. Garland, S. Irani, and T. Inukai, "Fast Packet Based On-Board Switching for Advanced Business Services", in *Proc. Second European Conference on Satellite Communications*, Palais des Congres, Liege, Belgium, Oct. 22-24, 1991, pp. 127-136.

[13] W.D. Ivancic, M.J. Shalkhauser, and J.A. Quintana, "A Network Architecture for a Geostationary Communication Satellite", *IEEE Communications Magazine*, Jul. 1994. pp. 72-84.

[14] T. Inukai, F. Faris, and D.J. Shyy, "On-Board Processing Satellite Network Architectures for Broadband ISDN", in *Proc. AIAA 14th International Communication Satellite Systems Conference*, Washington D.C., Mar. 1992, pp.1471-1484.

[15] P. Garland, T. Le-Ngoc, and P. Takats, "Fast Packet Switches for Next Generation SATCOM Applications", in *Proc. ICDSC9*, 9th International Conference on Digital Satellite Communications, Copenhagen, May 18-22, 1992, pp. 63-70.

[16] J. Gilderson, J. Cherkaoui, "Onboard Switching for ATM via Satellite", *IEEE Communications Magazine*, Vol. 35, No. 7, Jul. 1997, pp. 66-70.

[17] Teledesic LLC, http://www.teledesic.com/

[18] Motorola, http://www.mot.com/

[19] ATM Over Satellite, http://www.ee.surrey.ac.uk/Personal/T.Ors/atmsat/

[20] J.Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, 1990.

[21] H. Ahmadi, W.E. Denzel, "A Survey of Modern High-Performance Switching Techniques", *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 7, Sep. 1989, pp. 4-16.

[22] F.A. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks", in *Proceedings of the IEEE*, Vol. 78, no. 1, Jan. 1990, pp. 133-167.

[23] J. Garcia-Haro, A. Jajszczyk, "ATM Shared-Memory Swicthing Architectures", *IEEE Network*, Jul./Aug. 1994, pp. 18-26.

[24] Y. Oie, et al., "Survey of Switching Techniques in High-Speed Networks and their Performance", *International Journal of Satellite Communications*, Vol. 9, 1991, pp. 285-303.

[25] R. Rooholamini, V. Cherkassky, and M. Garver, "Finding the Right ATM Switch for the Market", *Computer*, Apr. 1994, pp. 16-28.

[26] H. Heffes and D. M. Lucantoni, "A Markov Modulated Characterization of Packetized Voice and Data Traffic and Related Statistical Multiplexer Performance", *IEEE JSAC*, Vol. SAC-4, No. 6, Sept. 1986, pp. 856-868.

[27] T. Le-Ngoc, S.N. Subramanian, "A Pareto Modulated Poisson Process (PMPP) Model for Long-Range Dependent Traffic", in *Proc. Interop'97*, Las Vegas, May 7-8, 1997.

[28] H.J. Fowler, W.E. Leland, "Local Area Network Traffic Characteristics with Implications for Broadband Network Congestion Management", *IEEE JSAC*, Sept. 1991, pp.1139-1149.

[29] T. Le-Ngoc, "Dynamic Resource Allocation Schemes for Multimedia Satellite Communications", in *PIMRC'93*, Japan, Sept. 8-11, 1993, pp. 552-556.

[30] I.F. Akyildiz, S.H. Jeong, "Satellite ATM Networks: A Survey", *IEEE Communications Magazine*, Vol. 35, No.7, Jul. 1997, pp. 30-43.

[31] W.E. Denzel, A.P.J. Eenggbersen, and I. Iliadis, "A Flexible Shared-buffer Switch for ATM at Gb/s Rates", *Computer Networks and ISDN Systems*, Vol. 27, 1995, pp. 611-624.

[32] V.E. Benes, "Optimal Rearrangeable Multistage Connecting Networks", *Systems Technical Journal*, Vol. 43, No. 7, Jul. 1964, pp. 1641-1656.

[33] K.E. Batcher, "Sorting Networks and Their Applications", in *AFIPS Proc. 1968 Spring Joint Computer Conf.*, Vol. 32, pp. 307-314.

[34] A. Huang, S. Knauer, "Starlite: A Wideband Digital Switch", in *Proc. Globecom 84*, Atlanta, Georgia, Dec. 1984, pp. 121-125.

[35] J. Giacopelli, M. Littlewood, and W.D. Sincoslie, "Sunshine: A High Performance Self-routing Broadband Packet Swicth Architecture", submitted to ISS 90.

[36] C.P. Kruskal, M. Snir, "The Performance of Multistage Interconnection Networks for Multiprocessors", *IEEE Trans. Computers*, Vol. C-32, No. 12, Dec. 1983, pp. 1091-1098.

[37] M. Kumar, J.R. Jump, "Performance of Unbuffered Shuffle-exchange Networks", *IEEE Trans. Computers*, Vol. c-35, No. 6, Jun. 1986, pp. 573-577.

[38] T. Kwok, F. Tobagi, "Tandem-banyan Switching Fabric", *Computer Systems Laboratory*, Technical Report, Mar. 1990.

[39] Y. Yeh, M. Hluchyj, and A. Acampora, "The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching", *IEEE JSAC*, Vol. SAC-5, No. 8, Oct. 1987, pp. 1274-1283.

[40] S.N. Subramanian, "Traffic Modeling in a Multimedia Environment", M.A.Sc. Thesis, Department of ECE, Concordia University, Montréal, 1996.

[41] S.N. Subramanian, T. Le-Ngoc, "Traffic Modeling in a Multi-media Environment", in *Proc. CCECE/CCGEI 95*, Montréal, 1995, pp. 838-841.

[42] T. Le-Ngoc, S.V. Krishnamurthy, "Performance of Combined Free/Demand Assignment Multiple-Access Schemes in Satellite Communications", *Int. Jour. of Sat. Comm.*, Vol. 14, 1996, pp.11-21.

[43] J. Huang, T. Le-Ngoc, and J.F. Hayes, "Broadbaand SATCOM System for Multimedia Services", in *Proc. ICC'96*, Dallas, 1996, pp. 906-909.

[44] T.V.J. Ganesh Babu, T. Le-Ngoc, and J.F.Hayes, "Performance Evaluation of Priority Based Service in Multimedia ATM Networks", Second IFIP Workshop on Traffic Management and Synthesis of ATM Networks, Montréal Sept. 24-26, 1997.

[45] B. Maglaris, et al., "Performance Models of Statistical Multiplexing in Packet Video Communications", *IEEE Trans. on Comm.*, Vol. 36, No. 7, July 1988, pp.834-843.

[46] A. Arcidiacono, "Multimedia Services and Data Broadcasting via Satellite", *Electronics and Communication Engineering Journal*, February 1997, pp.33-37.

APPENDIX A
PROGRAM LISTING

Our satellite-switched system was developed on a simulation package called OPtimized Network Engineering Tools (OPNET). The simulation is built with independent building blocks called process models. The operations of these process models are specified by finite state machines, translated into C code.

Figure A.1 shows the network model of the system employing a 4×4 switch. The system can easily be modified to accommodate a 16×16 switch (Figure A.2). However, for simplicity, in the following we will give the process model and the program code for each of the parts comprising the 4×4 satellite-switched system, namely, the uplink (up00, up01, up02, and up03), the scheduler, and the switch. The traffic generator model (gen00, gen01, gen02, and gen03) and program code are given in [40].

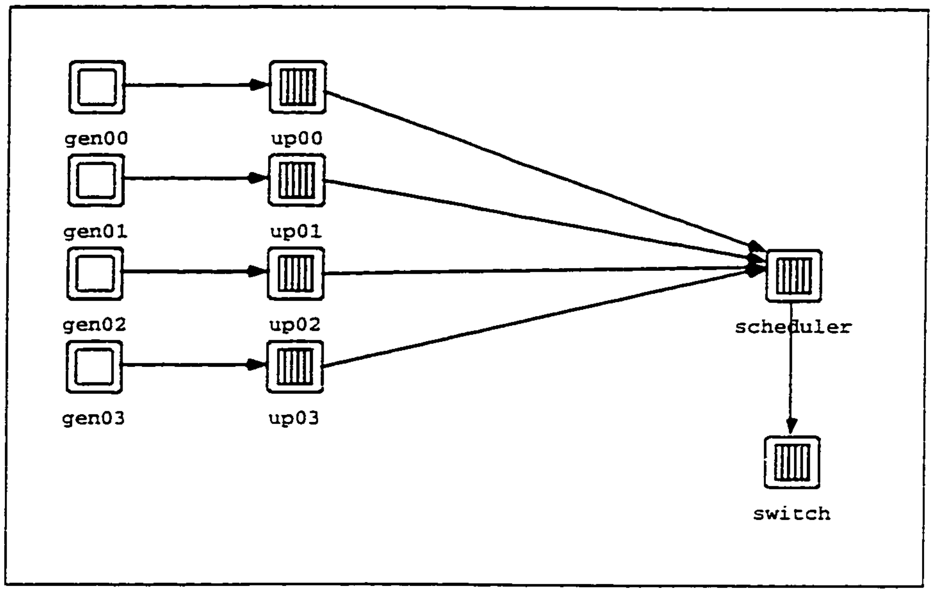

Figure A.1: Network model of the system employing a 4 × 4 switch

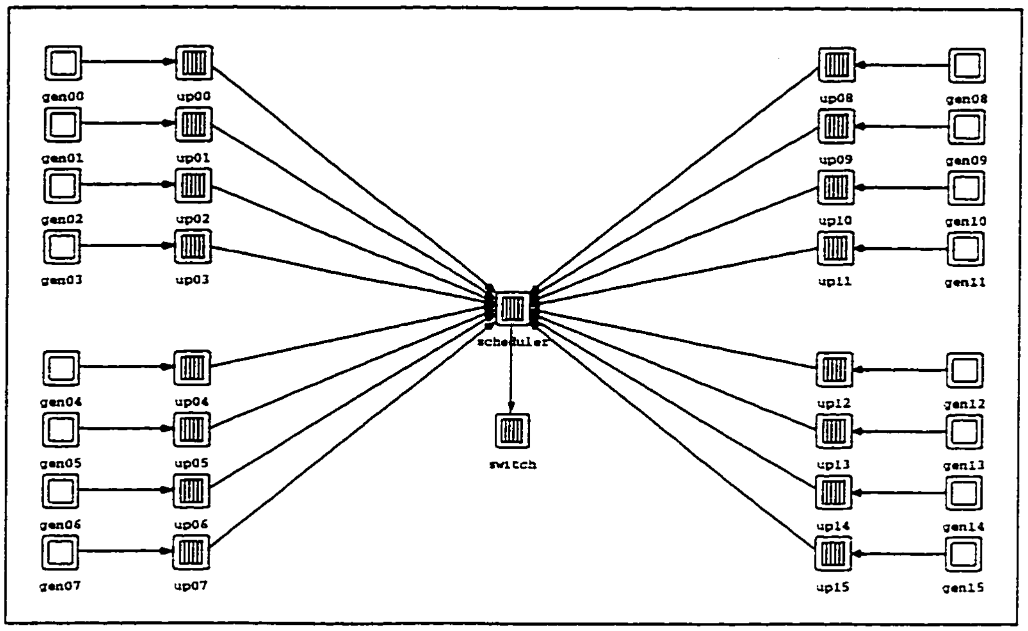

Figure A.2: Network model of the system employing a 16 × 16 switch

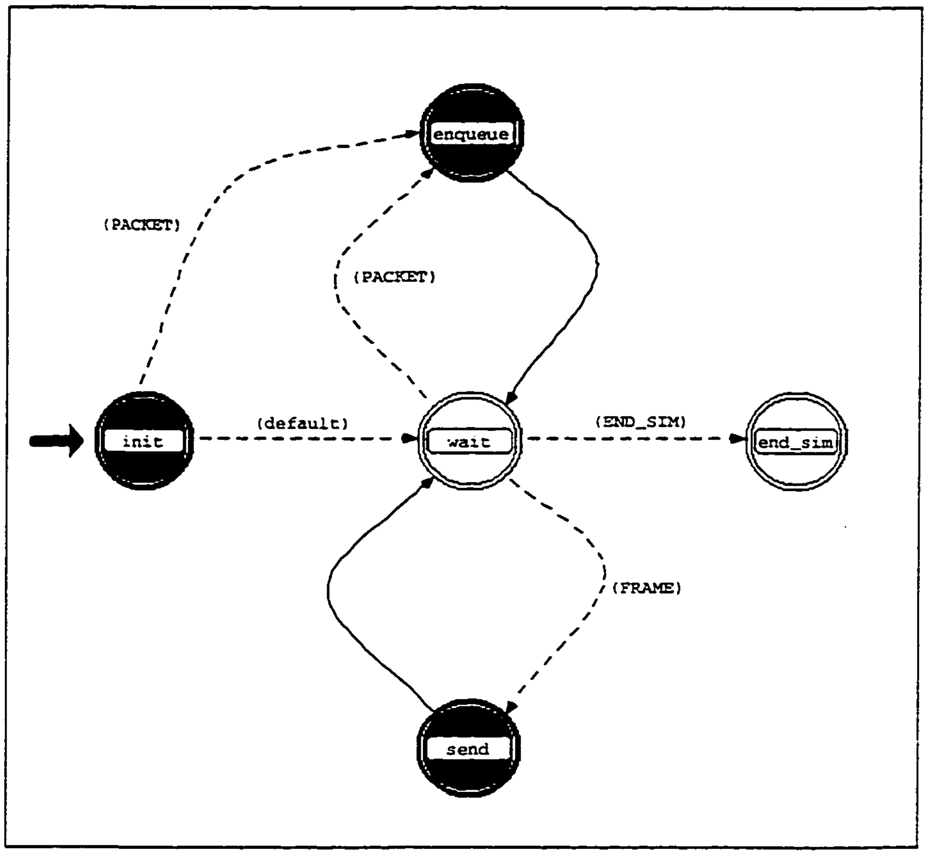

Figure A.3: Process model of the uplink

...
...

Header Block

```
#include<math.h>
#include<stdio.h>
#include<sys/time.h>

#define N_inputs            4
#define SLOT_TIME_CODE      2
#define FRAME_TIME_CODE    22
#define frame               0.024
#define C                   512

#define ARRIVAL             (op_intrpt_type() == OPC_INTRPT_STRM)
#define SLOT_INTRPT         (op_intrpt_type() == OPC_INTRPT_SELF && \
                            op_intrpt_code() == SLOT_TIME_CODE)
#define FRAME_INTRPT        (op_intrpt_type() == OPC_INTRPT_SELF && \
                            op_intrpt_code() == FRAME_TIME_CODE)
```

State Variable Block

```
double \slot_time;
int \a[16];
int \status[16][512];
int \pt;
```

Temporary Variable Block

```
packet *pkptr;
int i;
int j;
int pack;
int count;
int stream;
int flag;
```

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**

```
slot_time=frame/C;

for (i=0; i<N_inputs; i++)
      {a[i]=0;
      for (j=0; j<C; j++)
            {status[i][j]=0;
            }
      }

pt=0;
```

...
...

```
op_intrpt_schedule_self(op_sim_time()+frame, FRAME_TIME_CODE);
op_intrpt_schedule_self(op_sim_time()+slot_time, SLOT_TIME_CODE);
```

transition **init -> queue**

attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition	ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **init -> wait**

attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **wait**

attribute	*value*	*type*	*default value*
name	wait	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **wait -> queue**

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

transition **wait -> frame**

attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition	FRAME_INTRPT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **wait -> slot**

attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition	SLOT_INTRPT	string	
executive		string	

...
...

color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **queue**

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

enter execs **queue**

```
stream=op_intrpt_strm();

pkptr=op_pk_get(stream);

op_subq_pk_insert(stream, pkptr, OPC_QPOS_TAIL);
a[stream]++;
```

transition **queue -> wait**

attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **frame**

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

enter execs **frame**

```
pt=0;
for(i=0; i<N_inputs; i++)
      {for (j=0; j<C; j++)
            {status[i][j]=0;
            }
      }

for (i=0; i<N_inputs; i++)
      {count=C;
      for(j=1; j<=a[i]; j++)
            {pack=(int)op_dist_uniform(count);
            flag=0;
            while (pack>=0)
                  {if (status[i][flag] == 0)
                        {pack--;
```

```
                    flag++;
                    }
                else
                    {flag++;
                    }
                }
            flag--;
            status[i][flag]=1;
            count--;
            }
        }

    for (i=0; i<N_inputs; i++)
        {a[i]=0;
        }

    op_intrpt_schedule_self(op_sim_time()+frame, FRAME_TIME_CODE);
```

transition **frame -> wait**

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

forced state **slot**

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

enter execs **slot**

```
for (i=0; i<N_inputs; i++)
    {if (status[i][pt] == 1)
        {pkptr = op_subq_pk_remove(i, OPC_QPOS_HEAD);
        op_pk_send(pkptr, 0);
        }
    }

pt++;

op_intrpt_schedule_self(op_sim_time()+slot_time, SLOT_TIME_CODE);
```

transition **slot -> wait**

attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition		string	
executive		string	

...
...

color	RGB333	color	RGB333
drawing style	spline	toggle	spline

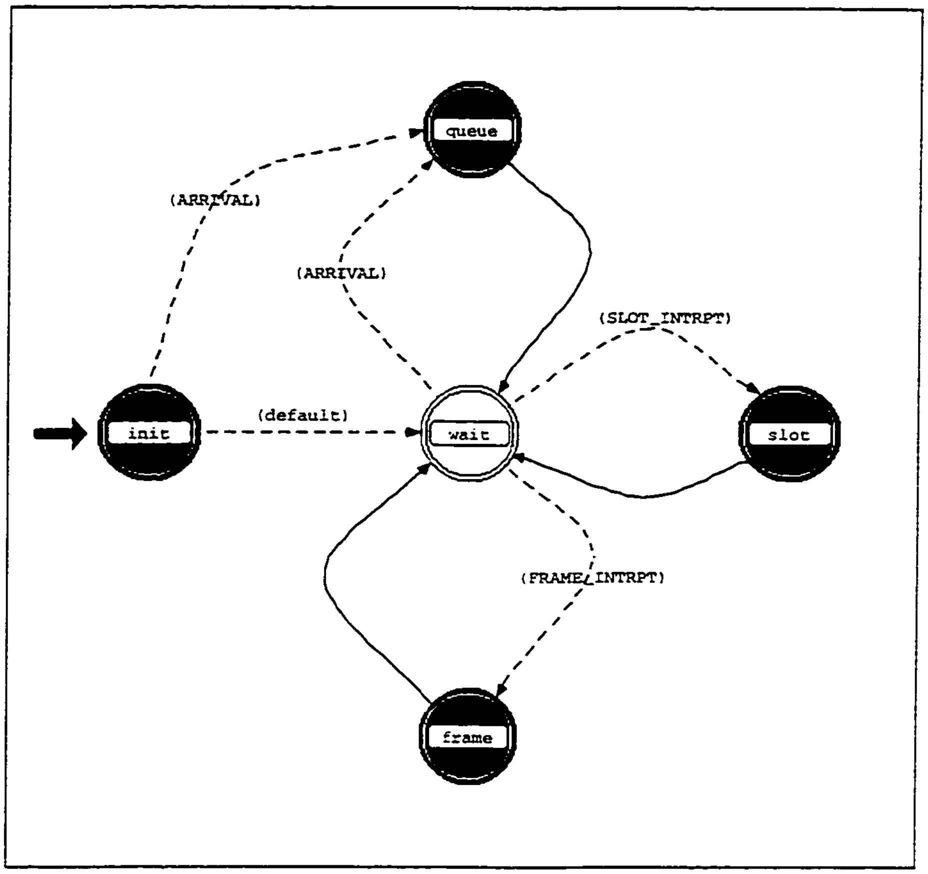

Figure A.4: Process model of the scheduler

Process Model Attributes

attribute	*value*	*type*	*default value*
out_filename	promoted	string	out.dat

Header Block

```
#include<math.h>
#include<stdio.h>
#include<sys/time.h>

#define N_inputs      4
#define L             4
#define M             3
#define N             2
#define O             1
#define frame         0.024
#define cell          512
#define QUEUE         0
#define SLOT_CODE     7
#define FRAME_CODE      9
#define PACKET   op_intrpt_type() == OPC_INTRPT_STRM
#define SLOT     op_intrpt_type() == OPC_INTRPT_SELF &&\
                 op_intrpt_code() == SLOT_CODE
#define FRAME    op_intrpt_type() == OPC_INTRPT_SELF &&\
                 op_intrpt_code() == FRAME_CODE
#define END_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM

int flag;
```

State Variable Block

```
FILE* \fp;
char \outfile[40];
int \id;
int \dest_addr;
int \type;

int \qloss1, \qloss2, \qloss3;
int \qlossm0, \qlossm1, \qlossm2, \qlossm3;
int \qlossn0, \qlossn1, \qlossn2, \qlossn3;
int \qlosso0, \qlosso1, \qlosso2, \qlosso3;

double \totqloss;
int \totqlossm, \totqlossn, \totqlosso;
double \totrtlossm, \totrtlossn, \totrtlosso;
double \totdalossm, \totdalossn, \totdalosso;

int \as0, \as1, \as2, \as3;
int \m0, \n0, \o0, \m1, \n1, \o1, \m2, \n2, \o2, \m3, \n3, \o3;
int \rtm0, \rtn0, \rto0, \rtm1, \rtn1, \rto1, \rtm2, \rtn2, \rto2, \rtm3, \rtn3, \rto3;
int \rtdm0, \rtdn0, \rtdo0, \rtdm1, \rtdn1, \rtdo1, \rtdm2, \rtdn2, \rtdo2, \rtdm3, \rtdn3, \rtdo3;
int \nm_lossm0, \nm_lossn0, \nm_losso0, \nm_lossm1, \nm_lossn1, \nm_losso1, \nm_lossm2, \nm_lossn2, \nm_losso2, \nm_lossm3
int \nm_lossdm0, \nm_lossdn0, \nm_lossdo0, \nm_lossdm1, \nm_lossdn1, \nm_lossdo1, \nm_lossdm2, \nm_lossdn2, \nm_lossdo2, \r
int \dam0, \dan0, \dao0, \dam1, \dan1, \dao1, \dam2, \dan2, \dao2, \dam3, \dan3, \dao3;
int \dadm0, \dadn0, \dado0, \dadm1, \dadn1, \dado1, \dadm2, \dadn2, \dado2, \dadm3, \dadn3, \dado3;

double \nm_sent_datam, \nm_sent_datan, \nm_sent_datao;
double \delaym, \delayn, \delayo;
double \avg_delaym, \avg_delayn, \avg_delayo;
double \square_delaym, \square_delayn, \square_delayo;
double \variance_delaym, \variance_delayn, \variance_delayo;

int \a0, \a1, \a2, \a3;
int \at0, \at1, \at2, \at3;
int \am0, \am1, \am2, \am3;
int \an0, \an1, \an2, \an3;
int \ao0, \ao1, \ao2, \ao3;
int \rt1, \rt2, \rt3;
int \da1, \da2, \da3;
int \rtd1, \rtd2, \rtd3;
int \dad1, \dad2, \dad3;
double \queued_real;
double \queued_data;

/* packets destinated to each output BEFORE being switched */
int \data_to_out0;
int \voice_to_out0;
int \video_to_out0;

/* packets destinated to output 0 AFTER being switched */
int \out0_data;
int \out0_realtime;

/* packets at output0 after being served at output buffer */
int \outbuffer0_data;
int \outbuffer0_realtime;

double \nm_loss;
double \avg_loss;
double \sw_total;
double \total_arrived;
double \nm_sw_real;
double \nm_sw_data;
double \slot_time;
double \total_delay_data;
double \avg_delay_data;
double \q_loss;
double \avg_q_loss;
double \nm_received_total;
double \nm_sent_real;
double \nm_sent_data;
double \tmp_delay;
double \delay;
double \square_delay;
double \avg_delay;
double \variance_delay;
```

Temporary Variable Block

```
Packet *pkptr;
Packet *new_pkptr;
int i;
int count;
int pksize;
double cr_time;
```

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

```
/* initialization of parameters */

qloss1=0; qloss2=0; qloss3=0;
qlossm0=0; qlossm1=0; qlossm2=0; qlossm3=0;
qlossn0=0; qlossn1=0; qlossn2=0; qlossn3=0;
qlosso0=0; qlosso1=0; qlosso2=0; qlosso3=0;
as0=0; as1=0; as2=0; as3=0;
am0 = 0; am1 = 0; am2 = 0; am3 = 0;
an0 = 0; an1 = 0; an2 = 0; an3 = 0;
ao0 = 0; ao1 = 0; ao2 = 0; ao3 = 0;
m0=0; n0=0; o0=0; m1=0; n1=0; o1=0;
m2=0; n2=0; o2=0; m3=0; n3=0; o3=0;
rtm0=0; rtn0=0; rto0=0; rtm1=0; rtn1=0; rto1=0;
rtm2=0; rtn2=0; rto2=0; rtm3=0; rtn3=0; rto3=0;
rtdm0=0; rtdn0=0; rtdo0=0; rtdm1=0; rtdn1=0; rtdo1=0;
rtdm2=0; rtdn2=0; rtdo2=0; rtdm3=0; rtdn3=0; rtdo3=0;
nm_lossm0=0; nm_lossn0=0; nm_losso0=0; nm_lossm1=0; nm_lossn1=0; nm_losso1=0;
nm_lossm2=0; nm_lossn2=0; nm_losso2=0; nm_lossm3=0; nm_lossn3=0; nm_losso3=0;
nm_lossdm0=0; nm_lossdn0=0; nm_lossdo0=0; nm_lossdm1=0; nm_lossdn1=0; nm_lossdo1=0;
nm_lossdm2=0; nm_lossdn2=0; nm_lossdo2=0; nm_lossdm3=0; nm_lossdn3=0; nm_lossdo3=0;
dam0=0; dan0=0; dao0=0; dam1=0; dan1=0; dao1=0;
dam2=0; dan2=0; dao2=0; dam3=0; dan3=0; dao3=0;
dadm0=0; dadn0=0; dado0=0; dadm1=0; dadn1=0; dado1=0;
dadm2=0; dadn2=0; dado2=0; dadm3=0; dadn3=0; dado3=0;
nm_sent_datam=0.0;
nm_sent_datan=0.0;
nm_sent_datao=0.0;
delaym=0.0;
delayn=0.0;
delayo=0.0;
avg_delaym=0.0;
avg_delayn=0.0;
avg_delayo=0.0;
square_delaym=0.0;
square_delayn=0.0;
square_delayo=0.0;
variance_delaym=0.0;
variance_delayn=0.0;
variance_delayo=0.0;

a0 = 0; a1 = 0; a2 = 0; a3 = 0;
at0 = 0; at1 = 0; at2 = 0; at3 = 0;
rt1 = 0; rt2 = 0; rt3 = 0;
da1 = 0; da2 = 0; da3 = 0;
rtd1 = 0; rtd2 = 0; rtd3 = 0;
dad1 = 0; dad2 = 0; dad3 = 0;

slot_time = frame/cell;
```

```
queued_real = 0.0;
queued_data = 0.0;
nm_loss = 0.0;
avg_loss = 0.0;
sw_total = 0.0;
total_arrived = 0.0;
nm_sw_real = 0.0;
nm_sw_data = 0.0;
total_delay_data = 0.0;
avg_delay_data = 0.0;
q_loss = 0.0;
avg_q_loss = 0.0;
nm_sent_real = 0.0;
nm_sent_data = 0.0;
tmp_delay = 0.0;
delay = 0.0;
square_delay = 0.0;
avg_delay = 0.0;
variance_delay = 0.0;
data_to_out0 = 0;
voice_to_out0=0;
video_to_out0=0;
out0_data=0;
out0_realtime=0;
outbuffer0_data=0;
outbuffer0_realtime=0;

id=op_id_self();
op_ima_obj_attr_get(id,"out_filename",outfile);
fp=fopen(outfile,"w");

/* schedule the first slot going out of the switch */
/* note: the incremental time shift to assure that only
previous packets are being served, not the ones that just
arrived. */

op_intrpt_schedule_self(op_sim_time()+frame, FRAME_CODE);
op_intrpt_schedule_self(op_sim_time()+slot_time, SLOT_CODE);
```

transition **init -> wait**

attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **init -> enqueue**

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

...
...

unforced state **wait**

attribute	*value*	*type*	*default value*
name	wait	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **wait -> enqueue**

attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition	PACKET	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **wait -> end_sim**

attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition	END_SIM	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **wait -> slot_time**

attribute	*value*	*type*	*default value*
name	tr_8	string	tr
condition	SLOT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **wait -> frame_time**

attribute	*value*	*type*	*default value*
name	tr_10	string	tr
condition	FRAME	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **enqueue**

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

...
...

enter execs **enqueue**

```
/* get info about incoming packet */

pkptr = op_pk_get(op_intrpt_strm());
op_pk_nfd_get(pkptr, "dest_addr", &dest_addr);
op_pk_nfd_get(pkptr, "type", &type);
op_pk_nfd_get(pkptr, "cr_time", &cr_time);

total_arrived++;

/* keep packets going to output 0 only */

switch(dest_addr)
{case 0:
    a0++;
    at0++;
    sw_total++;
    as0++;
    if (as0>M) m0=1;
    if (as0>N) n0=1;
    if (as0>O) o0=1;
    switch(type)
    {case 1:
    video_to_out0++;
        queued_real++;
        nm_sw_real++;
        out0_realtime++;
        op_pk_destroy(pkptr);
        if (m0==0)
            {rtm0++;
            am0++;}
        else nm_lossm0++;
        if (n0==0)
            {rtn0++;
            an0++;}
        else nm_lossn0++;
        if (o0==0)
            {rto0++;
            ao0++;}
        else nm_losso0++;
        break;
    case 2:
    voice_to_out0++;
        queued_real++;
        nm_sw_real++;
        out0_realtime++;
        op_pk_destroy(pkptr);
        if (m0==0)
            {rtm0++;
            am0++;}
        else nm_lossm0++;
        if (n0==0)
            {rtn0++;
            an0++;}
        else nm_lossn0++;
        if (o0==0)
            {rto0++;
            ao0++;}
        else nm_losso0++;
```

```
                break;
        case 3:
                data_to_out0++;
                queued_data++;
                nm_sw_data++;
                out0_data++;
                total_delay_data += op_sim_time() - op_pk_creation_time_get(pkptr);
                if (m0==0)
                        {dam0++;
                        am0++;
                        new_pkptr=op_pk_copy(pkptr);
                        op_pk_nfd_set(new_pkptr, "cr_time", cr_time);
                        op_subq_pk_insert(1,new_pkptr,OPC_QPOS_TAIL);
                        }
                else nm_lossdm0++;
                if (n0==0)
                        {dan0++;
                        an0++;
                        new_pkptr=op_pk_copy(pkptr);
                        op_pk_nfd_set(new_pkptr, "cr_time", cr_time);
                        op_subq_pk_insert(2,new_pkptr,OPC_QPOS_TAIL);
                        }
                else nm_lossdn0++;
                if (o0==0)
                        {dao0++;
                        ao0++;
                        new_pkptr=op_pk_copy(pkptr);
                        op_pk_nfd_set(new_pkptr, "cr_time", cr_time);
                        op_subq_pk_insert(3,new_pkptr,OPC_QPOS_TAIL);
                        }
                else nm_lossdo0++;
                op_subq_pk_insert(QUEUE,pkptr,OPC_QPOS_TAIL);
                break;
        }
        break;
case 1:
        a1++;
        at1++;
        as1++;
        if (as1>M) m1=1;
        if (as1>N) n1=1;
        if (as1>O) o1=1;
        switch(type)
        {case 1:
        rt1++;
                if (m1==0)
                        {rtm1++;
                        am1++;}
                else nm_lossm1++;
                if (n1==0)
                        {rtn1++;
                        an1++;}
                else nm_lossn1++;
                if (o1==0)
                        {rto1++;
                        ao1++;}
                else nm_losso1++;
                break;
        case 2:
         rt1++;
              if (m1==0)
                   {rtm1++;
                   am1++;}
              else nm_lossm1++;
              if (n1==0)
                   {rtn1++;
                   an1++;}
              else nm_lossn1++;
              if (o1==0)
                   {rto1++;
                   ao1++;}
              else nm_losso1++;
              break;
         case 3:
         da1++;
              if (m1==0)
                   {dam1++;
                   am1++;}
              else nm_lossdm1++;
              if (n1==0)
                   {dan1++;
                   an1++;}
              else nm_lossdn1++;
              if (o1==0)
                   {dao1++;
                   ao1++;}
              else nm_lossdo1++;
              break;
         }
         op_pk_destroy(pkptr);
         break;
case 2:
         a2++;
         at2++;
         as2++;
         if (as2>M) m2=1;
         if (as2>N) n2=1;
         if (as2>O) o2=1;
         switch(type)
         {case 1:
         rt2++;
              if (m2==0)
                   {rtm2++;
                   am2++;}
              else nm_lossm2++;
              if (n2==0)
                   {rtn2++;
                   an2++;}
              else nm_lossn2++;
              if (o2==0)
                   {rto2++;
                   ao2++;}
              else nm_losso2++;
              break;
         case 2:
         rt2++;
              if (m2==0)
                   {rtm2++;
```

```
                    am2++;}
                else nm_lossm2++;
                if (n2==0)
                    {rm2++;
                    an2++;}
                else nm_lossn2++;
                if (o2==0)
                    {ro2++;
                    ao2++;}
                else nm_losso2++;
                break;
            case 3:
            da2++;
                if (m2==0)
                    {dam2++;
                    am2++;}
                else nm_lossdm2++;
                if (n2==0)
                    {dan2++;
                    an2++;}
                else nm_lossdn2++;
                if (o2==0)
                    {dao2++;
                    ao2++;}
                else nm_lossdo2++;
                break;
            }
            op_pk_destroy(pkptr);
            break;
        case 3:
            a3++;
            at3++;
            as3++;
            if (as3>M) m3=1;
            if (as3>N) n3=1;
            if (as3>O) o3=1;
            switch(type)
            {case 1:
            rt3++;
                if (m3==0)
                    {rm3++;
                    am3++;}
                else nm_lossm3++;
                if (n3==0)
                    {rn3++;
                    an3++;}
                else nm_lossn3++;
                if (o3==0)
                    {ro3++;
                    ao3++;}
                else nm_losso3++;
                break;
            case 2:
            rt3++;
                if (m3==0)
                    {rm3++;
                    am3++;}
                else nm_lossm3++;
                if (n3==0)
```

...
...

```
                {rm3++;
                an3++;}
            else nm_lossn3++;
            if (o3==0)
                {rto3++;
                ao3++;}
            else nm_losso3++;
            break;
        case 3:
        da3++;
            if (m3==0)
                {dam3++;
                am3++;}
            else nm_lossdm3++;
            if (n3==0)
                {dan3++;
                an3++;}
            else nm_lossdn3++;
            if (o3==0)
                {dao3++;
                ao3++;}
            else nm_lossdo3++;
            break;
        }
        op_pk_destroy(pkptr);
        break;
    }
```

transition **enqueue -> wait**

attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **slot_time**

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

enter execs **slot_time**

```
as0=0; as1=0; as2=0; as3=0;
m0=0; n0=0; o0=0; m1=0; n1=0; o1=0;
m2=0; n2=0; o2=0; m3=0; n3=0; o3=0;

/* schedule next time slot */

op_intrpt_schedule_self(op_sim_time() + slot_time, SLOT_CODE);
```

transition **slot time -> wait**			
attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **end sim**

```
fclose(fp):
avg_loss = nm_loss/sw_total:
totrtlossm=nm_lossm0+nm_lossm1+nm_lossm2+nm_lossm3:
totdalossm=nm_lossdm0+nm_lossdm1+nm_lossdm2+nm_lossdm3:
totrtlossn=nm_lossn0+nm_lossn1+nm_lossn2+nm_lossn3:
totdalossn=nm_lossdn0+nm_lossdn1+nm_lossdn2+nm_lossdn3:
totrtlosso=nm_losso0+nm_losso1+nm_losso2+nm_losso3:
totdalosso=nm_lossdo0+nm_lossdo1+nm_lossdo2+nm_lossdo3:

printf("****************************\n");
printf("        SWITCH  RESULTS       \n");
printf("****************************\n");
printf("total arrived 0 = %d\n", ar0);
printf("total arrived 1 = %d\n", ar1);
printf("total arrived 2 = %d\n", ar2);
printf("total arrived 3 = %d\n", ar3);
printf("video to out0 = %d\n", video_to_out0);
printf("voice to out0 = %d\n", voice_to_out0);
printf("data to out0 = %d\n", data_to_out0);
printf("average loss = %11.9f\n", avg_loss);
printf("number of real-time packet loss at 0 (L=%d) = %d\n", M,nm_lossm0);
printf("number of real-time packet loss at 1 (L=%d) = %d\n", M,nm_lossm1);
printf("number of real-time packet loss at 2 (L=%d) = %d\n", M,nm_lossm2);
printf("number of real-time packet loss at 3 (L=%d) = %d\n", M,nm_lossm3);
printf("total real-time packet loss (L=%d) = %f\n", M,totrtlossm);
printf("number of data packet loss at 0 (L=%d) = %d\n", M,nm_lossdm0);
printf("number of data packet loss at 1 (L=%d) = %d\n", M,nm_lossdm1);
printf("number of data packet loss at 2 (L=%d) = %d\n", M,nm_lossdm2);
printf("number of data packet loss at 3 (L=%d) = %d\n", M,nm_lossdm3);
printf("total data packet loss (L=%d) = %f\n", M,totdalossm);
printf("number of real-time packet loss at 0 (L=%d) = %d\n", N,nm_lossn0);
printf("number of real-time packet loss at 1 (L=%d) = %d\n", N,nm_lossn1);
printf("number of real-time packet loss at 2 (L=%d) = %d\n", N,nm_lossn2);
printf("number of real-time packet loss at 3 (L=%d) = %d\n", N,nm_lossn3);
printf("total real-time packet loss (L=%d) = %f\n", N,totrtlossn);
printf("number of data packet loss at 0 (L=%d) = %d\n", N,nm_lossdn0);
printf("number of data packet loss at 1 (L=%d) = %d\n", N,nm_lossdn1);
printf("number of data packet loss at 2 (L=%d) = %d\n", N,nm_lossdn2);
printf("number of data packet loss at 3 (L=%d) = %d\n", N.nm_lossdn3);
printf("total data packet loss (L=%d) = %f\n", N.totdalossn);
printf("number of real-time packet loss at 0 (L=%d) = %d\n", O.nm_losso0);
printf("number of real-time packet loss at 1 (L=%d) = %d\n", O.nm_losso1);
printf("number of real-time packet loss at 2 (L=%d) = %d\n", O.nm_losso2);
printf("number of real-time packet loss at 3 (L=%d) = %d\n", O.nm_losso3);
printf("total real-time packet loss (L=%d) = %f\n", O.tortlosso);
printf("number of data packet loss at 0 (L=%d) = %d\n", O.nm_lossdo0);
printf("number of data packet loss at 1 (L=%d) = %d\n", O.nm_lossdo1);
printf("number of data packet loss at 2 (L=%d) = %d\n", O.nm_lossdo2);
printf("number of data packet loss at 3 (L=%d) = %d\n", O.nm_lossdo3);
printf("total data packet loss (L=%d) = %f\n", O.totdalosso);

/* printf("delay from begin sync to begin switch\n");
printf("average delay (data) = %11.9f\n", avg_delay_data); */

printf("total arrived to switch = %f\n", total_arrived);
printf("total arrived and destination 0 = %f\n", sw_total);
printf("real packets switched = %f\n", nm_sw_real);
printf("data packets switched = %f\n", nm_sw_data);
printf("total loss = %f\n", nm_loss);

nm_received_total = nm_sent_real + nm_sent_data + queued_real + queued_data + q_loss;
avg_q_loss = q_loss/nm_received_total;
avg_delay = delay/nm_sent_data;
avg_delaym = delaym/nm_sent_datam;
avg_delayn = delayn/nm_sent_datan;
avg_delayo = delayo/nm_sent_datao;
variance_delay = (square_delay/nm_sent_data) - (avg_delay * avg_delay);
variance_delaym = (square_delaym/nm_sent_datam) - (avg_delaym * avg_delaym);
variance_delayn = (square_delayn/nm_sent_datan) - (avg_delayn * avg_delayn);
variance_delayo = (square_delayo/nm_sent_datao) - (avg_delayo * avg_delayo);

printf("****************************\n");
printf("    OUTPUT QUEUE RESULTS    \n");
printf("****************************\n");
printf("average loss = %11.9f\n", avg_q_loss);
printf("delay is from switch to output queue\n");
printf("average delay (data) = %11.9f\n", avg_delay);
printf("average delay (data)(L=%d) = %11.9f\n", M.avg_delaym);
printf("average delay (data)(L=%d) = %11.9f\n", N.avg_delayn);
printf("average delay (data)(L=%d) = %11.9f\n", O.avg_delayo);
printf("delay variance (data) = %11.9f\n", variance_delay);
printf("delay variance (data)(L=%d) = %11.9f\n", M.variance_delaym);
printf("delay variance (data)(L=%d) = %11.9f\n", N.variance_delayn);
printf("delay variance (data)(L=%d) = %11.9f\n", O.variance_delayo);
printf("real in queue = %f\n", queued_real);
printf("data in queue = %f\n", queued_data);
printf("real packets sent = %f\n", nm_sent_real);
printf("data packets sent = %f\n", nm_sent_data);
totqloss=q_loss+qloss1+qloss2+qloss3;
totqlossm=qlossm0+qlossm1+qlossm2+qlossm3;
totqlossn=qlossn0+qlossn1+qlossn2+qlossn3;
totqlosso=qlosso0+qlosso1+qlosso2+qlosso3;
printf("queue loss at 0 = %f\n", q_loss);
printf("queue loss at 1 = %d\n", qloss1);
printf("queue loss at 2 = %d\n", qloss2);
printf("queue loss at 3 = %d\n", qloss3);
printf("total queue loss = %f\n", totqloss);
printf("queue loss at 0 (L=%d) = %d\n", M.qlossm0);
printf("queue loss at 1 (L=%d) = %d\n", M.qlossm1);
printf("queue loss at 2 (L=%d) = %d\n", M.qlossm2);
printf("queue loss at 3 (L=%d) = %d\n", M.qlossm3);
printf("total queue loss (L=%d) = %d\n", M.totqlossm);
printf("queue loss at 0 (L=%d) = %d\n", N.qlossn0);
printf("queue loss at 1 (L=%d) = %d\n", N.qlossn1);
printf("queue loss at 2 (L=%d) = %d\n", N.qlossn2);
printf("queue loss at 3 (L=%d) = %d\n", N.qlossn3);
printf("total queue loss (L=%d) = %d\n", N.totqlossn);
printf("queue loss at 0 (L=%d) = %d\n", O.qlosso0);
printf("queue loss at 1 (L=%d) = %d\n", O.qlosso1);
printf("queue loss at 2 (L=%d) = %d\n", O.qlosso2);
printf("queue loss at 3 (L=%d) = %d\n", O.qlosso3);
printf("total queue loss (L=%d) = %d\n", O.totqlosso);
```

forced state **frame_time**

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

enter execs **frame_time**

```
/* reinitialize count. the counter that keeps track of
the number of packets served */

count = 0;

rtdm0=0; rtdn0=0; rtdo0=0; rtdm1=0; rtdn1=0; rtdo1=0;
rtdm2=0; rtdn2=0; rtdo2=0; rtdm3=0; rtdn3=0; rtdo3=0;
rtd1 = 0; rtd2 = 0; rtd3 = 0;
fprintf(fp,"1 %d %d %d 0\n",data_to_out0,voice_to_out0,video_to_out0);
fprintf(fp,"2 %d %d %d %d\n",a0,a1,a2,a3);
fprintf(fp,"3 %d %d %d %d\n",am0,am1,am2,am3);
fprintf(fp,"4 %d %d %d %d\n",an0,an1,an2,an3);
fprintf(fp,"5 %d %d %d %d\n",ao0,ao1,ao2,ao3);
fprintf(fp,"6 %d %d %d 0\n",rt1,rt2,rt3);
fprintf(fp,"7 %d %d %d %d\n",rtm0,rtm1,rtm2,rtm3);
fprintf(fp,"8 %d %d %d %d\n",rtn0,rtn1,rtn2,rtn3);
fprintf(fp,"9 %d %d %d %d\n",rto0,rto1,rto2,rto3);

data_to_out0 = 0;
voice_to_out0=0;
video_to_out0=0;
outbuffer0_data=0;
outbuffer0_realtime=0;

a0 = 0; a1 = 0; a2 = 0; a3 = 0;
am0 = 0; am1 = 0; am2 = 0; am3 = 0;
an0 = 0; an1 = 0; an2 = 0; an3 = 0;
ao0 = 0; ao1 = 0; ao2 = 0; ao3 = 0;
dad1 += da1; dad2 += da2; dad3 += da3;
dadm0 += dam0; dadm1 += dam1; dadm2 += dam2; dadm3 += dam3;
```

```
dadn0 += dan0; dadn1 += dan1; dadn2 += dan2; dadn3 += dan3;
dado0 += dao0; dado1 += dao1; dado2 += dao2; dado3 += dao3;

if (queued_real != 0)
      {if (queued_real > cell)
            {q_loss += queued_real - cell;
            nm_sent_real += cell;
      outbuffer0_realtime += cell;
            count = cell;
            }
      else
            {nm_sent_real += queued_real;
            count = queued_real;
      outbuffer0_realtime += queued_real;
            }
      queued_real = 0.0;
      }

/* check if we still have capacity */
/* if yes, serve up to capacity */

if (count < cell)
      {if (!op_subq_empty(QUEUE))
            {pksize = op_subq_stat(QUEUE, OPC_QSTAT_PKSIZE);
            if ((count + pksize) > cell)
                  {nm_sent_data += cell - count;
                  pksize = cell - count;
         outbuffer0_data += cell -count;
                  }
            else
                  {nm_sent_data += pksize;
          outbuffer0_data += pksize;
          }

/* send up to C packets (max. capacity); */

            for (i=1; i<=pksize; i++)
                  {pkptr = op_subq_pk_remove(QUEUE, OPC_QPOS_HEAD);
                  tmp_delay = op_sim_time() - op_pk_creation_time_get(pkptr);
                  delay += tmp_delay;
                  square_delay += (tmp_delay * tmp_delay);
                  op_pk_destroy(pkptr);
                  queued_data--;
                  }
            }
      }

if (rtm0 != 0)
      {if (rtm0 > cell)
            {rtdm0=cell;
            qlossm0 += rtm0-cell;
            }
      else rtdm0=rtm0;
   }
if (rtdm0 < cell)
      {if (!op_subq_empty(1));
            {dadm0=op_subq_stat(1,OPC_QSTAT_PKSIZE);
            if ((rtdm0+dadm0) > cell)
                  {dam0=cell-rtdm0;
```

```
                    nm_sent_datam += cell-rtdm0;
                    }
              else
                    {dam0=dadm0;
                    nm_sent_datam += dadm0;
                    }
              for (i=1;i<=dam0;i++)
                    {pkptr=op_subq_pk_remove(1,OPC_QPOS_HEAD);
                    op_pk_nfd_get(pkptr, "cr_time", &cr_time);
                    tmp_delay = op_sim_time()-cr_time;
                    delaym += tmp_delay;
                    square_delaym += (tmp_delay*tmp_delay);
                    op_pk_destroy(pkptr);
                    }
              }
        }

if (rm0 != 0)
        {if (rm0 > cell)
              {rtdn0=cell;
              qlossn0 += rm0-cell;
              }
        else rtdn0=rm0;
  }
if (rtdn0 < cell)
        {if (!op_subq_empty(2));
              {dadn0=op_subq_stat(2,OPC_QSTAT_PKSIZE);
              if ((rtdn0+dadn0) > cell)
                    {dan0=cell-rtdn0;
                    nm_sent_datan += cell-rtdn0;
                    }
              else
                    {dan0=dadn0;
                    nm_sent_datan += dadn0;
                    }
              for (i=1;i<=dan0;i++)
                    {pkptr=op_subq_pk_remove(2,OPC_QPOS_HEAD);
                    op_pk_nfd_get(pkptr, "cr_time", &cr_time);
                    tmp_delay = op_sim_time()-cr_time;
                    delayn += tmp_delay;
                    square_delayn += (tmp_delay*tmp_delay);
                    op_pk_destroy(pkptr);
                    }
              }
        }

if (rto0 != 0)
        {if (rto0 > cell)
              {rtdo0=cell;
              qlosso0 += rto0-cell;
              }
        else rtdo0=rto0;
  }
if (rtdo0 < cell)
        {if (!op_subq_empty(3));
              {dado0=op_subq_stat(3,OPC_QSTAT_PKSIZE);
              if ((rtdo0+dado0) > cell)
                    {dao0=cell-rtdo0;
                    nm_sent_datao += cell-rtdo0;
                    }
               else
                    {dao0=dado0;
                    nm_sent_datao += dado0;
                    }
               for (i=1;i<=dao0;i++)
                    {pkptr=op_subq_pk_remove(3,OPC_QPOS_HEAD);
                    op_pk_nfd_get(pkptr, "cr_time", &cr_time);
                    tmp_delay = op_sim_time()-cr_time;
                    delayo += tmp_delay;
                    square_delayo += (tmp_delay*tmp_delay);
                    op_pk_destroy(pkptr);
                    }
               }
          }

if (rt1 != 0)
     {if (rt1 > cell)
          {rtd1 = cell;
          qloss1 += rt1-cell;
          }
     else rtd1 = rt1;
     }
if (rtd1 < cell)
     {if (dad1 != 0)
          {if ((rtd1 + dad1) > cell) dad1 = dad1 - (cell - rtd1);
          else dad1 = 0;
          }
     }
if (rtm1 != 0)
     {if (rtm1 > cell)
          {rtdm1 = cell;
          qlossm1 += rtm1-cell;
          }
     else rtdm1 = rtm1;
     }
if (rtdm1 < cell)
     {if (dadm1 != 0)
          {if ((rtdm1 + dadm1) > cell) dadm1 = dadm1 - (cell - rtdm1);
          else dadm1 = 0;
          }
     }
if (rtn1 != 0)
     {if (rtn1 > cell)
          {rtdn1 = cell;
          qlossn1 += rtn1-cell;
          }
     else rtdn1 = rtn1;
     }
if (rtdn1 < cell)
     {if (dadn1 != 0)
          {if ((rtdn1 + dadn1) > cell) dadn1 = dadn1 - (cell - rtdn1);
          else dadn1 = 0;
          }
     }
if (rto1 != 0)
     {if (rto1 > cell)
          {rtdo1 = cell;
          qlosso1 += rto1-cell;
```

```
            }
        else rtdo1 = rto1;
        }
    if (rtdo1 < cell)
        {if (dado1 != 0)
            {if ((rtdo1 + dado1) > cell) dado1 = dado1 - (cell - rtdo1);
            else dado1 = 0;
            }
        }

    if (rt2 != 0)
        {if (rt2 > cell)
            {rtd2 = cell;
            qloss2 += rt2-cell;
            }
        else rtd2 = rt2;
        }
    if (rtd2 < cell)
        {if (dad2 != 0)
            {if ((rtd2 + dad2) > cell) dad2 = dad2 - (cell - rtd2);
            else dad2 = 0;
            }
        }

    if (rtm2 != 0)
        {if (rtm2 > cell)
            {rtdm2 = cell;
            qlossm2 += rtm2-cell;
            }
        else rtdm2 = rtm2;
        }
    if (rtdm2 < cell)
        {if (dadm2 != 0)
            {if ((rtdm2 + dadm2) > cell) dadm2 = dadm2 - (cell - rtdm2);
            else dadm2 = 0;
            }
        }
    if (rtn2 != 0)
        {if (rtn2 > cell)
            {rtdn2 = cell;
            qlossn2 += rtn2-cell;
            }
        else rtdn2 = rtn2;
        }
    if (rtdn2 < cell)
        {if (dadn2 != 0)
            {if ((rtdn2 + dadn2) > cell) dadn2 = dadn2 - (cell - rtdn2);
            else dadn2 = 0;
            }
        }
    if (rto2 != 0)
        {if (rto2 > cell)
            {rtdo2 = cell;
            qlosso2 += rto2-cell;
            }
        else rtdo2 = rto2;
        }
    if (rtdo2 < cell)
        {if (dado2 != 0)
```

```
                {if ((rtdo2 + dado2) > cell) dado2 = dado2 - (cell - rtdo2);
                else dado2 = 0;
                }
        }

if (rt3 != 0)
        {if (rt3 > cell)
        {rtd3 = cell;
                qloss3 += rt3-cell;
                }
        else rtd3 = rt3;
        }
if (rtd3 < cell)
        {if (dad3 != 0)
                {if ((rtd3 + dad3) > cell) dad3 = dad3 - (cell - rtd3);
                else dad3 = 0;
                }
        }
if (rtm3 != 0)
        {if (rtm3 > cell)
                {rtdm3 = cell;
                qlossm3 += rtm3-cell;
                }
        else rtdm3 = rtm3;
        }
if (rtdm3 < cell)
        {if (dadm3 != 0)
                {if ((rtdm3 + dadm3) > cell) dadm3 = dadm3 - (cell - rtdm3);
                else dadm3 = 0;
                }
        }
if (rtn3 != 0)
        {if (rtn3 > cell)
                {rtdn3 = cell;
                qlossn3 += rtn3-cell;
                }
        else rtdn3 = rtn3;
        }
if (rtdn3 < cell)
        {if (dadn3 != 0)
                {if ((rtdn3 + dadn3) > cell) dadn3 = dadn3 - (cell - rtdn3);
                else dadn3 = 0;
                }
        }
if (rto3 != 0)
        {if (rto3 > cell)
                {rtdo3 = cell;
                qlosso3 += rto3-cell;
                }
        else rtdo3 = rto3;
        }
if (rtdo3 < cell)
        {if (dado3 != 0)
                {if ((rtdo3 + dado3) > cell) dado3 = dado3 - (cell - rtdo3);
                else dado3 = 0;
                }
        }

rt1 = 0; rt2 = 0; rt3 = 0;
```

```
dal = 0; da2 = 0; da3 = 0;
rnm0=0; rn0=0; rto0=0; rnm1=0; rn1=0; rto1=0;
rnm2=0; rn2=0; rto2=0; rnm3=0; rn3=0; rto3=0;
dam0=0; dan0=0; dao0=0; dam1=0; dan1=0; dao1=0;
dam2=0; dan2=0; dao2=0; dam3=0; dan3=0; dao3=0;

avg_delay = delay/nm_sent_data;
avg_delaym = delaym/nm_sent_datam;
avg_delayn = delayn/nm_sent_datan;
avg_delayo = delayo/nm_sent_datao;

dadm0=op_subq_stat(1.OPC_QSTAT_PKSIZE);
dadn0=op_subq_stat(2.OPC_QSTAT_PKSIZE);
dado0=op_subq_stat(3.OPC_QSTAT_PKSIZE);


fprintf(fp."10 %d %d %d %d\n". out0_realtime. out0_data.
    outbuffer0_data. outbuffer0_realtime);
fprintf(fp."11 %11.9f %11.9f %d %11.9f\n". queued_data. avg_delay.
    dadm0. avg_delaym);
fprintf(fp."12 %d %11.9f %d %11.9f\n". dadn0. avg_delayn.
    dado0. avg_delayo);
fprintf(fp."13 %d %d %d 0\n".dad1.dad2.dad3);
fprintf(fp."14 %d %d %d 0\n".dadm1.dadm2.dadm3);
fprintf(fp."15 %d %d %d 0\n".dadn1.dadn2.dadn3);
fprintf(fp."16 %d %d %d 0\n".dado1.dado2.dado3);

out0_realtime=0;
out0_data=0;

/* schedule next frame time */

op_intrpt_schedule_self(op_sim_time() + frame. FRAME_CODE);
```

transition **frame_time -> wait**

attribute	*value*	*type*	*default value*
name	tr_9	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

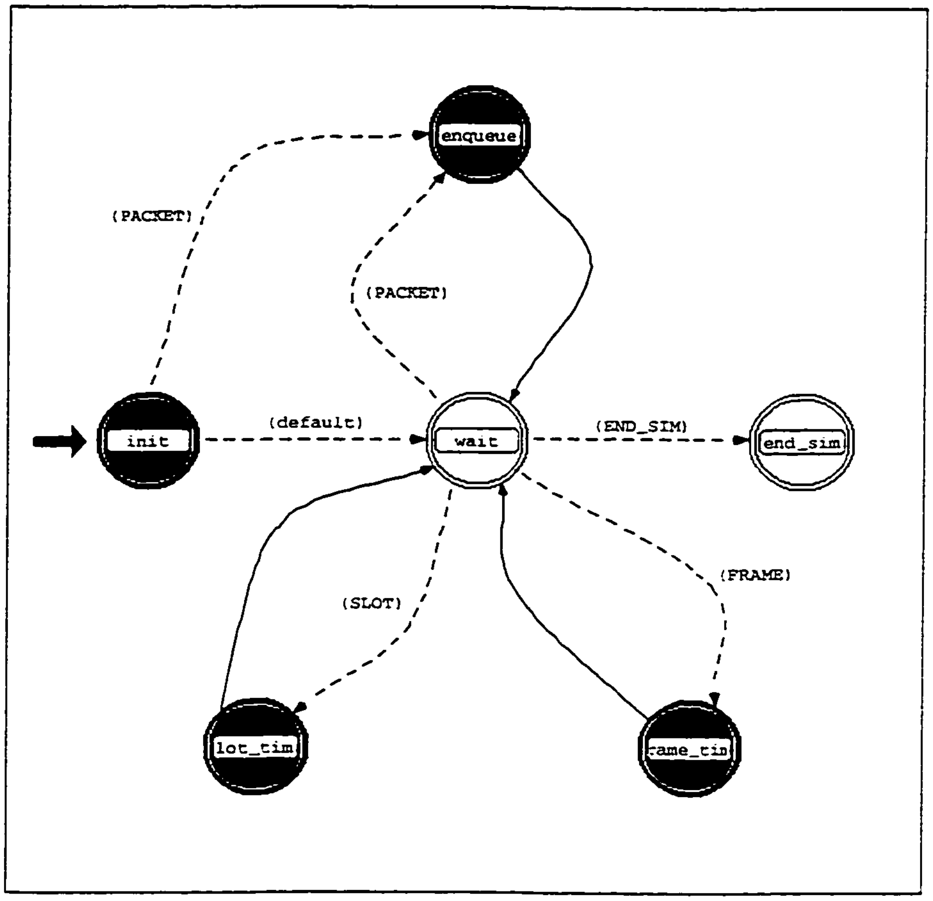

Figure A.5: Process model of the switch

...
...

Header Block

```
#include<math.h>
#include<stdio.h>
#include<sys/time.h>

#define N_inputs      4
#define C             512
#define frame_time    0.024
#define REAL          0
#define DATA          1
#define O_Q_STRM      0
#define Q_EMPTY       (op_q_empty())
#define FRAME_CODE      3
#define PACKET   op_intrpt_type() == OPC_INTRPT_STRM
#define FRAME    op_intrpt_type() == OPC_INTRPT_SELF &&\
                 op_intrpt_code() == FRAME_CODE
#define END_SIM op_intrpt_type() == OPC_INTRPT_ENDSIM
```

State Variable Block

```
/*    dest_addr: address of the destination of the packet
      type: type of packet (1: video. 2: voice. 3: data)
      queued_pk: keep track of # of packets queued
      queued_real: # of real packets waiting to be transmitted in next frame
      queued_data: # of data packets actually queued
      total_delay_data: total delay through whole system of data packets
      avg_delay_real: average delay through whole system of real packets
      avg_delay_data: average delay through whole system of data packets
      nm_sent_real: # of real packets that got through the output buffer
      nm_sent_data: # of data packets that got through the output buffer
      nm_loss: # of real packets blocked
      avg_loss: average packet loss
      nm_total: total # of packets
      tmp_delay: delay of one packet
      total_square_delay: to calculate expected value of square
      variance_delay: variance of delay (expected value of square - square of expected value)
*/

int \dest_addr;
int \type;
int \queued_pk;
int \queued_real;
int \queued_data;
double \total_delay_data;
double \avg_delay_data;
double \nm_loss;
double \avg_loss;
double \nm_total;
double \nm_sent_real;
double \nm_sent_data;
double \tmp_delay;
double \total_square_delay;
double \variance_delay;
int \video;
int \voice;
int \data;
int \frame_count;
```

Temporary Variable Block

```
packet *pkptr;
int i;
int count;
int pksize;
int voice_sent_per_frame;
int data_sent_per_frame;
int video_sent_per_frame;
```

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**

```
/* initialization of parameters */

video = 0;
voice = 0;
data = 0;
queued_pk = 0;
queued_real = 0;
queued_data = 0;
total_delay_data = 0.0;
avg_delay_data = 0.0;
nm_loss = 0.0;
avg_loss = 0.0;
nm_total = 0.0;
nm_sent_real = 0.0;
nm_sent_data = 0.0;
tmp_delay = 0.0;
total_square_delay = 0.0;
variance_delay = 0.0;
frame_count = 0;

/* schedule the first frame going out of the buffer */

op_intrpt_schedule_self(op_sim_time() + frame_time, FRAME_CODE);
```

transition **init -> wait**

attribute	*value*	*type*	*default value*
name	tr_0	string	tr
condition	default	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...

...

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

unforced state **wait**			
attribute	*value*	*type*	*default value*
name	wait	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **wait -> end_sim**			
attribute	*value*	*type*	*default value*
name	tr_1	string	tr
condition	END_SIM	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

transition **wait -> send**			
attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition	FRAME	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

...
...

enter execs **enqueue**

```
/* get info about incoming packet */

pkptr = op_pk_get(op_intrpt_strm());
op_pk_nfd_get(pkptr, "type", &type);
queued_pk++;
nm_total++;

switch(type)
{case 1:
     op_subq_pk_insert(REAL, pkptr, OPC_QPOS_TAIL);
     queued_real++;
     video++;
     break;
case 2:
     op_subq_pk_insert(REAL, pkptr, OPC_QPOS_TAIL);
     queued_real++;
     voice++;
     break;
case 3:
     op_subq_pk_insert(DATA, pkptr, OPC_QPOS_TAIL);
     queued_data++;
     data++;
     break;
}
```

transition **enqueue -> wait**

attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **send**

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

enter execs **send**

```
/* reinitialize count, the counter that keeps track of
the number of packets served during the present frame time */

count = 0;
frame_count++;

/* we only serve if the queues are not empty */
/* we start serving the real subqueue first, then, if
capacity allows us, we serve the data subqueues. */

voice_sent_per_frame=0;
```

```
video_sent_per_frame=0;
data_sent_per_frame=0;

if (!Q_EMPTY)
      {if (!op_subq_empty(REAL))
            {pksize = op_subq_stat(REAL, OPC_QSTAT_PKSIZE);
            queued_pk -= pksize;

/* case where real packets exceed capacity */

            if (pksize > C)
                  {nm_loss += pksize - C;
                  nm_sent_real += C;
                  pksize = C;
                  count = C;
                  }
            else
                  {nm_sent_real += pksize;
                  count = pksize;
                  }

/* send up to C packets (max. capacity) */

            for (i=1; i<=pksize; i++)
                  {pkptr = op_subq_pk_remove(REAL, OPC_QPOS_HEAD);
          op_pk_nfd_get(pkptr, "type", &type);
          switch(type)
           {case 1:
             video_sent_per_frame++;
             break;
            case 2:
             voice_sent_per_frame++;
             break;
           }
                  op_pk_send(pkptr, O_Q_STRM);
                  }
            op_subq_flush(REAL);
            queued_real = 0;
            }

/* we now check the data subqueue */

      if (!op_subq_empty(DATA))
            {pksize = op_subq_stat(DATA, OPC_QSTAT_PKSIZE);

/* check if we still have capacity */
/* if yes, serve up to capacity */

            if (count < C)
                  {if ((count + pksize) > C)
                        {nm_sent_data += C - count;
                        pksize = C - count;
                        }
                  else
                        nm_sent_data += pksize;

/* send up to C packets (max. capacity): */

                  for (i=1; i<=pksize; i++)
```

...
...

```
                    {pkptr = op_subq_pk_remove(DATA, OPC_QPOS_HEAD);
                    tmp_delay = op_sim_time() - op_pk_creation_time_get(pkptr);
                    total_delay_data += tmp_delay;
                    total_square_delay += (tmp_delay * tmp_delay);
                    op_pk_send(pkptr, O_Q_STRM);
            data_sent_per_frame++;
                    queued_pk--;
                queued_data--;
                    }
                }
            }
        }

printf("G %d %d %d %d %d %d %d\n", video, voice, data,
                    data_sent_per_frame,
                    voice_sent_per_frame,
                    video_sent_per_frame,
                        queued_data);

video=0;
voice=0;
data=0;

/* schedule next frame time */

op_intrpt_schedule_self(op_sim_time() + frame_time, FRAME_CODE);
```

transition **send -> wait**

attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **end_sim**

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

enter execs **end_sim**

```
avg_loss = nm_loss/nm_total;
avg_delay_data = total_delay_data/nm_sent_data;
variance_delay = (total_square_delay/nm_sent_data) - (avg_delay_data * avg_delay_data);

printf("%% ***************************\n");
printf("%%      INPUT QUEUE RESULTS    \n");
printf("%% ***************************\n");
printf("%% video = %d\n", video);
printf("%% voice = %d\n", voice);
printf("%% data = %d\n", data);
```

...

...

```
printf("%% average loss = %11.9f\n", avg_loss);
printf("%% delay from creation to end input queue\n");
printf("%% average delay (data) = %11.9f\n", avg_delay_data);
printf("%% delay variance (data) = %11.9f\n", variance_delay);
printf("%% total arrived to buffer = %f\n", nm_total);
printf("%% in queue = %d\n", queued_pk);
printf("%% real in queue = %d\n", queued_real);
printf("%% data in queue = %d\n", queued_data);
printf("%% real packets sent = %f\n", nm_sent_real);
printf("%% data packets sent = %f\n", nm_sent_data);
printf("%% total loss (blocked) = %f\n", nm_loss);
```