Titre: Title:	Performance of a multicast packet switch for Broadband ISDN
Auteur: Author:	Richard Breault
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Breault, R. (1989). Performance of a multicast packet switch for Broadband ISDN [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/56717/

UNIVERSITE DE MONTREAL

PERFORMANCE OF A MULTICAST PACKET SWITCH FOR BROADBAND ISDN

par

Richard Breault
DEPARTEMENT DE GENIE ELECTRIQUE
ECOLE POLYTECHNIQUE

MEMOIRE PRESENTE EN VUE DE L'OBTENTION
DU GRADE DE MAITRE EN INGENIERIE (M.Ing.)

MARS 1989

ISBN 0-315-50185-5

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE

Ce mémoire intitulé:

PERFORMANCE OF A MULTICAST PACKET SWITCH
FOR BROADBAND ISDN

présenté par: Richard Breault
en vue de l'obtention du grade de: MAITRE EN INGENIERIE (M.Ing.)
a été dûment accepté par le jury d'examen constitué de:

Mme Catherine Rosenberg, Ph.D., présidente
M. Jeremiah F. Hayes, Ph.D.
M. Jean-Louis Houle, Ph.D.
M. Marc Comeau, Ph.D.

SOMMAIRE

L'objectif du développement d'un réseau numérique à intégration de services (RNIS) à large bande est d'offrir un support universel et transparent pour le transport de l'information. A cause de l'incertitude quant à la quantité et la nature des services qui devront être supportés, la solution adoptée devra être évolutive au moindre coût.

La commutation temporelle asynchrone ("ATM") semble être la solution la plus prometteuse pour satisfaire les exigences mentionnées. L'"ATM" requiert cependant l'utilisation de commutateurs ultra-rapides permettant la multiplication de paquets. Ce type de commutateur est nécessaire pour supporter les services de diffusion et de multiconnexion en plus des services point-à-point standards.

Le but de ce mémoire est l'évaluation de performance d'un commutateur à diffusion et à diffusion restreinte de paquets.

Le chapitre 2.0 présente une liste sommaire des services envisagés pour le RNIS à large bande et énumère les concepts fondamentaux associés à la commutation temporelle asynchrone (ATM). L'architecture d'un

commutateur de paquets pour le RNIS à large bande est ensuite décrit brièvement.

Le chapitre 3.0 propose une analyse mathématique basée sur la théorie des files d'attente òu chaque port d'entrée du commutateur est représenté par une queue du type M/G/1. Cette analyse permet d'évaluer le délai moyen encouru par un paquet traversant le commutateur ainsi que le débit maximum supporté. Le commutateur modèlisé possède NI ports d'entrée et NO ports de sortie et son fonctionnement est le suivant:

Un paquet arrivant en tête de queue à un port d'entrée génère un nombre aléatoire de copies qui sont distribuées vers les ports de sortie. Les conflits d'accès aux ports de sortie, engendrés par plusieurs copies provenant de ports d'entrée différents, sont résolus en choisissant une copie au hasard parmi les copies en conflit. Les copies n'ayant pas obtenu l'accès aux ports de sortie sont retransmies durant le prochain intervalle de temps. En conséquence le temps de service d'un paquet traversant le commutateur est défini comme étant le temps nécessaire pour que l'ensemble des copies générées obtiennent l'accès à leur port de sortie respectif. Le délai moyen d'un paquet est obtenu à partir du temps moyen de service à l'aide de la formule de Pollaczek-Khinchin.

Trois modèles de diffusion de paquets sont analysés. Dans le premier modèle, chaque paquet distribue ses copies en faisant des essais de Bernouilli indépendants avec probabilité P pour chaque port de sortie. Le nombre de copies ainsi générées suit une loi binômiale de paramètres NO et P. Pour le deuxième modèle, chaque paquet génère un nombre de copies suivant une distribution binômiale modifiée de paramètres NO et P. Les copies provenant de chaque port d'entrée sont ensuite regroupées et distribuées aux ports de sortie suivant une distribution multinômiale.

Le troisième modèle diffère du deuxième non pas par la méthode de diffusion des copies mais plutôt par la méthode d'évaluation du niveau de congestion aux ports de sortie du commutateur. Ce modèle considère la distribution du nombre de copies résiduelles plutôt que la distribution du nombre de copies générées pour déterminer le nombre de copies en conflit. Enfin la dernière section du chapitre d'analyse propose une approximation pour un commutateur ayant plusieurs lignes de transmission à chaque port de sortie.

Le chapitre 4.0 traite de l'aspect simulation du commutateur. Le modèle proposé est du type Monte Carlo à intervalle de temps discret ôu l'arrivée des paquets, la diffusion des copies et la résolution des conflits aux ports de sortie sont modélisés par différentes fonctions de

probabilité. Comme pour l'analyse mathématique, la simulation permet d'évaluer le délai moyen d'un paquet traversant le commutateur.

La validité du modèle est assurée en adaptant celui-ci de façon à simuler un commutateur point-à-point pour lequel certains résultats analytiques sont connus. Un intervalle de confiance est calculé pour déterminer le nombre de simulations ainsi que le nombre d'échantillons nécessaires pour obtenir le niveau de précision requis.

Les résultats sont analysés et comparés au chapitre 5.0 pour des commutateurs de dimensions et configurations différentes. Ces résultats prouvent la validité du modèle analytique avec distribution de copies résiduelles. Ce modèle analytique permet d'obtenir une limite inférieur au délai moyen d'un paquet traversant le commutateur. Cette limite s'approche des résultats de simulation lorsque les dimensions du commutateur augmentent.

ABSTRACT

The major goal of Broadband ISDN is to support a wide range of communication services over a common transport network. To satisfy this goal, an integrated packet network based on statistical multiplexing has been proposed. One key component of such a network is the multicast switch capable of packet replication. This type of switch is needed to support multipoint as well as point-to-point communication services.

The main objective of this thesis is to evaluate the performance of a multicast packet switch. We first propose a queuing analysis where each input port is modeled as an M/G/1 queue. Performance estimates in terms of throughput and average packet delay are obtained for different packet replication techniques and for different traffic loading consideration.

A simulation model of the multicast packet switch is developed in chapter 4.0. We use a discrete time self-driven simulation where packet arrivals, packet replication, and conflicts resolution at output ports are modeled by probability distributions. The simulation model is validated by comparison to some published performance results.

Analytical and simulation results are obtained and compared in chapter 5.0 for various switch configurations and sizes.

ACKNOWLEDGEMENTS

I would like to thank Dr. J. F. Hayes for giving me the opportunity to work on this research subject and for his help and guidance during these two years. I also recognize the contribution of Dr. M. K. Mehmet-Ali who's suggestions were very helpful.

Thanks are also due to Dr. J. L. Houle for making this co supervision possible and to the CRIM organization for giving me the opportunity to use their computing facilities.

Finally I would like to mention the patience and support of my wife Hélène during long evenings of work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

This thesis presents the performance evaluation of a Multicast Packet Switch for Broadband ISDN (BISDN).[1] Switches capable of packet replication are needed for supporting multi-point services in a broadband packet switch network. To obtain some performance estimates for this type of switch, we propose an analysis based on a theoric model of the system.

The main body of the thesis begins in chapter two, where we first introduce the BISDN concepts and enumerate some services that are expected to be supported. We then describe the transport technique that has been proposed for supporting these services and the architecture of a typical Broadband Packet Switch with multicast capability.

A performance analysis is developed in chapter 3.0 where maximum throughput and average packet delay are evaluated. Different approaches for modeling the replicating technique and the service discipline are

---

[1] This work was started by one of the co supervisor of the thesis, Dr. J. F. Hayes, while he was at Bell Communication Research, Morristown, NJ, during the summer of 1987.

treated. In all cases, a queuing model is used where each input port is modeled as an M/G/1 queue.

Chapter four deals with the simulation of the multicast packet switch. We describe the simulation model in terms of the random variables generation and the data structures used. The simulation model is validated by comparison to some published performance results. The confidence interval method is used to assure the results accuracy.

Finally in chapter five we look at the different results obtained from the analysis and we compare them with the simulation results.

2. Information transport in Broadband ISDN

Recently, the I series recommendations for an Integrated Services Digital Network (ISDN) has been approved by the CCITT. This ISDN network should provide an end-to-end digital connectivity to support voice and non-voice services through a limited set of interfaces (basic access and primary rate access). While this first version of ISDN is presently in field trials in several countries, there is a wide agreement that the circuit switched approach and the limited bandwidth of present ISDN can not meet the long term requirements of multimedia and multirate communications. To satisfy the future needs of communication services, Broadband ISDN has been proposed.

The goal of Broadband ISDN (BISDN) is to provide an integrated transport network for a wide range of services. Each of these services may have very different traffic characteristics in terms of bit rate and burstiness and may require point-to-point or multi-point support. Estimates of traffic characteristics for some services are given in table 2.1, where the burstiness of a traffic source is defined as the ratio between the time during which information is sent and the time for which the channel is held.

The two key design issues for supporting this wide

range of services are bandwidth availability and flexibility. To satisfy these requirements, one proposal based on statistical multiplexing and fast cell (packet) switching has emerged as the preferred transport technique for Broadband ISDN (BISDN). This technique commonly referred to as the Asynchronous Transfer Mode (ATM) within the CCITT was originally proposed under different appellations such as Fast Packet Switching (FPS), Asynchronous Time-Division technique (ATD) or Dynamic Time-Division Multiplexing (DTDM) [1]-[4].

Services	Bit rate (bps)	Burstiness	Call duration(s)
Telephony	16k -- 64k	.3 -- .6	10^2 --- 10^3
Telemetry	10 -- 10k	.01 - .1	1 --- 10
Facsimile	1k -- 100k	1	10 --- 10^2
Videoconf.	1M -- 70M	.3 -- 1	10^3 --- 10^4
Videotex	1M -- 70M	.1 -- 1	10^2 --- 10^3
Bulk Data	1M -- 50M	.1 - .8	10 --- 10^3
Interac. data	1k -- 1M	.05 - .2	10^2 --- 10^4
LAN Intercon.	10M -- 100M	.3 -- 1	10^2 --- 10^4
Electro. mail	1k -- 100k	.2 -- 1	10 --- 10^2
TV	30M -- 70M	1	10^3 --- 10^4
HDTV	140M -- 565M	1	10^3 --- 10^4

Table 2.1 Various services characteristics

2.1 Asynchronous Transfer Mode concept

The asynchronous transfer mode (ATM) is considered to be the best transport (multiplexing and switching) technique for BISDN [5]-[6]. It is expected to have both the bandwidth flexibility of packet switching and the time transparency of circuit switching. The ATM technique is based upon the use of cells of fix length (packets) used to carry all information within the network. These cells are multiplexed in periodic time intervals called slots. For synchronization purposes the time slots can be organized in a pre-defined frame structure as in DTDM. If no framing is used (ATD), synchronization cells must be inserted in unused time slots.

ATM differs from Synchronous Time Division Multiplexing (STDM) in that the time slots are assigned dynamically rather than on a per call basis. This dynamic bandwidth allocation implies that a particular connection can no longer be identified by the position of the time slot within a frame structure but by a label contained in the header field of each cell as in packet switching. The header field can also be used for media access control, error control and for cell priority if necessary.

In order to maintain high throughput and low delay, no link-to-link flow control is involved and error detection

and correction are handled from end-to-end by higher protocol layers. This migration of protocols complexity from network to terminal has been proven feasible by virtue of the increased network reliability provided by the optical and VLSI technology. Routing within the network is performed by high speed packet switches and is based on the virtual circuit label contained in the header field of each cell. These switches use a massively parallel architecture to obtain a throughput of millions of packets per second with switching delay of a few milliseconds.

The CCITT is now proposing ATM interfaces operating at 150 Mbits/sec. and 600 Mbits/sec. for Broadband ISDN. These proposed interfaces should provide the necessary bandwidth for supporting present and future services as they become available. A number of issues still remain to be solved before a final agreement is reached on Broadband ISDN. The major ones are related to cell format and size, voice delay impact, cell lost, signaling, header organization and network monitoring. Figure 2.1 depict an example of a multi-rate ATM interface.

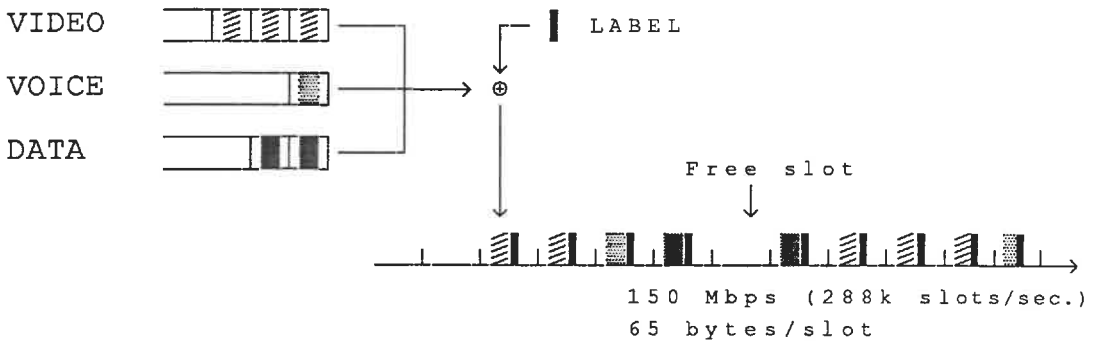
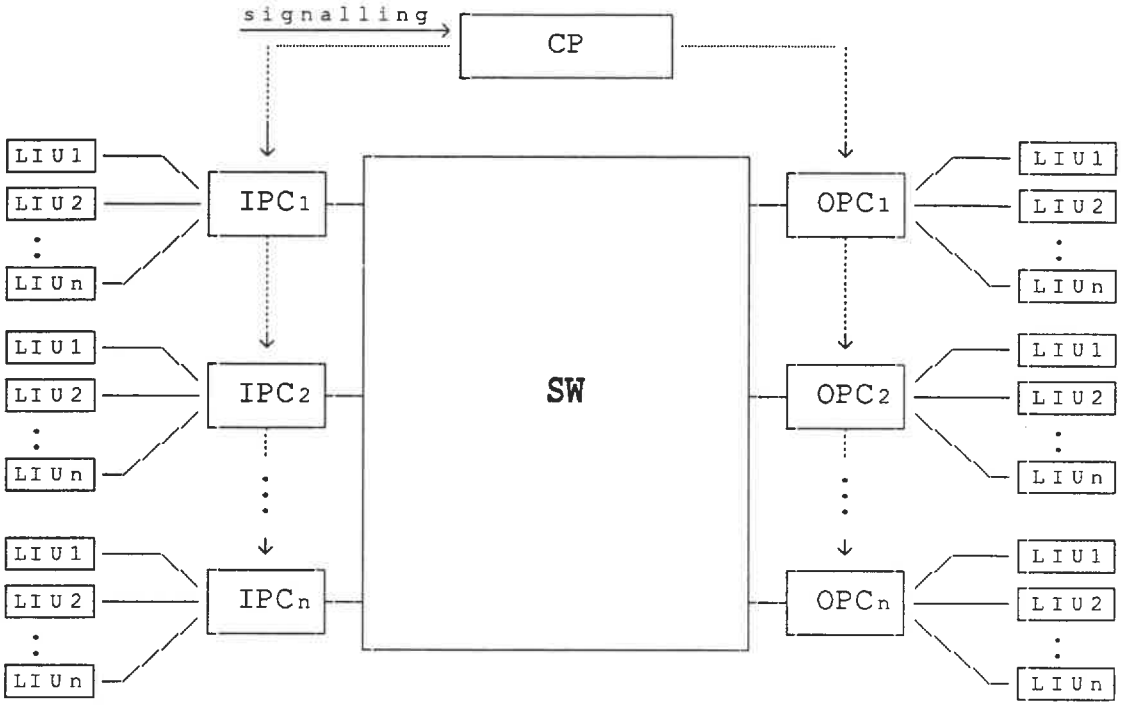


Figure 2.1 An ATM multi-rate interface

2.2 Broadband ISDN packet switch architecture

The utilization of optical fiber as a transmission media for BISDN has moved the bandwidth problem from transmission to processing within the switches. As a consequence of this, a new switching architecture called Broadband Packet Switch (BPS) has been developed. BPS will provide switching capacity in the range of megapackets per second. These switches are expected to be available by the mid-1990s. Figure 2.2 shows a Broadband Packet Switch (BPS) model.



LIU :Line Interface Unit
IPC :Input Port Controler
OPC :Output Port Controler
SW :Switching fabric
CP :Control Processor

Figure 2.2 A broadband packet switch model

The Line Interface Unit (LIU) performs the interfacing functions to high speed optical lines. It contains optical transmitters and receivers.

The Input Port Controller (IPC) and Output Port Controller (OPC) provide the queuing necessary at each port. The IPC also controls the routing through the switch by identifying the virtual circuit label of each cell. The IPC and OPC can also perform some link level protocol functions if necessary.

The Switching fabric (SW) is a space-division packet switch that can be constructed from any type of interconnection topology (crossbar,multiple bus,..etc). However, current design approaches employ a series of self-routing multistage interconnection networks (MIN) that perform the packet routing from any IPC to any OPC without blocking. The Switching fabric can also support multicast services by duplicating packets as they go through the switch. The regular organization and the self-routing property of this type of network make possible the construction of large switches without control bottlenecks and permit a cost-effective VLSI implementation.

The Control Processor (CP) handles call processing and supplies the IPCs and OPCs with routing information. The CP also performs maintenance and administrative functions.

2.2.1 Switching fabric for multicasting

The proposed point-to-point switch are usually constructed from MIN networks consisting of a combination of a Batcher network followed by a Banyan network as shown in figure 2.3 [7]-[8]. The Batcher network compacts and sorts the packets according to their destination address and the Banyan network routes the packet to the proper output port. It can be shown [9]-[10] that this arrangement guarantees that the packets will not block within the

Banyan network (internal blocking) if the requested destination addresses are all distinct. The Batcher-Banyan interconnection thus eliminates the need of internal buffering and a constant latency switch can be obtained.

A multicast packet switch can be constructed by adding a copy network in front of a routing network (point-to-point switch) as shown in figure 2.4. The function of the copy network is to replicate input packets according to the requested number of copies. The routing network then performs the final routing. The copy network can in turn be made from a combination of multistage interconnection networks (MIN) with packet replication capability.

To respect the non-blocking conditions of MIN based switches, a certain control over the traffic entering the switch is needed. This control implies the utilisation of input buffers and algorithms to deal with the following two problems:

1) Overflow of the copy network when the total number of copy requests exceeds the number of output ports.

2) Output port conflicts in the routing network when multiple packets request the same output port.

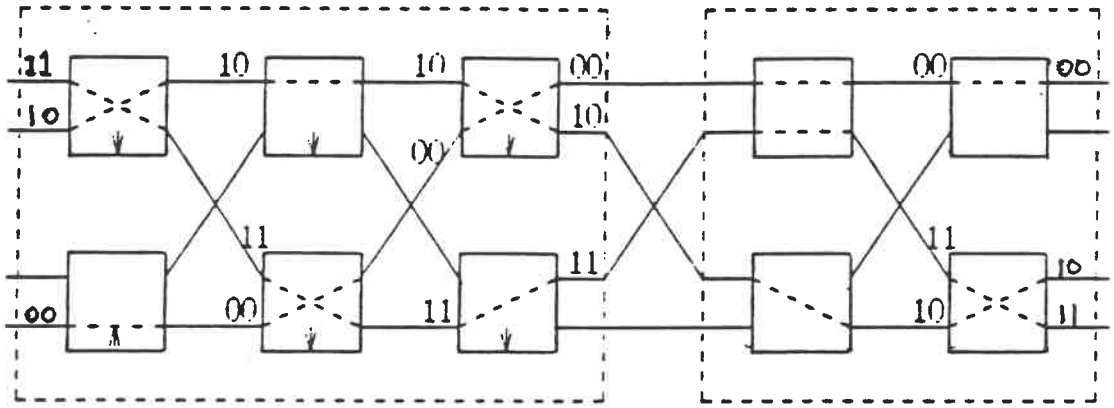


Figure 2.3 A Batcher-Banyan network

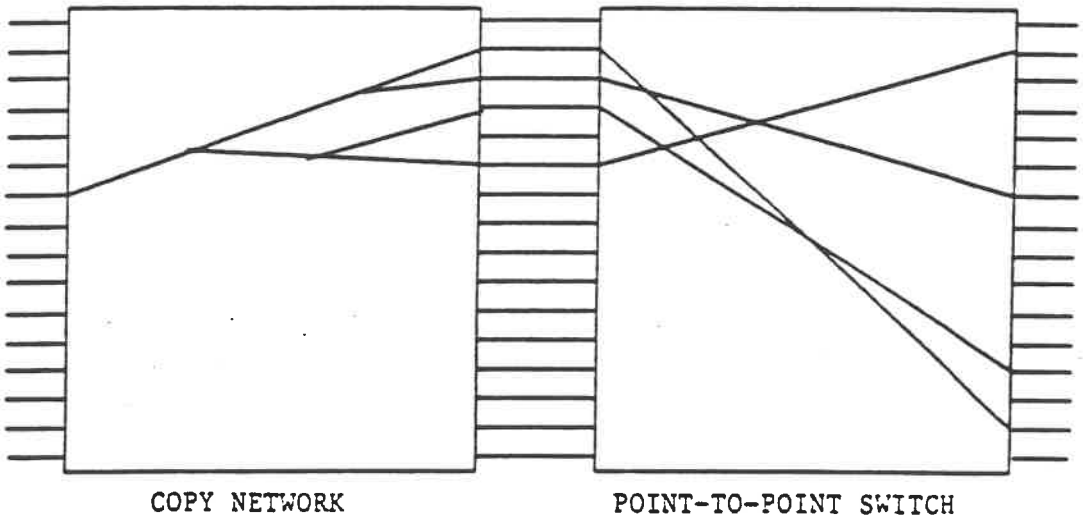


Figure 2.4 A multicast switch :a copy network and a point-to-point switch

Some multicast packet switch design proposals have emerged from research laboratories during the past few years [10]-[12]. These systems use a series of copy-routing network with some specialized hardware and algorithms to deal with copy overflow and output ports contention. In both of these cases, a retransmission process is implemented for the overflowed copies and for the copies loosing contention for access to output ports. According to this, we can affirm that in a general way the analysis of the next section can be applied to these switch architectures in order to obtain some performance evaluation, independent of implementation.

3. Analysis

This section presents the analysis of a multicast packet switch. Four different cases are developed and analyzed. In all cases the switch is viewed as a black box and no detail of internal functionality is given. The analysis is general enough to be valid for any switch with no internal blocking. This type of switch can be constructed using a crossbar structure or multistage interconnection networks as discussed in the previous section. Figure 3.1 shows the sketch of such a switch.

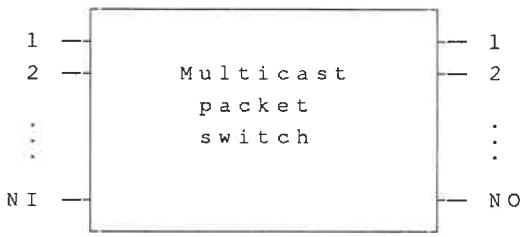


Figure 3.1 A (NIxNO) non-blocking switch

The switch has NI input ports and NO output ports. These output ports can be bundled into NT trunks consisting of S ports where NO = NTxS. We first consider the case where S = 1 and then we propose an approximation for the case where S > 1. The multicast switch model works as follow:

A packet reaching the head of an input port queue

generates a random number of copies which are switched to output ports (trunks). Only one copy of a packet is distributed to each output port (trunk). The system operates in discrete time intervals called slots. A slot corresponds to the minimum time needed to receive a packet on an input trunk. It is also assumed that a slot is the time required to switch a packet from input to output; accordingly, the durations of events in the analysis are measured in slot times. As mentioned above, the switch is assumed to be non blocking.

The property of the switch which poses the key obstacle to analysis is interference among queues; two or more input ports seek to put copies on the same output port (trunk) during the same time slot. The model assumes that conflicts are resolved by random selection, i.e., if M copies are in conflict, one of them is chosen with probability 1/M. In the case of multiple ports per output trunk , M copies are chosen with probability 1 if $M \leq S$ or S copies are chosen with probability S/M if $M > S$.

3.1 Queuing Model

Input queuing as opposed to output queuing was adopted for the switch analysis. This choice reflects more closely the multicast switch architectures proposed for broadband ISDN. As mentionned in section 2.2.1, input buffering is

needed to control traffic and to avoid packet loss in a multicast MIN based switch. Accordingly, the performance analysis is based on queuing theory where each input port is modeled as an M/G/1 queue. Figure 3.2 shows the queuing model adopted.

Since we are dealing with M/G/1 queues, the main aspect of the analysis is the derivation of the probability distribution of the service time. The service time of a packet going through the switch is defined as the number of slots necessary so that all of the copies generated by that packet have gained access to an output port. As mentioned previously, this implies that each of the generated copies has won contention at their respective output port (trunk).

The measures of performance which are the goals of the analysis are packet delay and throughput. The delay is the time elapsing between the packet arrival at an input queue and the time when all the copies that it has generated have gained access to an output port. The delay is comprised of the service time and the queuing time. The queuing time is the time that the packet resides in the queue before its service begin. The throughput is the maximum rate of packet arrivals while maintaining system stability, i.e., finite queues. The throughput is bound closely to the average service time of a packet.

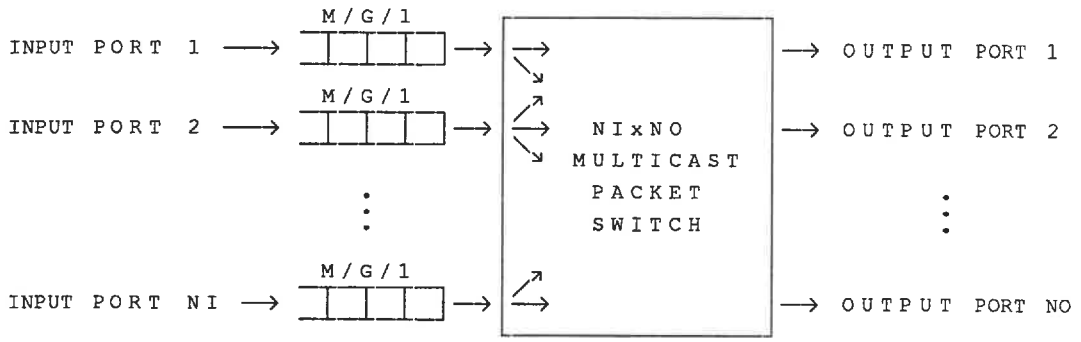


Figure 3.2 A multicast switch queuing model

3.2 Basic Assumptions

We can review the basic assumptions that we adopted for the proposed analysis:

The switch has NI input ports and NO (NTxS) output ports.
Packets arrive at input ports according to a Poisson distribution with average rate λ.
We assume that the event of an input port being occupied is independent from port to port and has probability ρ.
Packets arriving are handled first come first serve (FCFS).
We assume random selection of packets in conflict at the output ports (trunks).
We assume a non-blocking switch with constant latency.
A work conserving discipline, for the evaluation of the service duration, was adopted.
We assume a uniform traffic distribution through the switch.

3.3 Bernouilli trials traffic distribution model

In this section we find the service time and delay of a packet going through the switch. The replication model that is assumed is that of Bernouilli trials; a packet at an input port is copied to each of the output ports independently with probability P. The simplicity of the model is evident in that there is a nonzero probability that a packet at an input port will generate no copy. We are interested in the case where this probability is small. In the sequel, we consider models for which the anomaly does not occur.

3.3.1 Mathematical analysis

We begin the analysis by first focusing our attention on a particular input port. We want to find an expression for the number of copies, coming from the other NI-1 input ports, which interfere with that input port. Let i_j denote the random variable representing the number of interfering copies in output port j; j = 1,2,...NO. Clearly i_j can take values ranging from 0 to NI-1. Now let's consider a single interfering input port conditioned on it having a packet (active). The joint probability generating function for $i_1, i_2, \ldots, i_{NO}$ due to that interfering input is given by

$$I(z_1, z_2, \ldots, z_{NO} \mid \text{active}) \triangleq E\Big[\prod_{j=1}^{NO} z^{i_j} \mid \text{active}\Big] \qquad (3.1)$$

$$= \prod_{j=1}^{NO} (Pz_j + 1-P)$$

The independent trials are reflected by the product form of that equation. We can now average over the event of the interfering input port being active to obtain

$$I(z_1, z_2, \ldots, z_{NO}) = \rho\Big[\prod_{j=1}^{NO} (Pz_j + 1-P)\Big] + 1-\rho \qquad (3.2)$$

If we have NI-1 interfering input ports, each active independently, then the joint probability generating function for the total number of interfering copies is

$$T(z_1, z_2, \ldots z_{NO}) = \left(\rho\Big[\prod_{j=1}^{NO} (Pz_j + 1-P)\Big] + (1-\rho)\right)^{NI-1} \qquad (3.3)$$

Now suppose that a packet at the input port which is the focus of our attention generates K copies with $K \leq NO$. We would like to calculate the probability that L of these K copies ($L \leq K$) are chosen by the output port contention process. We begin with the assumption that these L successful copies are on a particular set of output ports. Since the process is symmetric, this set can be any set. For simplicity of exposition, we choose the first L output ports. Since the interfering copies at the output ports are

designated as $i_1, i_2, \ldots, i_{NO}$ the probability that the first L or more are chosen ($L \leq K$) given $i_1, i_2, \ldots, i_L$ is

$$\text{Prob (first L or more} \mid i_1, .., i_L, K) = \prod_{j=1}^{L} 1/ (i_j + 1) \quad (3.4)$$

The equation (3.4) reflects the fact that choices are made independently among output ports. By averaging over $i_1, i_2, \ldots, i_L$ we get

$$\text{Prob (first L or more} \mid K) = \quad (3.5)$$

$$\sum_{i_1} \sum_{i_2} \cdots \sum_{i_L} \prod_{j=1}^{L} 1/(i_j + 1)\ \text{Prob}(i_1, i_2, \ldots, i_L)\ ; L \leq K$$

Notice that the conditioning on K is manifested only through the condition $L \leq K$. Now we go back to the joint probability generating function of the interfering copies. From its definition we have

$$T(z_1, z_2, .., z_{NO}) \triangleq \quad (3.6)$$

$$\sum_{i_1} \sum_{i_2} \cdots \sum_{i_{NO}} \prod_{j=1}^{NO} z_j^{i_j}\ \text{Prob}\ (i_1, i_2, .., i_{NO})$$

If we set $z_{L+1} = z_{L+2} = \ldots = z_{NO} = 1$ and we integrate the expression over $z_1, z_2, \ldots, z_L$ we obtain

$$\int_0^1 dz_1 \int_0^1 dz_2 \ldots \int_0^1 dz_L \; T(z_1, z_2, \ldots, z_L, 1, \ldots, 1) = \tag{3.7}$$

$$\sum_{i_1} \sum_{i_2} \ldots \sum_{i_L} \prod_{j=1}^{L} z_j^{i_j + 1} / (i_j + 1) \; \text{Prob}(i_1, i_2, \ldots, i_L) \Big|_0^1$$

Thus from equations (3.7) and (3.5) and (3.3) we get

$$\text{Prob (first L or more } | \text{ K)} = \tag{3.8}$$

$$\int_0^1 dz_1 \int_0^1 dz_2 \ldots \int_0^1 dz_L \left(\rho \left[\prod_{j=1}^{L} (Pz_j + 1 - P) \right] + (1-\rho) \right)^{NI-1} ; L \le K$$

We can integrate equation (3.8) by first applying the binomial expansion and by taking advantage of the linearity of the summation and integration operator.

$$\text{Prob (first L or more } | \text{ K)} = \tag{3.9}$$

$$\sum_{n=0}^{NI-1} \binom{NI-1}{n} \rho^n (1-\rho)^{NI-1-n} \prod_{j=1}^{L} \int_0^1 dz_j \; (Pz_j + 1 - P)^n \; ; L \le K$$

$$\text{Prob (first L or more } | \text{ K)} = \tag{3.10}$$

$$\sum_{n=0}^{NI-1} \binom{NI-1}{n} \rho^n (1-\rho)^{NI-1-n} \left[\frac{1-(1-P)^{n+1}}{(n+1)P} \right]^L ; L \le K$$

At this point we have found the probability that the copies from our particular input port have won contentions at least at the first L output ports. From the symmetry of the model, this probability is the same for any other set of L or more output ports (L ≤ K). Thus equation (3.10)

gives the probability that any particular set of L or more of the K copies have been chosen at output ports. Notice that these sets are not disjoint; however they can be expressed as the union of disjoint sets. Suppose that our input port has generated K copies. We define $M_L^{(x)}$ as the event that a particular set X of L or more of the K copies get through. We also define $O_L^{(x)}$ as the event that only a particular set X of L copies get through. Since the input port that we are focusing on has generated $K \leq NO$ copies we have

$$M_K^{(x)} = O_K^{(x)} \tag{3.11a}$$

$$M_{K-1}^{(x)} = O_{K-1}^{(x)} \cup O_K^{(x)} \tag{3.11b}$$

$$M_{K-2}^{(x)} = O_{K-2}^{(x)} \cup \left(O_{K-1}^{(x)} \cup O_{K-1}^{(y)} \right) \cup O_K^{(x)} \tag{3.11c}$$

Notice that in equation (3.11c) a set y may intersect with the set x. In general we get the following expression

$$M_{K-n}^{(Z)} = \bigcup_{j=0}^{n} \left[\bigcup_{i=1}^{\binom{n}{j}} O_{K-n+j}^{(i)} \right] \tag{3.11d}$$

Since the sets on the right hand side of equation (3.11d) represent disjoint events, we can write the related probabilities. Notice that all of the sets of the same size have equal probability, we can then express the equation for any particular set by dropping the superscripts and by

replacing (K-n) by L.

$$\text{Prob}(M_L \mid K) = \sum_{i=0}^{K-L} \binom{K-L}{i} \text{Prob}(O_{L+i} \mid K) \qquad (3.12)$$

By inverting this equation we can get the probability of any particular set of L copies getting through.

$$\text{Prob}(O_L \mid K) = \sum_{i=0}^{K-L} \binom{K-L}{i} (-1)^i \text{Prob}(M_{L+i} \mid K) \qquad (3.13)$$

Up to this point we have been dealing exclusively with a particular set of L output ports chosen from K. We can find the probability for any of the possible sets of L of K copies. Let us define this probability to be Prob(L | K). Since there are $\binom{K}{L}$ possible combinations of L copies on a total of K copies, we get the following expression.

$$\text{Prob}(L \mid K) = \binom{K}{L} \text{Prob}(O_L \mid K) \qquad (3.14)$$

A work conserving discipline implies that if L copies get through in the first slot, an attempt is made to get the remaining K-L copies through in the next slot. This process continues until all of the copies have been successfully transmitted. Let Q(n | K) denote the

probability that n slots are required to transmit the K copies generated by a packet. We then have the following equations:

$$Q(1 \mid K) = \text{Prob} (K \mid K) \qquad (3.15a)$$

$$Q(n \mid K) = \sum_{L=0}^{K} \text{Prob}(L \mid K) \; Q(n-1 \mid K-L) \qquad (3.15b)$$

Q(n) can then be evaluate by averaging over K, the number of copies generated by our input port. For the Bernouilli trials traffic distribution model, the random variable K is binomially distributed with parameters NO and P.

$$Q(n) = \sum_{j=0}^{NO} \text{Prob} (K = j) \; Q(n \mid j) \qquad (3.16)$$

$$\text{Prob}(K = j) = \binom{NO}{j} P^{j} \; (1-P)^{NO-j} \qquad (3.17)$$

The average service time and the second moment are given by

$$\bar{S} = \sum_{n=0}^{\infty} n \; Q(n) \qquad (3.18a) \qquad\qquad \overline{S^2} = \sum_{n=0}^{\infty} n^2 \; Q(n) \qquad (3.18b)$$

Our derivation of the service time distribution assumes a particular value of ρ, the probability that an input port is busy. A packet that comes to a busy input port waits in the queue until its service begins. In this

case, the probability that an input port is busy is $\rho = \lambda \overline{S}$ where λ is the packet arrival rate. We can evaluate the average delay with the Pollaczek-Khinchin formula [13].

$$\bar{D} = \bar{S} + \lambda \overline{S^2} / \, 2(1-\rho) \qquad (3.19)$$

The results presented in chapter 5.0 are found for different values of λ by solving

$$\lambda = \rho / \, \overline{S} \qquad (3.20)$$

3.4 Multinomial traffic distribution model

This section presents the analysis of the multinomial traffic distribution model. The analysis follows closely that of the Bernouilli traffic distribution model presented in the previous section. The main difference from the previous analysis resides in the way the traffic is distributed from the input ports to the output ports. A packet reaching the head of an input port queue generates a random number of copies. This random variable can follow an arbitrary distribution. To avoid the drawback associated with the binomial distribution we choose a modified binomial distribution. The modified binomial distribution has the advantage that at least 1 copy is always generated for every packet.

The copies generated by each input port are then pooled together and distributed to the output ports according to a multinomial distribution where each output port has equal probability of being chosen. The choice of the multinomial distribution in the context of this analysis has one drawback in that there is a non zero probability of two or more copies from the same input port going to the same output port. For relatively large switches we can assume that this probability is very small.

3.4.1 Mathematical analysis

We again focus our attention on a particular input port. We define the random variable M as the number of copies generated by one of the NI-1 interfering input ports. We want to find the probability distribution and probability generating function of this random variable by conditioning on the input port being active. The probability that i copies are generated (M = i) is given by equation (3.21) where i can take values ranging from 1 to NO.

$$\text{Prob}(M=i \mid \text{active}) = \binom{NO}{i} \frac{P^i (1-P)^{NO-i}}{1 - (1-P)^{NO}} \quad ;1 \le i \le NO \qquad (3.21)$$

The probability generating function of the modified binomial distribution is given by equation (3.22). This expression is developed in Appendix A.

$$M(z \mid \text{active}) = \frac{\left(Pz + (1-P) \right)^{NO} - (1-P)^{NO}}{1 - (1-P)^{NO}} \qquad (3.22)$$

Notice that $\left(Pz + (1-P) \right)^{NO}$ is the probability generating function of the binomial distribution with parameters P and NO. If ρ is the probability of an input port being active, we can average over that event.

$$M(z) = \rho \left[\frac{\left(Pz + (1-P) \right)^{NO} - (1-P)^{NO}}{1 - (1-P)^{NO}} \right] + (1-\rho) \qquad (3.23)$$

There are NI-1 input ports generating copies independently. The total number of conflicting copies is then the sum of all the copies generated by these input ports. The probability generating function of the total number of conflicting copies seen by our selected input port is then

$$M_T(z) = \left[\rho \left[\frac{\left(Pz + (1-P) \right)^{NO} - (1-P)^{NO}}{1 - (1-P)^{NO}} \right] + (1-\rho) \right]^{NI-1} \qquad (3.24)$$

We consider now our selected input port which has generated K copies. Each of these copies will suffer interference from the NI-1 other input ports. We denote these interfering copies in each of these K output ports as $i_1, i_2, \ldots, i_K$. We want to find the probability distribution of these interfering copies by conditioning on the NI-1 ports generating M_T copies. Notice that i_{K+1} contains all the copies of non conflicting output ports. Clearly the number of ways in which the M_T interfering copies can be divided into K+1 groups of which the first port contains i_1, the second port contains i_2, etc, is

$$\frac{M_T!}{i_1!\ i_2!\ \ldots\ i_K!\ i_{K+1}!} \qquad (3.25)$$

So $(i_1, i_2, \ldots, i_K)$ follows a multinomial distribution where 1/NO is the probability of choosing an output port.

$$\text{Prob}\ (i_1, i_2, \ldots, i_K \mid M_T) = \tag{3.26a}$$

$$(M_T)!\ \left[\frac{(1/NO)^{i_1}}{(i_1)!} \quad \frac{(1/NO)^{i_2}}{(i_2)!} \ .. \ \frac{(1/NO)^{i_K}}{(i_K)!} \quad \frac{(1- k/NO)^{i_{K+1}}}{(i_{K+1})!} \right]$$

$$\text{where} \quad i_{K+1} = \left[M_T - \sum_{j=1}^{K} i_j \right] \tag{3.26b}$$

As mentioned, the j^{th} of the K copies generated by our input port is conflicting with i_j interfering copies. We can find the probability of getting L or more of these K copies ($L \leq K$) through a particular set of output ports. As in the previous analysis we can take this set to be the first L, for example.

$$\text{Prob}\ (\text{first L or more} \mid i_1, i_2, \ldots, i_L, K) = \tag{3.27}$$

$$\prod_{j=1}^{L} 1/\ (i_j + 1) \ ; L \leq K$$

By averaging over Prob $(i_1, i_2, \ldots, i_L \mid M_T)$ we get

$$\text{Prob (first L or more} \mid M_T, K) = \tag{3.28}$$

$$\sum_{i_1} \sum_{i_2} \cdots \sum_{i_L} \prod_{j=1}^{L} 1/(i_j + 1)\ \text{Prob}\ (i_1, i_2, \ldots, i_L \mid M_T)\ ; L \leq K$$

Notice again that Prob (first L or more $\mid M_T$, K) is the same for any values of K ($K \leq NO$) and that the conditioning on K is due to the fact that L must be smaller or equal to K ($L \leq K$). We can get the marginal probability mass function of $(i_1, i_2, \ldots, i_L)$ from equation (3.26a) by summing over all possible values of $i_{L+1}, i_{L+2}, \ldots, i_K$ and using the binomial formula.

$$\text{Prob}\ (i_1, i_2, \ldots, i_L \mid M_T) = \tag{3.29a}$$

$$(M_T)! \left[\frac{(1/NO)^{i_1}}{(i_1)!} \quad \frac{(1/NO)^{i_2}}{(i_2)!} \cdots \frac{(1 - L/NO)^{i_{L+1}}}{(i_{L+1})!} \right]$$

$$\text{where} \quad i_{L+1} = \left[M_T - \sum_{j=1}^{L} i_j \right] \tag{3.29b}$$

Let $T(z_1, z_2, \ldots, z_L \mid M_T)$ be the joint probability generating function of this multinomial distribution conditioned on the total number of interfering copies being equal to M_T.

$$T(z_1, z_2, \ldots, z_L \mid M_T) = \left[\sum_{j=1}^{L} \frac{z_j}{NO} + \left(1 - \frac{L}{NO} \right) \right]^{M_T} \tag{3.30}$$

We can now combine equations (3.28) and (3.29a) to obtain

$$\text{Prob (first L or more} \mid M_T, K) = \tag{3.31}$$

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_L} M_T! \left[\prod_{j=1}^{L} \frac{(1/NO)^{i_j}}{(i_j+1)!}\right] \frac{(1-L/NO)^{i_{L+1}}}{(i_{L+1})!} \quad ;L \le K$$

From the definition of the probability generating function

$$T(z_1, z_2, \ldots, z_L \mid M_T) \triangleq$$

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_L} M_T! \prod_{j=1}^{L} Z_j^{i_j} \frac{(1/NO)^{i_j}}{(i_j)!} \frac{(1-L/NO)^{i_{L+1}}}{(i_{L+1})!} \tag{3.32}$$

We can integrate equation (3.32) over $z_1, z_2, \ldots, z_L$ to obtain

$$\int_0^1 dz_1 \int_0^1 dz_2 \cdots \int_0^1 dz_L \; T(z_1, z_2, \ldots, z_L \mid M_T) = \tag{3.33}$$

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_L} (M_T)! \left[\prod_{j=1}^{L} \frac{(1/NO)^{i_j}}{(i_j+1)!}\right] \frac{(1-L/NO)^{i_{L+1}}}{(i_{L+1})!}$$

Thus from equations (3.30), (3.31) and (3.33) we get

$$\text{Prob (first L or more} \mid M_T, K) = \tag{3.34}$$

$$\int_0^1 dz_1 \int_0^1 dz_2 \cdots \int_0^1 dz_L \left[\sum_{j=1}^{L} \frac{z_j}{NO} + \left(1 - \frac{L}{NO}\right)\right]^{M_T}$$

The detailed calculations of the integration are presented in Appendix B. The result is shown in equation (3.35).

$$\text{Prob (first L or more} \mid M_T, K) = \tag{3.35}$$

$$\prod_{j=1}^{L} \frac{(NO)^L}{(M_T + j)} \sum_{i=0}^{L} \binom{L}{i} (-1)^i \left[1 - i/NO \right]^{M_T + L} \quad ;L \le K$$

Up to now we have been dealing with probabilities that are conditioned on the event that a total of M_T copies were generated by the NI-1 conflicting input ports. We can average over that event to get

$$\text{Prob (first L or more} \mid K) = \tag{3.36}$$

$$\sum_{j=0}^{NO(NI-1)} \text{Prob } (M_T = j) \text{ Prob (first L or more} \mid M_T, K)$$

The individual probabilities of M_T can be compute from the probability generating function $M_T(z)$ given in equation (3.24), or by convolving the probability mass function of M with itself NI-1 times.

$$M_T(z) \triangleq \sum_{j=0}^{NO(NI-1)} \text{Prob } (M_T = j)\ z^j \tag{3.37}$$

$$\text{Prob } (M_T = j) \triangleq \left\{ \text{Prob } (M = i) \right\}^{(NI-1)*} \quad \text{where } \sum_i = j \tag{3.38}$$

At this point we have developed an expression for the probability of getting L out of K copies through the first

L output ports. Again since the traffic is symmetric, this probability is the same for any other set of L output ports. The remaining steps of this analysis are identical to the ones presented in the Bernouilli trials traffic distribution model. That is we first find Prob(L | K), the probability of getting any set of L of K copies through. We then derive Q(n | K), the probability that n slots are required to transmit the K copies. Finally we average over K and find the average service time and average delay. The reader is invited to refer to equation (3.11a) through (3.20) for more details.

3.5 Multinomial traffic distribution with residual service time

As mentioned previously, a packet reaching the head of an input port queue generates a random number of copies which are distributed to the output ports. The service time of this packet begins when the copies are generated and terminates when each of these copies has gained access to an output port.

In the previous two models it was assumed that the service initiations at the input ports were synchronized; the interfering traffic encountered by one input port was assumed to be equal to the number of copies generated by the other input ports. In others words, a packet arriving at the head of an input queue, always saw the other conflicting packets at the beginning of their service time.

This, more refined, third analysis takes a more realistic approach toward the evaluation of the interfering traffic, by allowing packets at different input ports to initiate their service times independently. To support this approach, the model needs to take into account the residual service time of all the interfering packets from the NI-1 conflicting input ports. Since the service duration is proportional to the number of copies generated, we consider the residual number of copies. Discussions on residual

waiting time can be found in [13] and [14].

We can illustrate this by considering the trivial case of only one input port B in conflict with input port A as shown in figure 3.3. Let τ_i and τ_j denote the beginning and the end of service of a packet at input port B. The service time of that packet is then equal to $(\tau_j - \tau_i)$. Now consider a random point in time, say t, when a packet from port A reaches the head of the input queue and begins service. This packet from port A can encounter an empty queue at port B or an interfering packet at port B.

In the case where input queue B is empty, no conflict occurs for this slot. Since ρ is the probability of an input port being occupied, this event happens with probability $(1-\rho)$. For the case where input port B is busy, the packet from port A encounters a number of interfering copies from port B. Clearly this number of interfering copies seen by the packet at input port A, is not the total number of copies generated at port B, but the remaining number of copies that have not gained access to an output port at time t. We can also show that the service time of the interfering packet, that input port A has selected, does not have the same distribution as a typical service time; a longer service time is more likely to be chosen. This approach for evaluating the interfering traffic can be

applied to the previous model by modifying the computation of the total number of interfering copies M_T. This third analysis is mainly concerned with this evaluation of the interfering traffic.

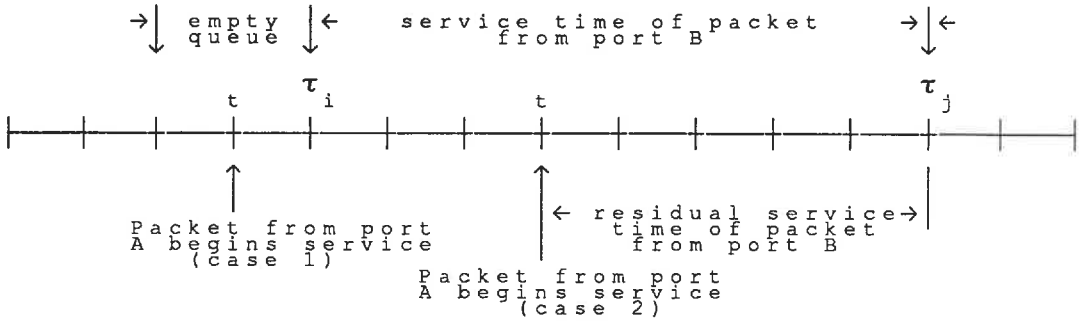


Figure 3.3 Service time and residual time

3.5.1 Mathematical analysis

We again consider one particular input port and NI-1 interfering input ports. Let M denote the the number of copies generated by one interfering input port. As mentioned previously, this random variable follows a modified binomial distribution with parameters NO and P. Since, to a typical service time duration, there corresponds a modified binomial number of copies, we want to find the probability mass function of the number of copies corresponding to the service time duration encountered by our input port. We define the random variables S and R representing respectively the number of copies of this selected interval and the residual number of

copies in conflict at time t. Renewal theory shows that the probability mass function of a selected interval is proportional to the length of that interval and its frequency of occurrence. Again we first condition on the event that the interfering input port is active.

$$\text{Prob}(S = i|\text{active}) = \frac{i\ \text{Prob}(M = i|\text{active})}{\overline{M}} \qquad (3.39)$$
$$i = 1,\ldots,NO$$

where $\overline{M}$ is the mean of the modified binomial distribution. Now we would like to find the probability mass function of the residual number of copies conditioned on S, the number of copies of the selected service time. Since R is uniformly distributed over the selected interval, we have

$$\text{Prob}\ (R = j \mid S = i) = \frac{1}{i} \quad \text{for} \quad j = 1,2,\ldots,i \qquad (3.40)$$
$$i \leq NO$$

$$\text{Prob}(S = i, R = j|\text{active}) = \qquad (3.41)$$

$$\text{Prob}(R = j|S = i)\ \text{Prob}\ (S = i|\text{active})$$

$$= \frac{\text{Prob}(M = i|\text{active})}{\overline{M}} \qquad ;i \leq NO \quad ;j \leq i$$

The marginal probability mass function of R can be compute by averaging over all values of i.

$$\text{Prob}(R = j|\text{active}) = \frac{1}{\bar{M}} \sum_{i=j}^{NO} \text{Prob}\ (M = i|\text{active}) \quad ;i \le NO \quad ;j \le i \tag{3.42}$$

The probability generating function R(z|active) can also be evaluated.

$$R(z|\text{active}) \triangleq \sum_{j=1}^{\infty} z^j\ \text{Prob}(R = j|\text{active}) \tag{3.43}$$

$$= \frac{1}{\bar{M}} \sum_{j=1}^{\infty} z^j \sum_{i=j}^{NO} \text{Prob}(M = i|\text{active})$$

The reader can refer to Appendix C for the development of the probability generating function (P.G.F.) R(z). The resulting expression is

$$R(z|\text{active}) = \frac{z}{NO\ P\ (1-z)} \left[\ 1 - [\ Pz + (1-P)]^{NO}\ \right] \tag{3.44}$$

We can average over the event of the interfering input port being active and get

$$R(z) = \rho\left[\ \frac{z}{NO\ P\ (1-z)}\ (\ 1 - [\ Pz + (1-P)]^{NO}\ )\right] + (1-\rho) \tag{3.45}$$

Equation (3.45) gives an expression for the P.G.F. of the residual number of conflicting copies coming from one

interfering input port. These copies were generated independently at each input port at the beginning of their service time, however in the process of contending for output ports, a certain coupling was created. Nevertheless we assume independence between input ports to keep the complexity manageable. The P.G.F. for the total number of conflicting copies is then given by equation (3.46).

$$R_T(z) = \left[\rho\left[\frac{z}{NO\ P\ (1-z)}\left(1-\left[Pz + (1-P)\right]^{NO}\right)\right] + (1-\rho)\right]^{NI-1} \qquad (3.46)$$

We can now replace $Prob(M_T = j)$ by $Prob(R_T = j)$ in equation (3.36) to obtain the probability of getting a particular set of L or more copies through the first L output ports.

$$Prob\ (\text{first L or more} \mid K) = \sum_{j=0}^{NO(NI-1)} Prob(R_T = j)\ Prob\ (\text{first L or more} \mid R_T, K) \qquad (3.47)$$

Because of its relative complexity, it appears difficult to isolate the individual probabilities from equation (3.46). Instead, we can work in the time domain to obtain the desired probabilities. Since the total number of residual copies is the sum of residual copies of the individual input ports, the probability mass function of R_T is equal to the (NI-1)-fold convolution of the probability mass function of R with itself.

$$\text{Prob}\ (R_T = k) = \left\{ \text{Prob}\ (R = j) \right\}^{(NI-1)^*} \quad \text{where} \sum_j = k \qquad (3.48)$$

Equation (3.48) was evaluated with a simple program routine that performs the convolution. The individual probabilities were then used in equation (3.47) to take out the condition on R_T. One other alternative that could have been used for evaluating the probabilities of R_T, is to approximate its probability mass function by a Gaussian density function using the Central Limit theorem. For the Gaussian distribution we only need the average and variance of R which can be computed from the probability generating function R(z). The results are given by equations (3.49) and (3.50).

$$\bar{R} = \left. \frac{\partial\, R(z)}{\partial z} \right|_{z=1} = \frac{NOP + 2 - P}{2} \qquad (3.49)$$

$$\text{Var}\ R = \left[\left. \frac{\partial^2\, R(z)}{\partial z^2} \right|_{z=1} + \left. \frac{\partial\, R(z)}{\partial z} \right|_{z=1} - \left[\left. \frac{\partial\, R(z)}{\partial z} \right|_{z=1} \right]^2 \right]$$

$$= \frac{(NO-5)\ (NO-1)\ P^2}{12} + \frac{(NO-1)\ P}{2} \qquad (3.50)$$

Since we assume that R_T is the sum of independent identically distributed random variables with mean and variance given by equations (3.49) and (3.50), the probability mass function of R_T is given by

$$\text{Prob}\ (R_T = k) = \text{Prob}\left[\ k-\alpha < R_T < k+\alpha\ \right] \qquad (3.51)$$

$$= \text{Prob}\left[\frac{(k-\alpha)-(NI-1)\bar{R}}{\sqrt{(NI-1)\text{Var}\ R}} < \frac{R_T-(NI-1)\bar{R}}{\sqrt{(NI-1)\text{Var}\ R}} < \frac{(k+\alpha)-(NI-1)\bar{R}}{\sqrt{(NI-1)\text{Var}\ R}}\right]$$

If we define

$$u = \frac{(k-\alpha)-(NI-1)\bar{R}}{\sqrt{(NI-1)\text{Var}\ R}} \quad \text{and} \quad v = \frac{(k+\alpha)-(NI-1)\bar{R}}{\sqrt{(NI-1)\text{Var}\ R}} \qquad (3.52)$$

we then have

$$\text{Prob}\ (R_T = k) = \left[\ \Phi\ (v)\ -\ \Phi\ (u)\ \right] \qquad (3.53)$$

where $\Phi(a)$ is the standard normal distribution given by

$$\Phi(a) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{a} e^{-x^2/2}\ \partial x \qquad (3.54)$$

Since R_T is a discrete random variable taking integer values, we choose $\alpha = .5$ to get a good approximation from the normal distribution. Having evaluated Prob(first L or more | K), we can now proceed with the remaining steps ,as described in section 3.3.1, to find the average service time and average delay.

3.6 Multinomial traffic distribution with residual service time and multiple output ports

This analysis considers the case where output ports are replaced by output trunks containing multiple ports. This type of switch arrangement is necessary for multicast applications because of the increase of traffic intensity at output ports. By having multiple lines at each output trunk, the congestion is reduced and the switch throughput is increased. For this case we propose an approximation that provides a lower bound on packet delay. This bound is expected to be tight for heavy loading and for relatively small values of S when compared to the size of the switch.

3.6.1 Mathematical analysis

In the foregoing it was assumed that each output trunk contained a single output port; that is $NO = NT_xS$ with $S = 1$. This constraint limits the performance of the switch especially if one considers that more traffic is created by the multicast switch. We now consider the case where output ports are bundled into trunks consisting of $S > 1$ ports. As in the previous cases, packets at the input ports generate multiple copies. We now assume that these copies are generated for output trunks instead of output ports.

The joint probability generating function of the number of interfering copies is still given by equation

(3.30) where NO and M_T are replaced by NT and R_T. The difference lies in the adjudication process at output trunks. If there are i_j interfering copies at an output trunk having S output ports, the probability that our copy gets through is

$$\text{Prob (success)} = \left\{ \begin{array}{ll} 1 & \text{if } i_j \leq (S - 1) \\ S/(i_j + 1) & \text{if } i_j > (S - 1) \end{array} \right\} \quad (3.55)$$

This can be written in terms of unit step functions

$$\text{Prob (success)} = [\, 1 - U(i_j - (S - 1))\,] + S/(i_j + 1)\,[\, U(i_j - (S - 1))\,] \quad (3.56)$$

$$\text{where } U(j) = \left\{ \begin{array}{ll} 0 & \text{if } j < 0 \\ 1 & \text{if } j \geq 0 \end{array} \right\} \quad (3.57)$$

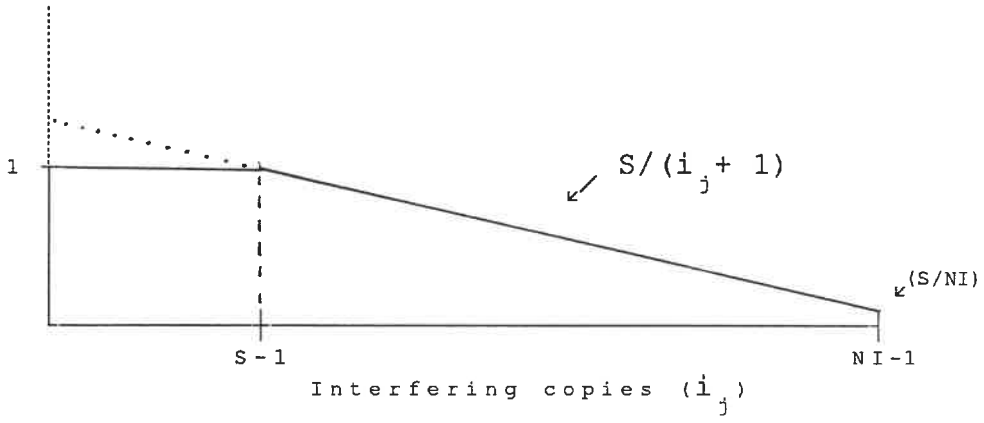


Figure 3.4 Probability of winning contention

The probability of winning contention at one output trunk is plotted in figure 3.4. The shape of this probability function makes it difficult to develop an exact expression for Prob(L or more |K) by using the method used in the previous sections. By looking at the probability function of figure 3.4, we observe that for values of i_j greater than (S-1) the probability of success is given by $S/(i_j + 1)$. Since the value of i_j is determined by the intensity of traffic at output trunk j, we can assume that under medium to heavy loading the contention resolution process would operate in the region for which $(i_j > S-1)$. This assumption is further reinforced if the number of output ports at each trunk is small compared to the the number of input ports.

If we now assume that Prob(success) = $S/(i_j+1)$ for all values of i_j, we obtain an upper bound on the actual probability function. The approximation is illustrated by the dotted line in figure 3.4. This assumption would give erroneous results for low traffic intensity because we have Prob(success) > 1 if $i_j < (S-1)$. We can now proceed and replaced equation (3.55) by equation (3.58).

$$\text{Prob (success)} \leq S/(i_j+1) \quad ;i_j = 0,1,\ldots,NI-1 \tag{3.58}$$

The probability that L or more copies get through in a single slot can be bounded by substituting equation (3.58) for $1/(i_j+1)$ in equation (3.28) and replacing M_T by R_T.

$$\text{Prob (first L or more} \mid R_T, K) \leq \tag{3.59}$$

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_L}\prod_{j=1}^{L} S/(i_j+1)\ \text{Prob}\ (i_1,i_2,\ldots,i_L \mid R_T)\ ;L \leq K$$

From its definition, the joint probability generating function for the number of interfering copies at each of the L output trunks is given by

$$T(z_1,z_2,\ldots z_L \mid R_T) \triangleq \tag{3.60}$$

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_L}\prod_{j=1}^{L} z_j^{i_j}\ \text{Prob}\ (i_1,i_2,\ldots,i_L \mid R_T)$$

where the joint probability generating function of the multinomial distribution is given by

$$T(z_1, z_2, \ldots, z_L \mid R_T) = \left[\sum_{j=1}^{L} \frac{z_j}{NT} + \left(1 - \frac{L}{NT} \right) \right]^{R_T} \quad (3.61)$$

Thus from equations (3.59), (3.60), (3.61) we obtain

$$\text{Prob (first L or more} \mid R_T, K) \leq \quad (3.62)$$

$$S^L \int_0^1 dz_1 \int_0^1 dz_2 \ldots \int_0^1 dz_L \left[\sum_{j=1}^{L} \frac{z_j}{NT} + \left(1 - \frac{L}{NT} \right) \right]^{R_T}$$

The result of the integration is given in equation (3.63)

$$\text{Prob (first L or more} \mid R_T, K) \leq \quad (3.63)$$

$$\prod_{j=1}^{L} \frac{S^L \, NT^L}{(R_T + j)} \sum_{i=0}^{L} \binom{L}{i} (-1)^i \left[1 - i/NT \right]^{R_T + L} ; L \leq K$$

We have now found an upper bound for the probability of winning contention at least at the first L output trunks. The analysis goes through by first taking out the condition on R_T as in section 3.5.1 and then by computing Prob (L|k), Q (n|k), Q(n), $\overline{S}$ and $\overline{D}$ as described in section 3.3.1. There is a problem however with proceeding from this point. The expression in equation (3.13) is a summation with alternating signs; consequently, the upper bound is

still valid only if the approximation is very tight. Caution must then be taken when looking at the results obtained from this analysis.

3.6 Numerical computation

The expressions that have been derived were evaluated on a VAX 8650 at Le Centre de Recherche Informatique de Montréal. The programs were implemented in the C language. One limitation related to the C language is the maximum precision and range of floating-point variables; that is, C supports only single and double precision variables. We used G type double precision which gives a range of $.56x10^{-308}$ to $.899x10^{308}$ with 15 decimal digits of precision. This permits us to perform the computation for a maximum switch size of 16x16.

Performance evaluation of larger switches can be obtained by translating the program from C to Pascal which supports quadruple-precision variables. These H type quadruple-precision variables have a range of $.84x10^{-4932}$ to $.59x10^{4932}$ with 33 decimal digits. This extended range and precision should permit us to evaluate the performance of switch of size up to 64x64. The C programs are included in Appendix D.

4. Simulation of a multicast packet switch

To verify the analytical model, an NI input ports and NO output ports multicast switch was simulated. The output ports can be bundled into NT trunks of S ports where NO = NTxS. The level of simulation adopted, represents the entire process of packets going through the switch. To achieve this, a self-driven simulation was developed where packet arrivals, copies distribution and conflicts resolution at output ports (trunks) are represented by probability distributions.

As mentioned in the analysis section, time is segmented into fixed size slots, and a slot is the minimum duration of any event. Because of this, synchronous timing was adopted and the clock increment was set to 1 slot duration. For verification of the analysis, the simulation is mainly concerned with estimating the mean service time and mean delay of packets going through the switch.

4.1 Simulation model

Two variations of the simulation model were developed. In the first case, a random number of copies is generated for each new head-of-queue packet and these copies are distributed to output ports. The method used for this distribution of traffic is explained in section 4.1.1.2.

The second variation modifies the distribution of copies to output ports for simulating a unicast switch. In this case, a packet at the head of an input queue generates 1 copy by choosing 1 output port with probability 1/NO. This variation is implemented for verification purposes; results from this model can be compared with published results on the unicast packet switch [15].

The simulation model is shown in figure 4.1. First, packet arrivals at each input port are generated according to a Poisson distribution with average rate λ. Then for each new head-of-queue packet, copies are distributed to output ports (trunks). Multiples copies or a single copy can be sent to output ports depending on the two traffic distributions described previously. The third step takes care of conflict resolution at each output port (trunk). If there are M packets in conflict, one is chosen with probability 1/M. In the case of multiple output ports per trunk, S copies are chosen with probability S/M if $S < M$ or M are chosen with probability 1 if $S \geq M$.

Following this, the clock is advanced by one slot duration. At this point each head-of-queue packet is checked for service completion and the statistical data is updated accordingly. These 5 steps are repeated for the desired duration of the simulation run. To achieve better

accuracy multiple runs are executed by varying the original seeds. At the end, the performance parameters are computed from the accumulated statistical data.

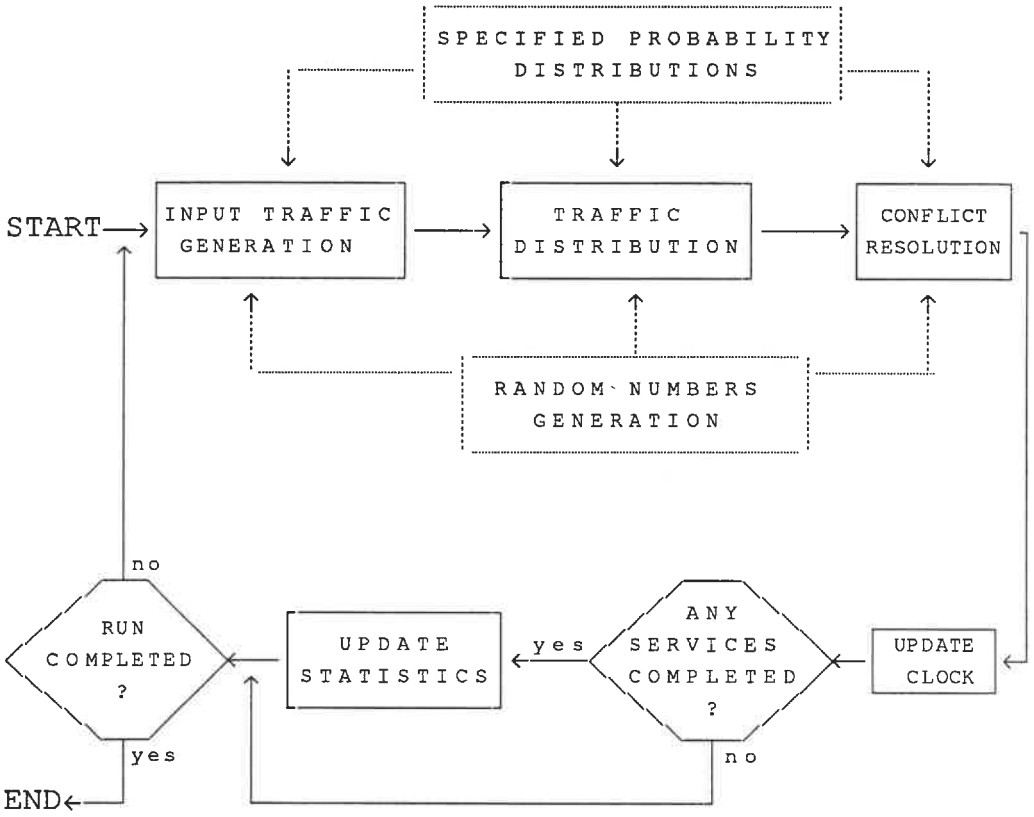


Figure 4.1 Simulation model

4.1.1 Random variables generation

With self-driven simulation, probability distributions on random variables are needed to generate the required stimulus to the system. The different probability distributions can be constructed from a sequence of random

numbers. Our simulation model relies on a built-in system routine to get the random numbers. This routine use the mixed congruential generator with the following recurrence relation.

$$Seed_i = (69069 * Seed_{i-1}) + 1 \quad \text{modulo } 2^{32} \qquad (4.1)$$

The seed is automatically updated for the next call. The routine returns a Uniformly distributed random number by converting the high order 24 bits of the seed to a floating point number between 0 and 1. Different random sequences can be obtain by initializing the seed to a different value on separate runs.

4.1.1.1 Packet arrivals

The switch model assume that packets arrivals are generated independently at each input port according to a Poisson distribution with mean λ. Since Poisson processes have exponentially distributed interarrival times, the method used to generate Poisson arrivals is based on results obtained from exponential random variables. This method is developed in [16] and contains the following steps:

Generate independent uniformly distributed (0,1) random variables $U_1, U_2, \ldots$ stopping at

$$N + 1 = \min \left\{ n : \prod_{i=1}^{n} U_i < e^{-\lambda} \right\} \qquad (4.2)$$

The random variable N has a Poisson distribution with mean λ, which can be seen by noting that

$$N = \max \left\{ n : \sum_{i=1}^{n} -\log U_i < \lambda \right\} \qquad (4.3)$$

From the inverse transformation method we have that $-\log U_i$ is exponential with mean 1. If $-\log U_i$, $i = 1,2,\ldots,n$, are the interarrival times of a Poisson process with rate 1, then N is equal to the number of arrivals by time λ. Then N is Poisson with mean arrival rate λ.

4.1.1.2 Copies generation and distribution

We mentioned previously that each packet reaching the head of an input port queue generates a random number of copies. The generation and distribution of copies is done by performing independent Bernouilli trials with parameter P for each output port (trunk). If no copy is generated after NO (NT) trials, the copy generation process is restarted. This process is repeated until at least one copy is generated. We then obtain a number of copies that follows a modified binomial distribution. The method used is the following:

Generate a uniformly distributed random number U_i and define the Bernouilli random variable B_i associated with output port (trunk) i. We then have

$$B_i = \left\{ \begin{array}{ll} \text{success} & \text{if } U_i < P \\ \text{failure} & \text{otherwise} \end{array} \right\} \qquad (4.4)$$

For the unicast switch case where 1 output port is chosen from the NO output ports with probability 1/NO, we use the following method:

Split the [0,1] interval into NO sub-intervals I_j of equal size. Generate a uniform random number U between 0 and 1. The chosen output port j is the one for which $U \in I_j$.

4.1.1.3 Conflicts resolution at output ports (trunks)

To resolve conflicts due to multiple copies trying to get access to the same output port (trunk) the following method was adopted.

Get the number of conflicting copies and split the [0,1] interval accordingly. In the case of 1 output port per trunk, generate 1 uniform random number U and find the matching interval . For multiple output ports

per trunk, we repeat this step S times if $S < M$ where M is the number of conflicting copies. Clearly if $S \geq M$ all the conflicting copies win access to an output port.

4.1.2 Data structures

In order to keep track of all events during the simulation and to facilitate the manipulation data, we use well defined data structures. These data structures take the form of tables and queues. Our simulation model contains 3 main data structures: input queues, output queues, and tables for recording performance statistics.

4.1.2.1 Input queues

Each input queue is a circular FIFO buffer where an entry corresponds to a packet waiting to be served. Each entry is subdivided into 5 fields containing the necessary information regarding the packet stay's in the switch. The queue has a finite length which determines the maximum number of waiting packets. A mechanism for detecting overflow of the input port queue is implemented. The packets arrivals and departures from the queue are controlled by a head pointer and a tail pointer.

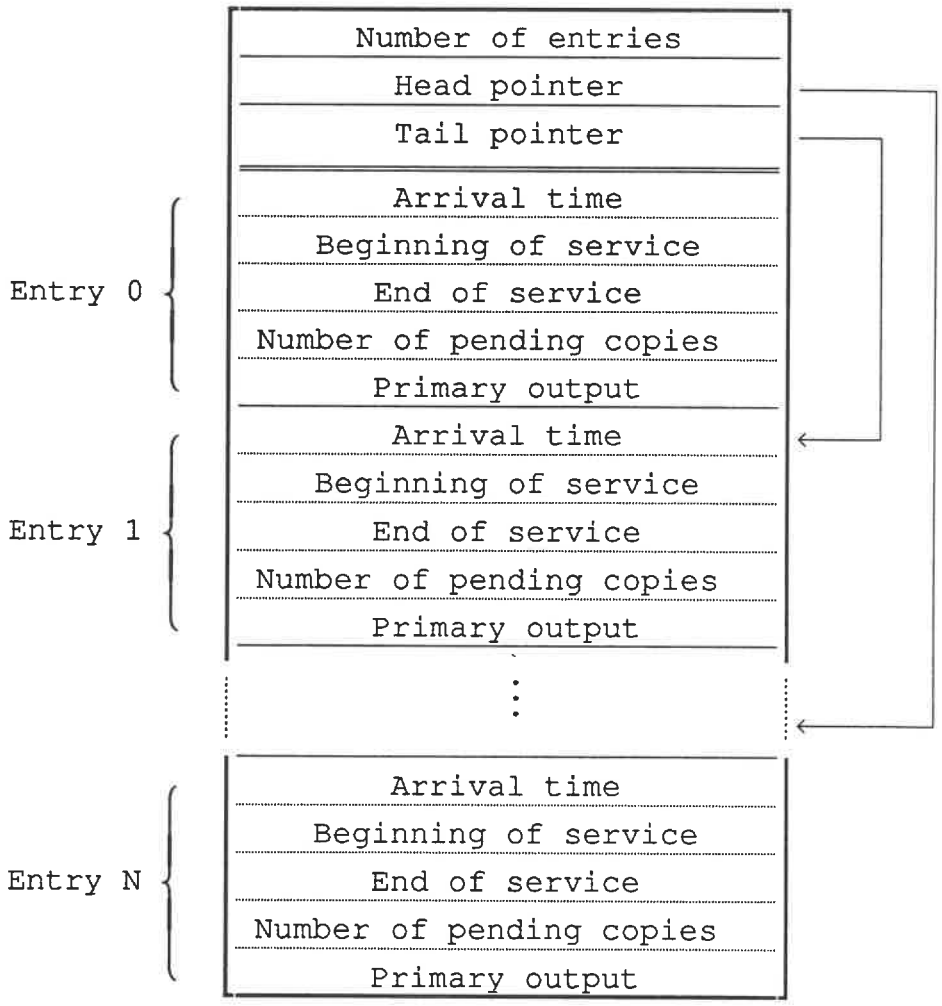


Figure 4.2 Input queue format

4.1.2.2 Output queues

Each output port (trunk) has an output queue that contains for each input port a field indicating a conflicting copy. The output queue also contains information about the number of copies in conflict and the queue occupancy. We should mention that the output queue is just a programming device used to keep track of the

interfering copies and that packets waiting for service completion are actually queued at the input port.

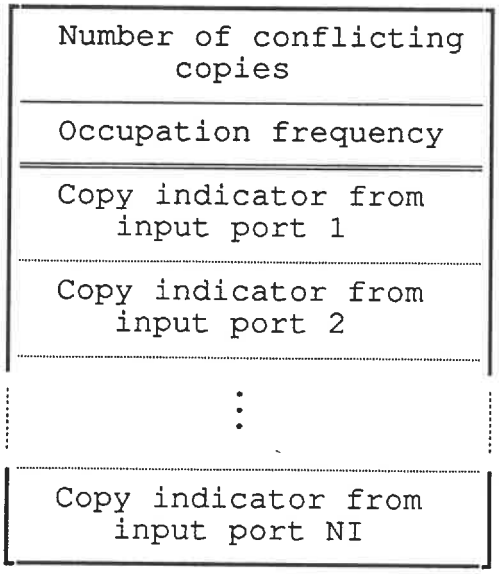


Figure 4.3 Output queue format

4.1.2.3 Performance statistics

The performance statistics data structure includes all the necessary measurements for performance evaluation of the multicast switch. It contains the statistical data from each individual run which are used to compute the total average results. The output traffic distribution statistics allow the evaluation of the total packet distribution at output ports (trunks). This is used for comparing the traffic distribution observed during simulation with the ones assumed in the analysis.

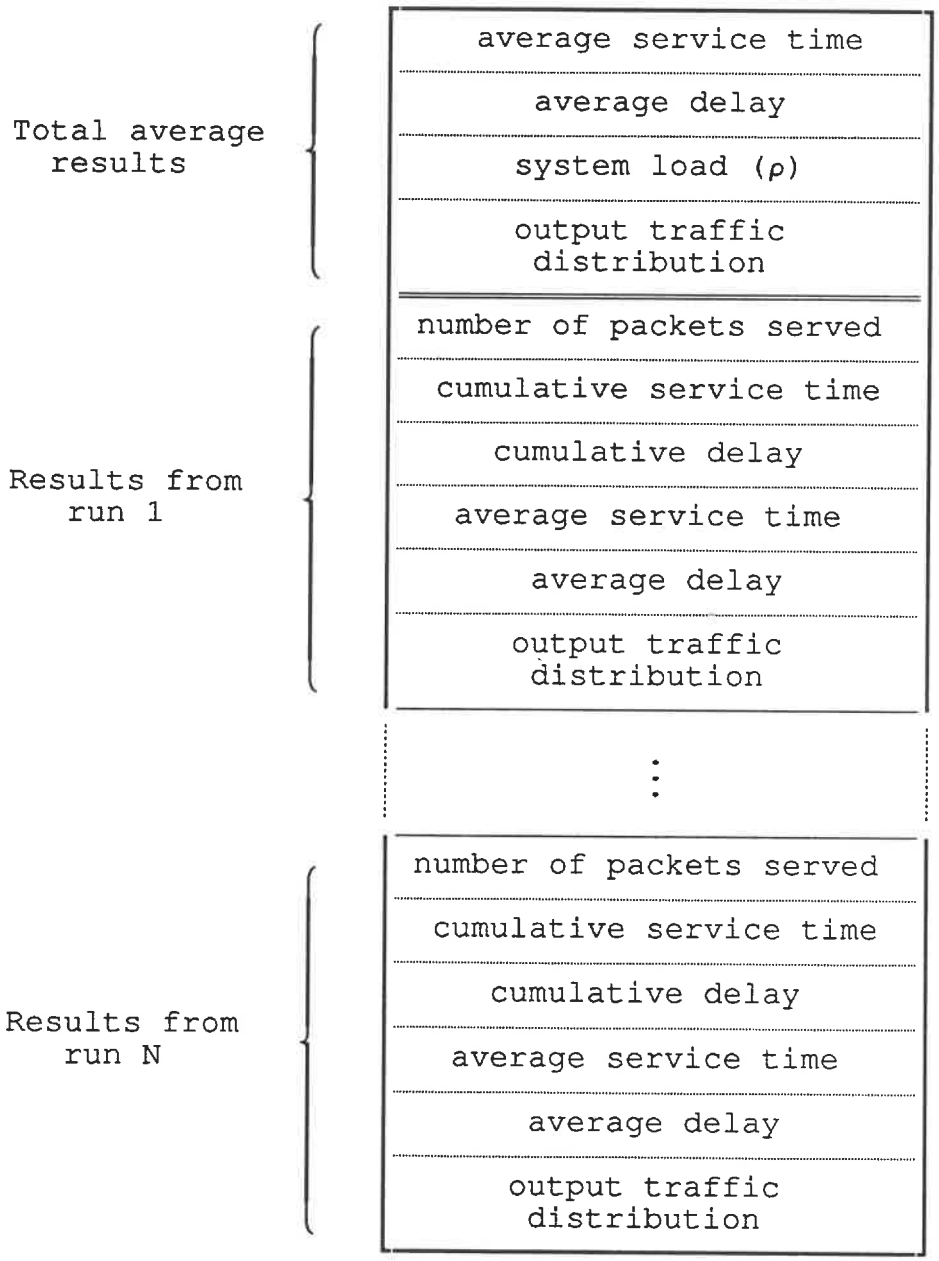


Figure 4.4 Performance statitics data structure

4.2 Validation and verification

We now turn our attention to verifying the correctness of the simulation program and the simulation model. In the first step, we do a comparative "walkthrough" of the simulation program and the model description. This permits the detection of obvious differences between the model and the program.

In the second step, the validation and verification is done by collecting comparative performance measures of existing systems. Since there is no available performance data of a similar system, we have to adapt the simulation program to a particular architecture or run it under special conditions for which an exact analytical solution is known. We should mentioned that this adaptation to a particular system does not guarantee 100 percent correctness of the program under the desired conditions.

There are two special cases for which the simulation results can be compared to analytical results. The first one is a 1x1 switch which simplifies to an M/D/1 queue. The second case is a NxN unicast packet switches for which asymptotic saturation throughput is known.

4.2.1 A 1x1 switch (M/D/1 Queue)

When the number of input and output ports are equal to

1, the service time becomes deterministic and the switch behaves like an M/D/1 queue with average delay given by equation 4.5.

$$\overline{D} = \overline{S} + \frac{\rho\ \overline{S}}{2\ (1-\rho)} \qquad (4.5)$$

In our case, the service time is always equal to 1 slot so the average delay becomes

$$\overline{D} = 1 + \frac{\lambda}{2\ (1-\lambda)} \qquad (4.6)$$

To verify this equation, we take the second variation of our model where 1 copy is generated with probability 1. The statistical method of section 4.3 is used to assure accuracy.

4.2.2 A NxN unicast packet switch

We can also verify the program by simulating a NxN unicast switch for small values of N. This type of switch has been studied extensively in the context of processors-memories interconnection for multiprocessor systems. For saturated input queues and $N \longrightarrow \infty$, the maximum throughput is equal to $(2 - \sqrt{2}) = 0.5858$. For small values of N, results have been obtain from Markov chain analysis. These results are included in table 4.1 and presented in more detail in [15].

N	Saturation Throughput
1	1.0000
2	0.7500
4	0.6553
8	0.6184
∞	0.5858

Table 4.1 Saturated Throughput for point-to point switches with input queuing

4.3 Simulation results analysis

This section is concerned with estimating and controlling the simulation results accuracy. In depth treatment of simulation results analysis can be found in [17]. Since we are evaluating the expected values of random variables, i.e, service time and delay, the accuracy is a measure of how close the sample mean is to the actual mean. To best evaluate these random variables, our simulation uses the replication method which consist of making K independent runs of M samples using a different random number stream for each run. A confidence interval is computed to determine the values of K and M satisfying the desired level of accuracy.

One other aspect related to results analysis is the evaluation of the transient period. The transient period is the time elapsing between the simulation starting point to the point beyond which the system is considered to be in equilibrium. To best estimate the service time and delay for steady-state operation, we have to ignore the observations during this transient period. Our approach is to make the run lengths long enough to eliminate all warmup effects. However, a deletion amount of 5 percent was chosen to be on the safe side.

4.3.1 Confidence interval estimation

As mentioned, the replication method implies that we make K runs with each run generating M sample values (M packets) of the variables to be measured. These variables are the service time and the waiting of a packet going through the switch. The waiting time is the time that a packet spends in an input port queue, waiting for service. The evaluation of delay is done by adding waiting time and service time.

Let $\bar{S}_1, \bar{S}_2, \ldots, \bar{S}_K$ and $\bar{W}_1, \bar{W}_2, \ldots, \bar{W}_K$ be the mean service times and the mean waiting times for each of the k runs. We then have the following relations:

$$\bar{S}_j = \sum_{i=1}^{M} \frac{S_i}{M} \qquad \text{where } S_i \text{ is the service time of packet (i) in run (j)} \qquad (4.7)$$

$$\bar{W}_j = \sum_{i=1}^{M} \frac{W_i}{M} \qquad \text{where } W_i \text{ is the waiting time of packet (i) in run (j)} \qquad (4.8)$$

The sample means of service time ($\bar{S}$), waiting time ($\bar{W}$) and delay ($\bar{D}$) are given by

$$\bar{S} = \sum_{j=1}^{K} \frac{\bar{S}_j}{K} \qquad (4.9)$$

$$\bar{W} = \sum_{j=1}^{K} \frac{\bar{W}_j}{K} \qquad (4.10)$$

$$\bar{D} = \bar{W} + \bar{S} \tag{4.11}$$

We want to evaluate the accuracy of our estimate of $\bar{S}$ and $\bar{W}$. Since we don't know the actual variances of $\bar{S}_j$ and $\bar{W}_j$, we have to approximate them with the sample variances. These sample variances are given by equations (4.12) and (4.13).

$$\tilde{Var}\ (\bar{S}_j) = \sum_{j=1}^{K} \frac{(\bar{S}_j - \bar{S})^2}{(K-1)} \tag{4.12}$$

$$\tilde{Var}\ (\bar{W}_j) = \sum_{j=1}^{K} \frac{(\bar{W}_j - \bar{W})^2}{(K-1)} \tag{4.13}$$

For a confidence coefficient of $(1-\alpha)$, we can construct confidence intervals for the mean service time and the mean waiting time. The confidence intervals are given by

$$\bar{S} \pm H_s \qquad \text{where} \quad H_s = (t_{\alpha/2;K-1}) \frac{\sqrt{\tilde{Var}\ (\bar{S}_j)}}{\sqrt{K}} \tag{4.14}$$

$$\bar{W} \pm H_w \qquad \text{where} \quad H_w = (t_{\alpha/2;K-1}) \frac{\sqrt{\tilde{Var}\ (\bar{W}_j)}}{\sqrt{K}} \tag{4.15}$$

The expression $t_{\alpha/2;K-1}$ is the upper $\alpha/2$ quantile of the **t** distribution with K-1 degrees of freedom. We can choose the confidence coefficient $(1-\alpha) = .95$. We are then 95 percent confident that the true mean service time and waiting time are within these intervals.

We now have to determine the values of M and K and the desired level of accuracy. As a rule of thumb, it is best to keep K relatively small, (< 10) and M relatively large (≅ 5000). An accuracy of 10 percent for a 95 percent confidence level also seems reasonable. Since our simulation model evaluates the run length in terms of time slots instead of samples (packets), we need a relation between the number of samples (M) and the number of slots in a run. For an NI input ports switch with average Poisson arrival rate of λ we have the following relation.

$$M \cong NI * \lambda * \text{Run length (slots)} \qquad (4.16)$$

The exact values of M and K are determined by experiment, starting with K = 8 and M = 2000 and increasing M until the following equations are satisfied.

$$H_s < .1\overline{S} \qquad \text{and} \qquad H_w < .1\overline{W} \qquad (4.17)$$

4.4 Implementation

The simulation program was implemented in the C language on a VAX 8650 running VMS version 4.7. The facilities were provided by Le Centre de Recherche Informatique de Montréal in collaboration with Concordia University. The program can simulate switches of any size and the only limitation is related to the amount of computation time necessary. The listings of the simulation programs are included in the Appendix E.

5. Results

This section presents the results computed from the analysis and the simulation programs. These results were obtained for switches of different configurations and sizes. In all cases the performance results are presented by plotting the average packet delay ($\overline{D}$) as a function of the packet arrival rate λ (Traffic intensity). Multiple curves are obtained for different values of P (multicast probability). The value of P, together with the traffic intensity λ, determine the loading of the multicast switch. Simulation and analysis results are compared in section 5.3.

5.1 Analytical results

Results for three analytical models are compared. These models were developed in section 3.3,3.4, and 3.5. We can review these three models.

Bernouilli model (B_model): Each packet chooses its output ports from independent Bernouilli trials with probability P. The number of copies generated at each input port is then binomially distributed with parameters NO and P.

Multinomial model (MM_model): Each packet generates a modified binomial number of copies with parameters NO

and P. The copies from each input port are pooled together and distributed to output ports according to a multinomial distribution.

Multinomial-Residual model (MR_model): Each packet generates a modified binomial number of copies with parameters NO and P. The copies from each input port are pooled together and distributed to output ports according to a multinomial distribution. The evaluation of the interfering traffic from each input port is done by considering the residual number of copies of the modified binomial distribution.

Results for these three models, (B, MM, MR), are plotted in figures 5.1, 5.2 and 5.3, respectively, for switches of sizes 4x4, 8x8 and 16x16. By comparing results from the three models, we can make the following remarks:

1) For small values of P, the B_model gives inaccurate results because the probability of a packet generating zero copy is not negligible. This was expected and mentioned in section 3.3.

2) For larger values of P, the B_model and the MM_model give comparable results especially for the 16x16 switch. This tends to show that for relatively

large switches, the multicast traffic distribution is not a critical factor as long as the average number of copies generated by each packet is the same.

3) The MR_model gives optimistic results when compared to the other two models. This is evident because the evaluation of the number of interfering copies was obtained by considering the residual number of copies instead of the total number of copies generated. We will compare this approach with the simulation in section 5.3.

5.2 Simulation results

Simulation results were obtained from the models described in section 4.0. Results from simulation of non-square switches (NO > NI) and for switches with multiple ports per trunk will be presented in section 5.3 with the respective analytical results.

5.2.1 Model validation

As mentioned in section 4.2, the model can be validated by adapting it to particular systems for which performance results are known. Since this adaptation requires that only a very small percentage of the program code (≅ 1 %) be modified, we can be confident about our model behavior under the desired conditions. In figure 5.4, the results obtained for a 1x1, 2x2, 4x4, 8x8 unicast switch are compared to the M/D/1 delay curve and to the saturation throughputs given in table 4.1.

We can see that the 1x1 delay curve matches almost perfectly the M/D/1 curve and that the saturation throughputs of the 2x2, 4x4, 8x8 switches are approaching the values given in table 4.1. This tends to support the validity of our simulation model.

5.2.2 Determining the sample size

We want to determine the number of samples (packets)

that are necessary in each run to achieve the level of accuracy given by equation (4.17). The sample size as a function of the run length (slots) is given by equation (4.16). The required run length was determined by experiment for $\lambda = .1$ and NI = 4 which are the smallest values for which the simulation was run. The desired accuracy was obtained for a run length greater than 4000 slots with the number of runs (K) equal to 8.

5.2.3 Performance results

Average packet delay as a function of the arrival rate λ for a 4x4, an 8x8 and a 16x16 switch are plotted in figures 5.5, 5.6 and 5.7. Again these curves were obtained for different multicast probabilities (P). These results, with others, will be compared to analytical results in the next section.

5.3 Comparison of analytical and simulation results

We now compare analytical and simulation results. The analytical model selected for comparison is the MR_model, which considers the residual distribution of the generated copies for evaluating the interfering traffic. This approach is more realistic and gives more accurate results when compared to simulation.

Besides the square switch configuration (NI = NO), we consider two other types of switches that are well suited for multicast applications. The first type is the distribution switch (NO > NI) and the second type is the grouped output ports switch where NO = NT x S with S > 1 (section 3.6). The analytical and simulation results for the different switch configurations and sizes are plotted in figures 5.8 to 5.14.

We first concentrate on the square and distribution switches for which analytical and simulation results can be compared directly (Figure 5.8 to 5.12). From these curves we can see that the analytical results are very close to the simulation results, especially when the switch size and the multicast probability (P) increase. The more optimistic results obtained from the analysis can possibly be attributed to the assumption that was made concerning the independence between input ports for the evaluation of the

interfering traffic. This was assumed in section 3.5.1 to simplify the development of the expression for the probability generating function $R_T(z)$.

We now look at results obtained for the grouped output switch configuration. An approximate analysis giving a lower bound on delay was developed in section 3.6. We found that the approximate results from this analysis were very poor and that there was no point in comparing them with simulation results. We also tried other approximation method like the Chernoff bound without getting better results. The reason for this is probably the alternating sign summation given in equation 3.13 which makes difficult the computation of any bound.

We propose another method to obtain an approximation to the performance of switches with grouped output ports. If we take for example a 8x16 distribution switch and we limit the number of copies generated at each input port so that it follows a modified binomial distribution with parameters 8 and P, we obtain an approximation for a 8x(8x2) switch. This method of limiting the number of copies generated at each input port gives an upper bound on the delay of grouped output ports switches. This is due to the fact that distributing the same number of copies to 16 single port outputs instead of 8 double port output trunks

results in having more output ports remaining idle. This approximation is compared with simulation results for an 8x(8x2) and a 16x(16x2) switch in figures 5.13 and 5.14.

The results show that the proposed analytical model gives a good approximation of delay for grouped output port switches and that the upper bound obtained gets tighter as the switch size and the multicast probability P increase. We expect that for larger switches, (64x64), the analytical results would be almost identical to simulation results.

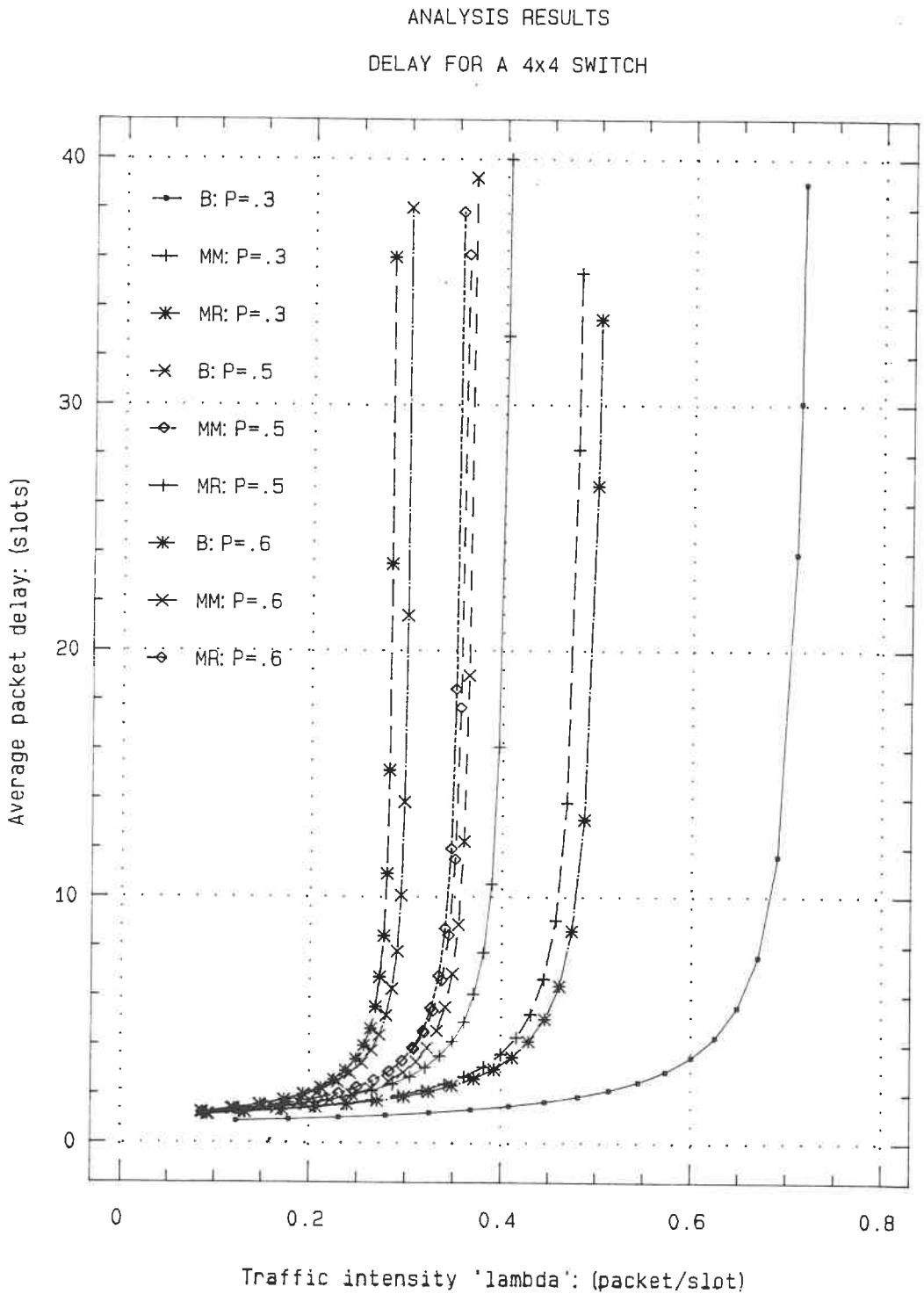

ANALYSIS RESULTS
DELAY FOR A 4x4 SWITCH
B: P=.3
MM: P=.3
MR: P=.3
B: P=.5
MM: P=.5
MR: P=.5
B: P=.6
MM: P=.6
MR: P=.6
Average packet delay: (slots)
0
10
20
30
40
0
0.2
0.4
0.6
0.8
Traffic intensity 'lambda': (packet/slot)

Figure 5.1

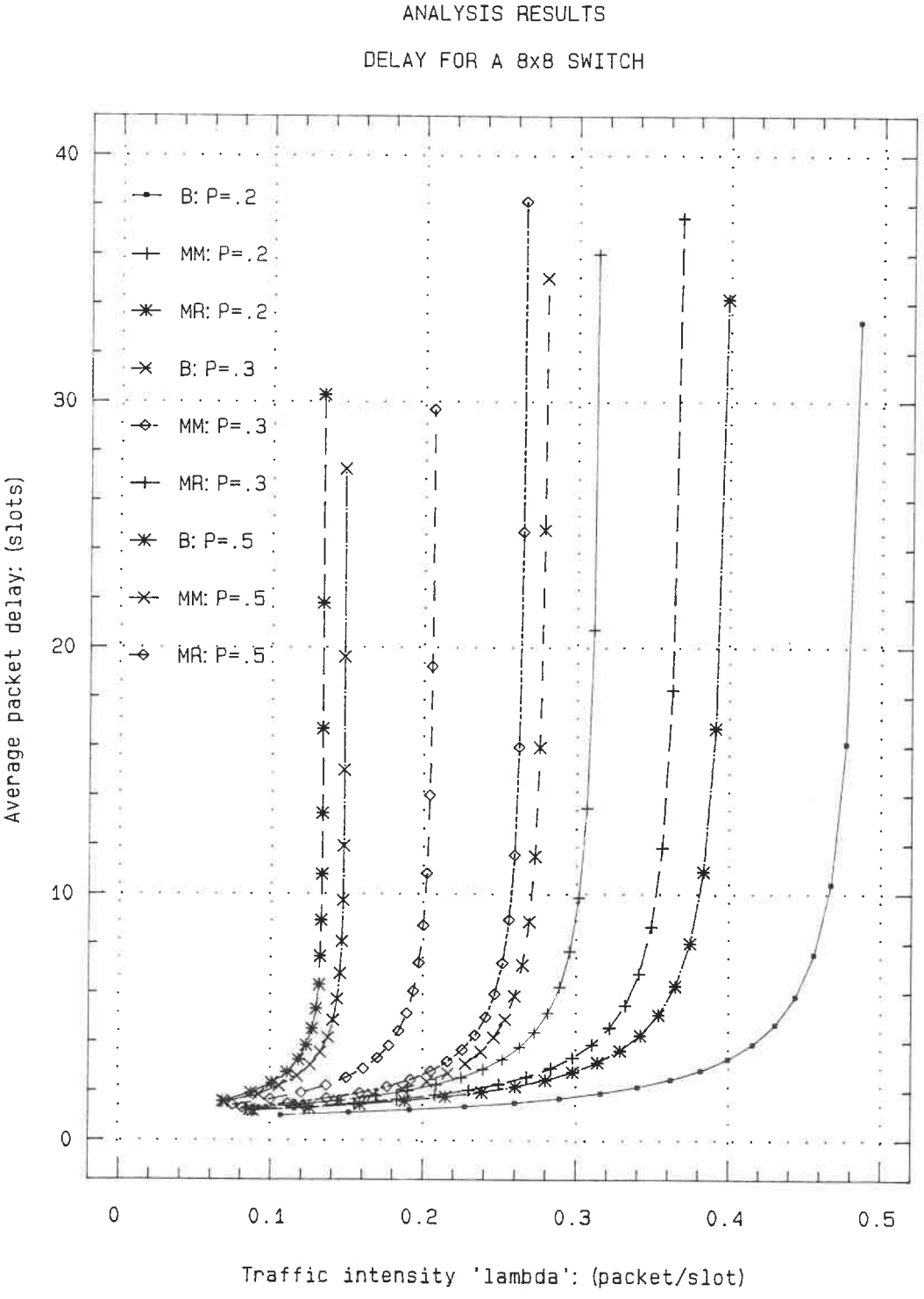

ANALYSIS RESULTS
DELAY FOR A 8x8 SWITCH
B: P=.2
MM: P=.2
MR: P=.2
B: P=.3
MM: P=.3
MR: P=.3
B: P=.5
MM: P=.5
MR: P=.5
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
Traffic intensity 'lambda': (packet/slot)

Figure 5.2

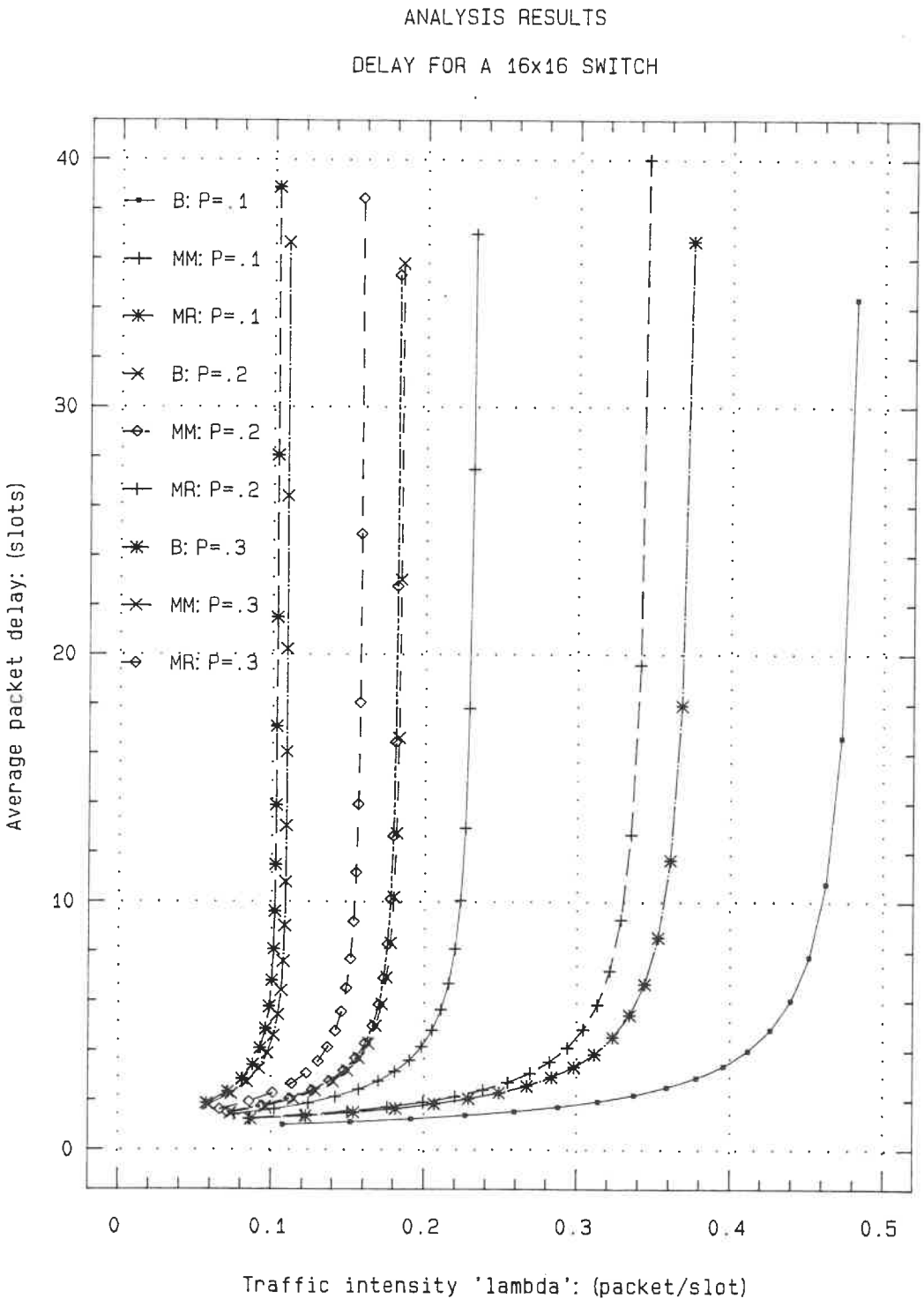

ANALYSIS RESULTS
DELAY FOR A 16x16 SWITCH
B: P=.1
MM: P=.1
MR: P=.1
B: P=.2
MM: P=.2
MR: P=.2
B: P=.3
MM: P=.3
MR: P=.3
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
Traffic intensity 'lambda': (packet/slot)

Figure 5.3

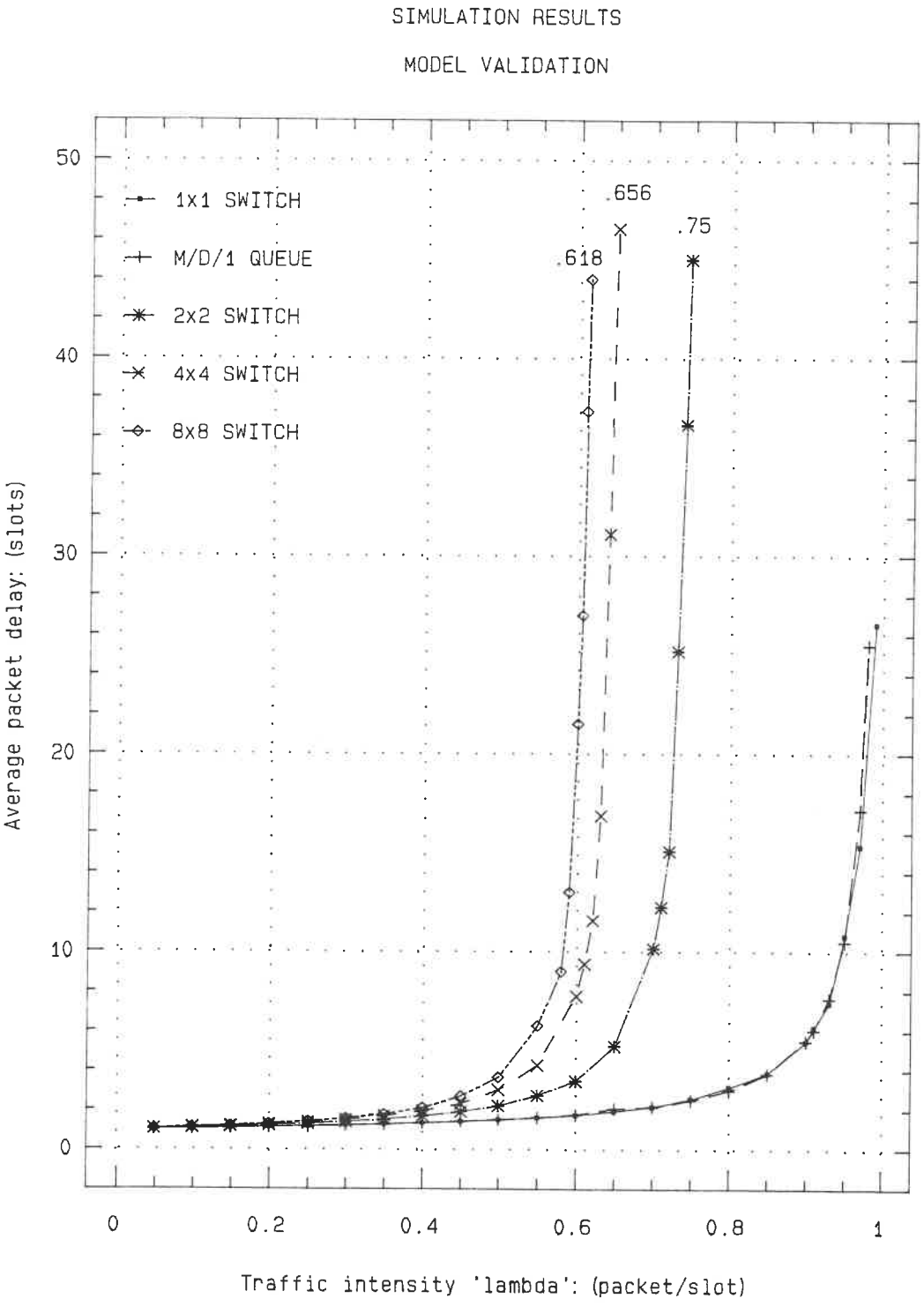

SIMULATION RESULTS
MODEL VALIDATION
1x1 SWITCH
M/D/1 QUEUE
2x2 SWITCH
4x4 SWITCH
8x8 SWITCH
.618
.656
.75
Average packet delay: (slots)
Traffic intensity 'lambda': (packet/slot)
0
10
20
30
40
50
0
0.2
0.4
0.6
0.8
1

Figure 5.4

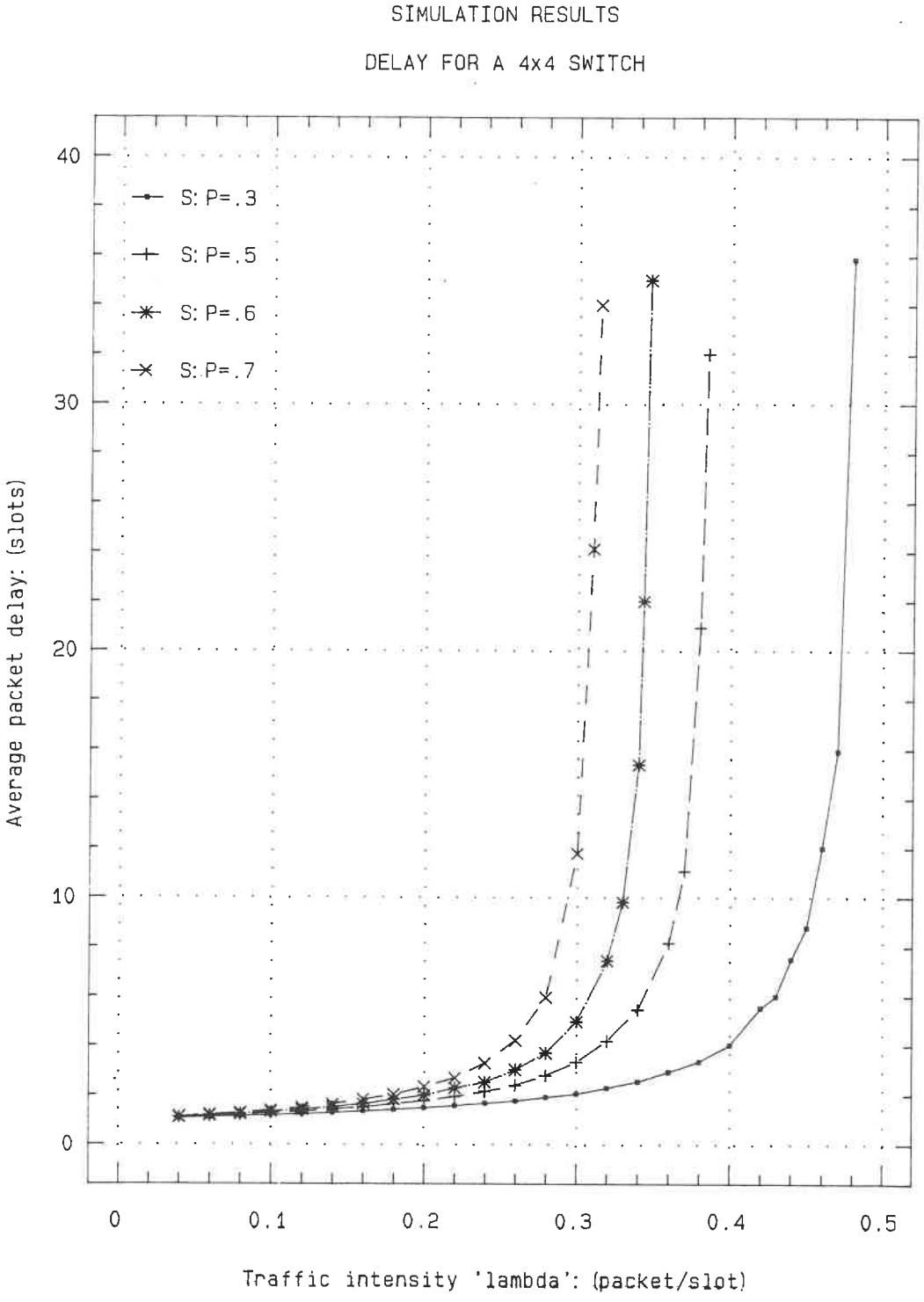

SIMULATION RESULTS
DELAY FOR A 4x4 SWITCH
S: P=.3
S: P=.5
S: P=.6
S: P=.7
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
Traffic intensity 'lambda': (packet/slot)

Figure 5.5

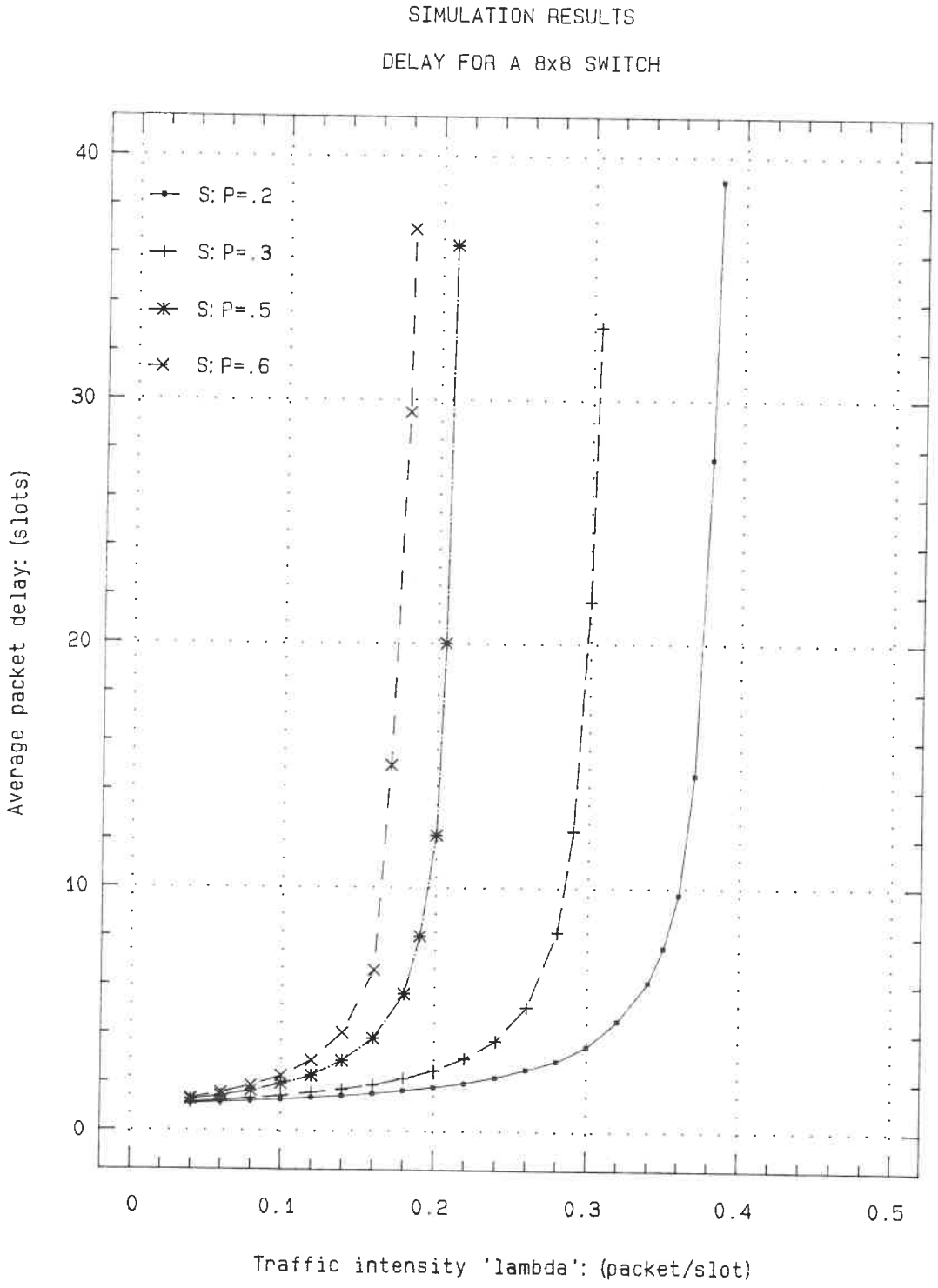

SIMULATION RESULTS
DELAY FOR A 8x8 SWITCH
S: P=.2
S: P=.3
S: P=.5
S: P=.6
40
30
20
10
0
0
0.1
0.2
0.3
0.4
0.5
Average packet delay: (slots)
Traffic intensity 'lambda': (packet/slot)

Figure 5.6

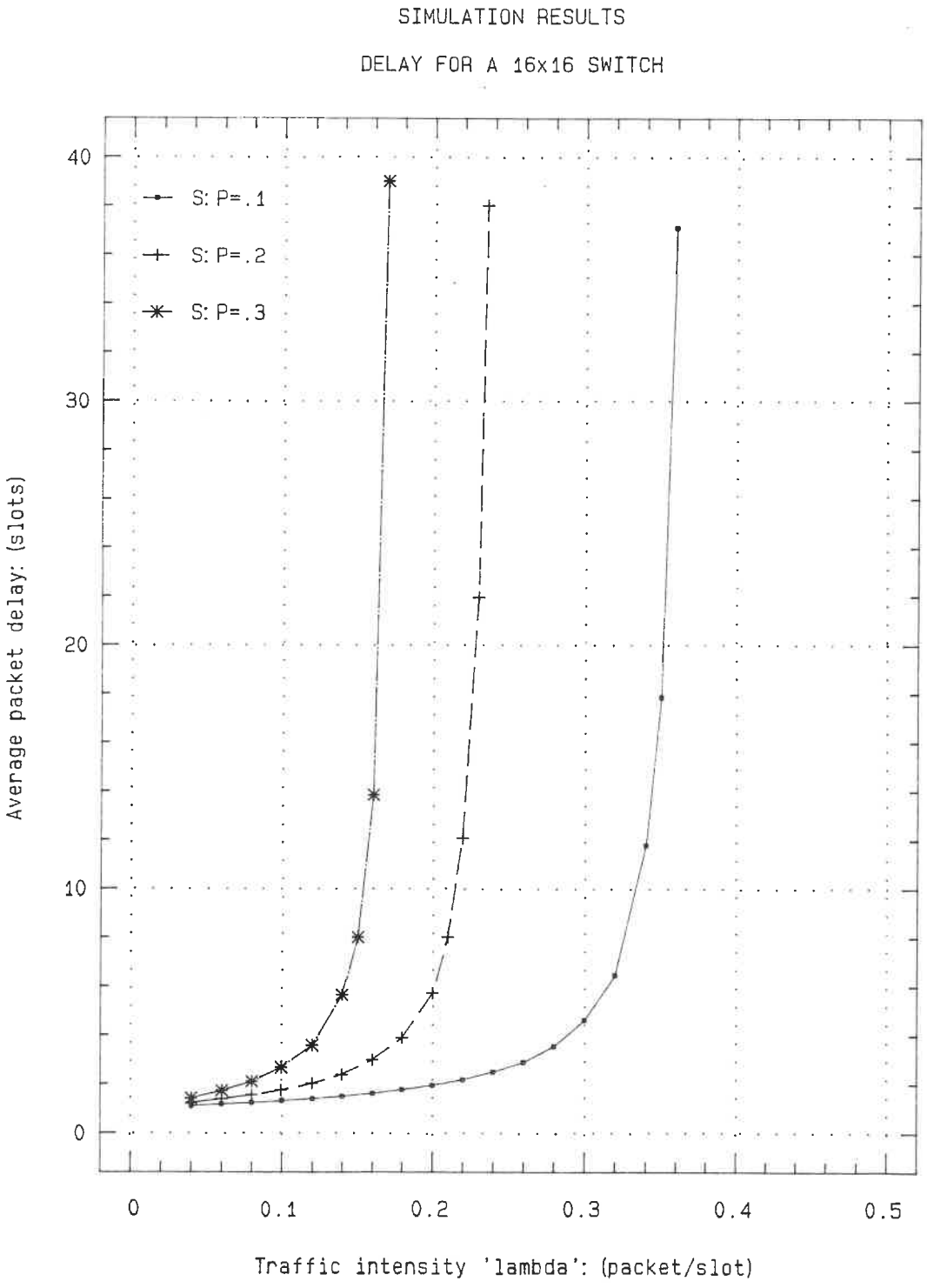

SIMULATION RESULTS
DELAY FOR A 16x16 SWITCH
S: P=.1
S: P=.2
S: P=.3
40
30
20
10
0
0
0.1
0.2
0.3
0.4
0.5
Average packet delay: (slots)
Traffic intensity 'lambda': (packet/slot)

Figure 5.7

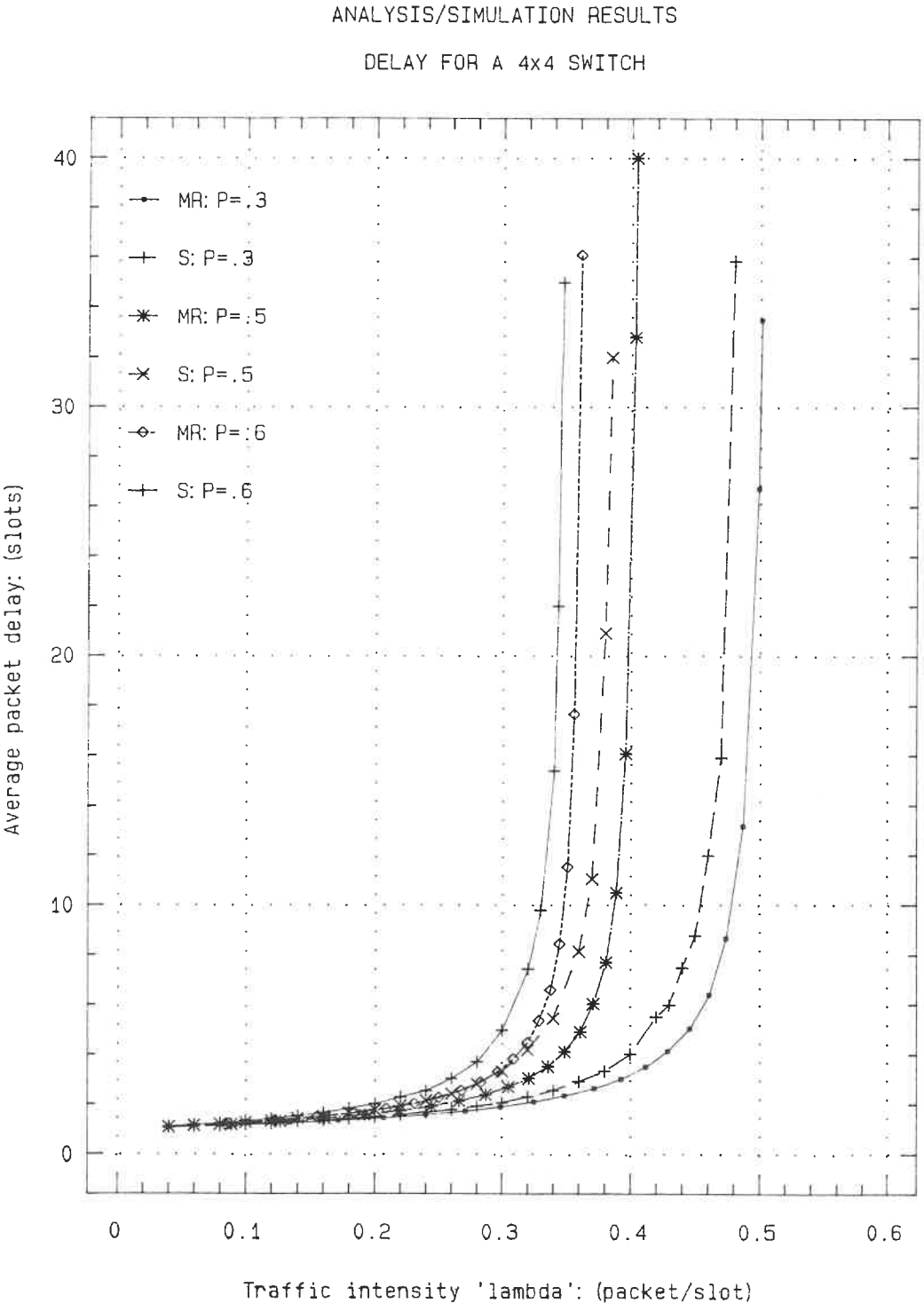

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 4x4 SWITCH
MR: P=.3
S: P=.3
MR: P=.5
S: P=.5
MR: P=.6
S: P=.6
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
0.6
Traffic intensity 'lambda': (packet/slot)

Figure 5.8

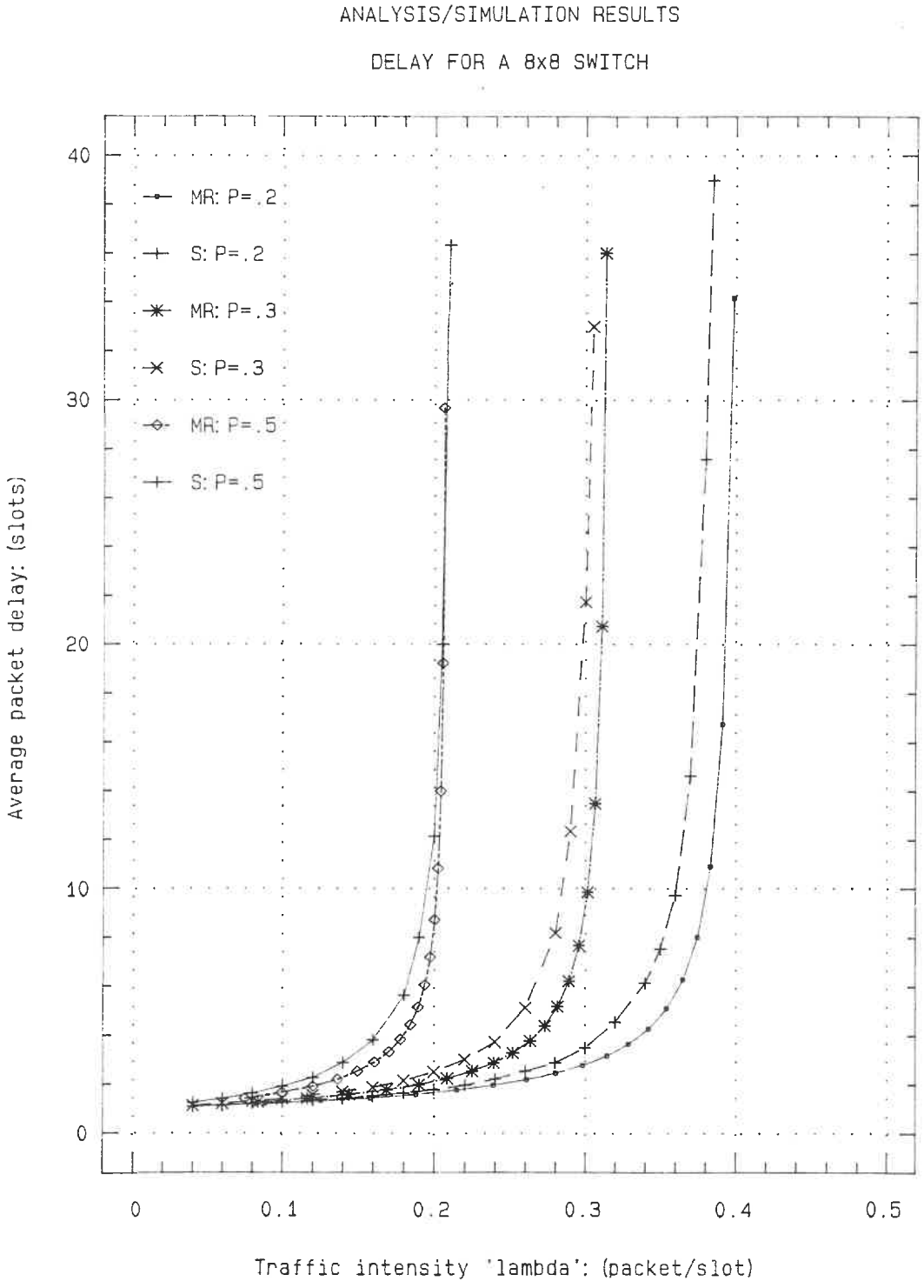

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 8x8 SWITCH
MR: P=.2
S: P=.2
MR: P=.3
S: P=.3
MR: P=.5
S: P=.5
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
Traffic intensity 'lambda': (packet/slot)

Figure 5.9

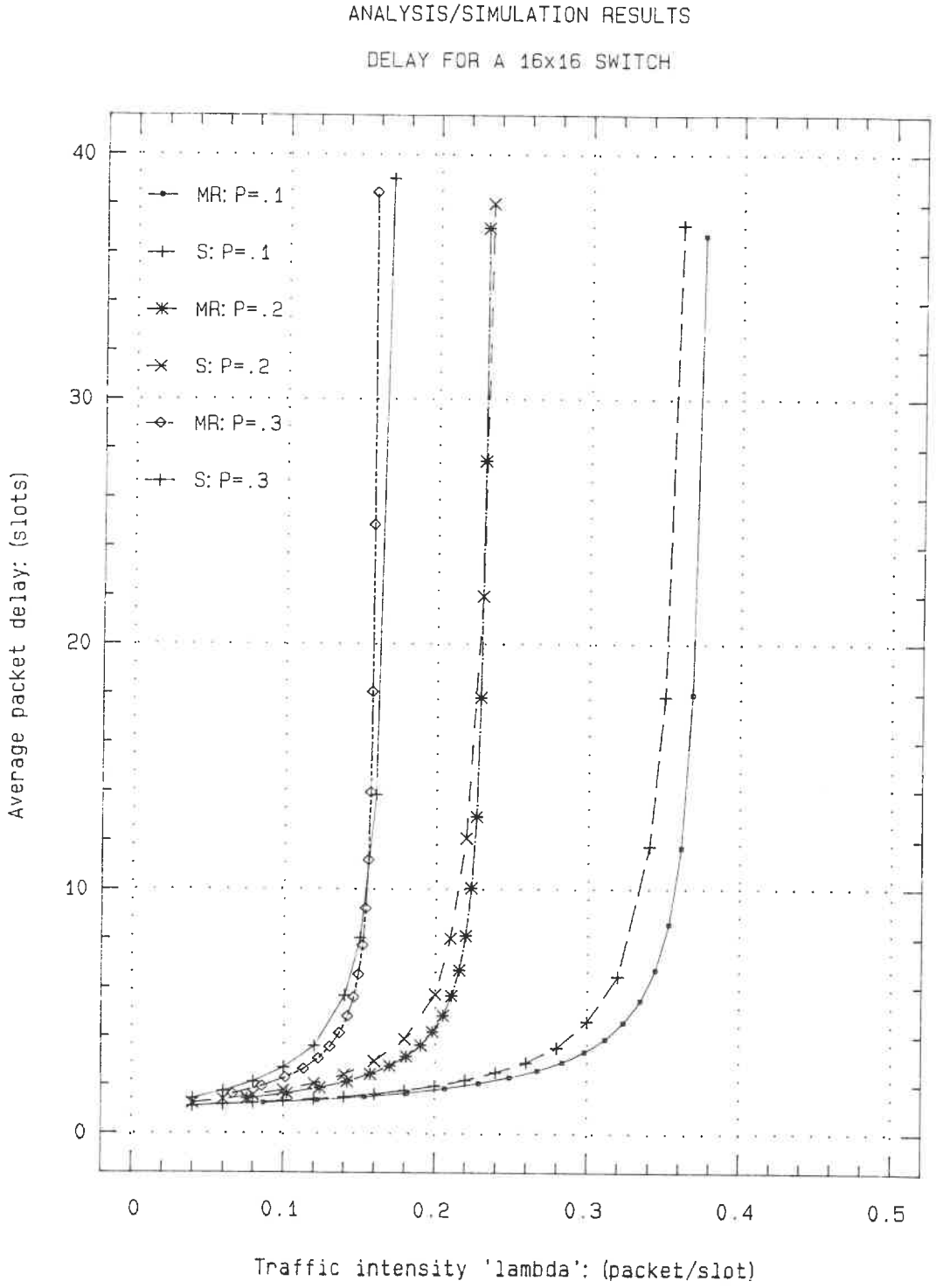

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 16x16 SWITCH
MR: P=.1
S: P=.1
MR: P=.2
S: P=.2
MR: P=.3
S: P=.3
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
Traffic intensity 'lambda': (packet/slot)

Figure 5.10

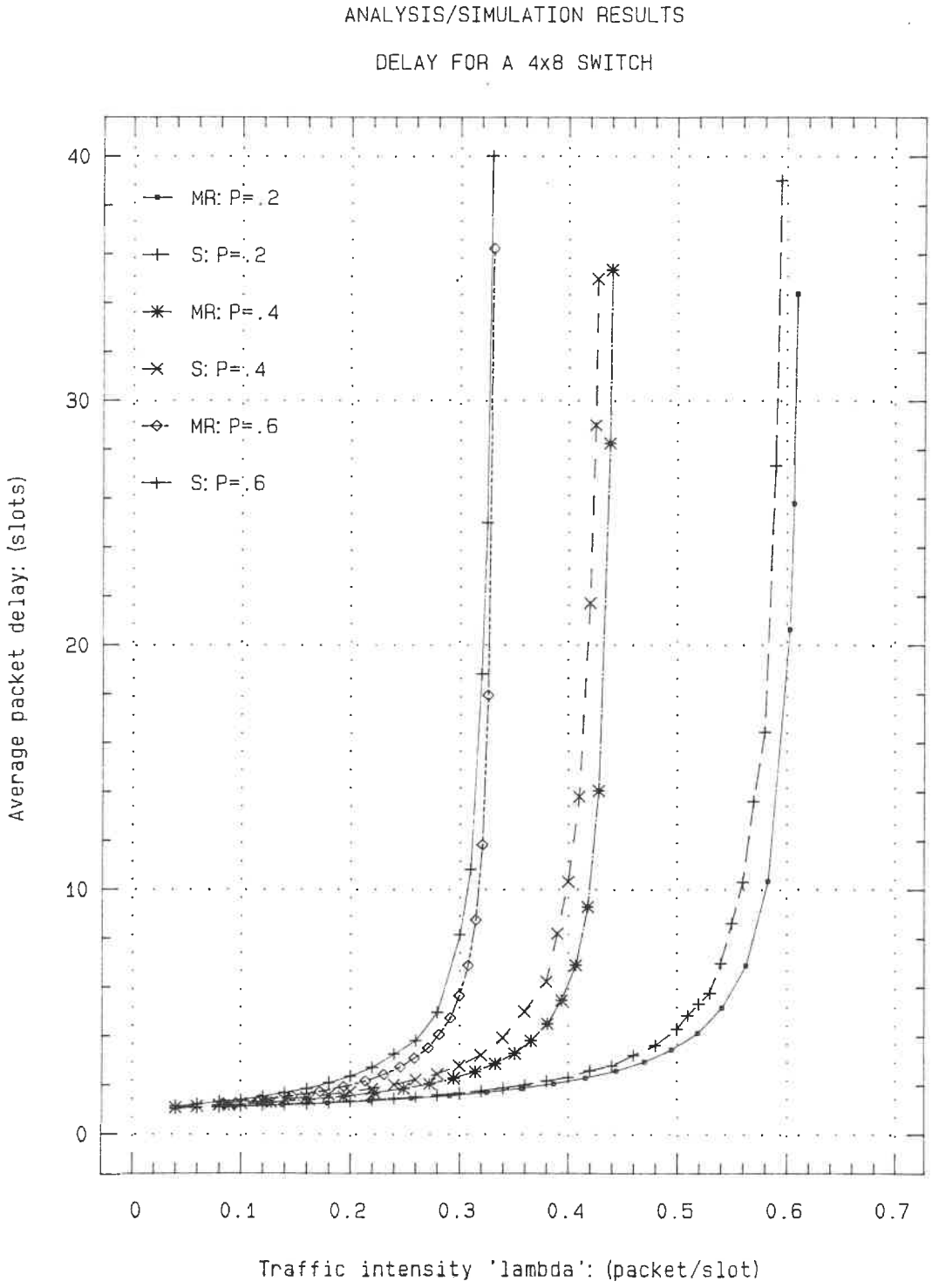

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 4x8 SWITCH
MR: P=.2
S: P=.2
MR: P=.4
S: P=.4
MR: P=.6
S: P=.6
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Traffic intensity 'lambda': (packet/slot)

Figure 5.11

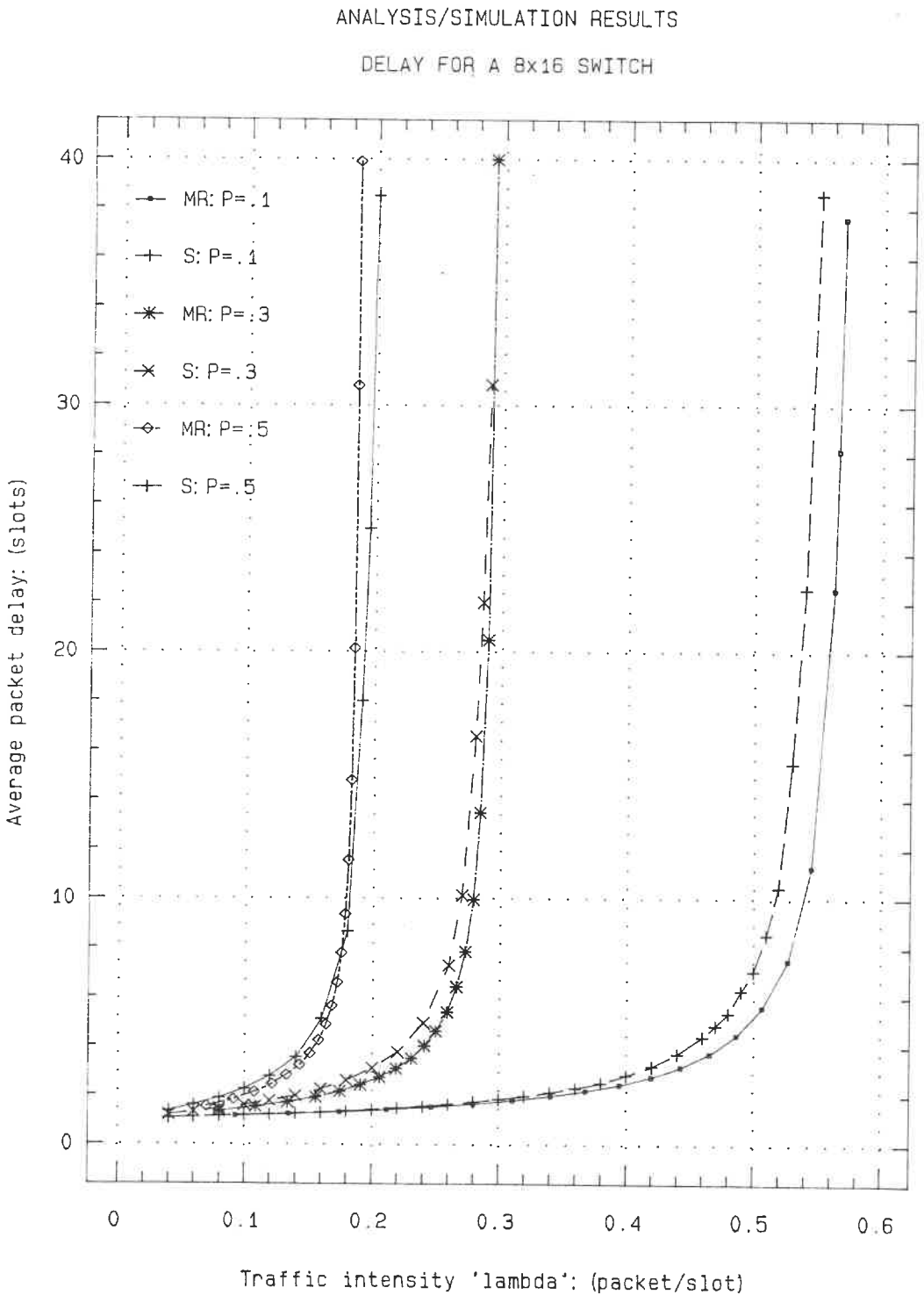

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 8x16 SWITCH
MR: P=.1
S: P=.1
MR: P=.3
S: P=.3
MR: P=.5
S: P=.5
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
0.6
Traffic intensity 'lambda': (packet/slot)

Figure 5.12

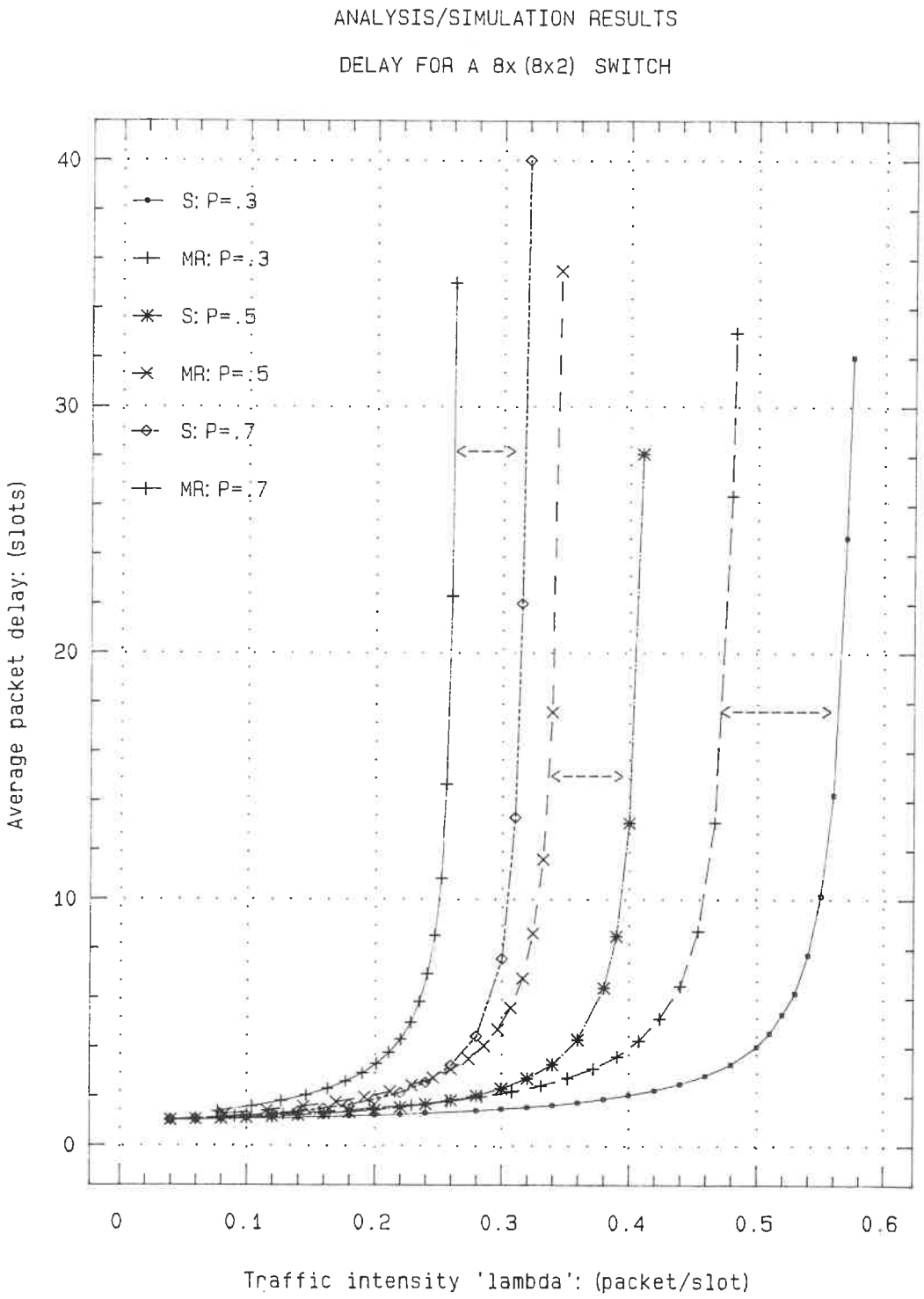

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 8x(8x2) SWITCH
S: P=.3
MR: P=.3
S: P=.5
MR: P=.5
S: P=.7
MR: P=.7
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
0.6
Traffic intensity 'lambda': (packet/slot)

Figure 5.13

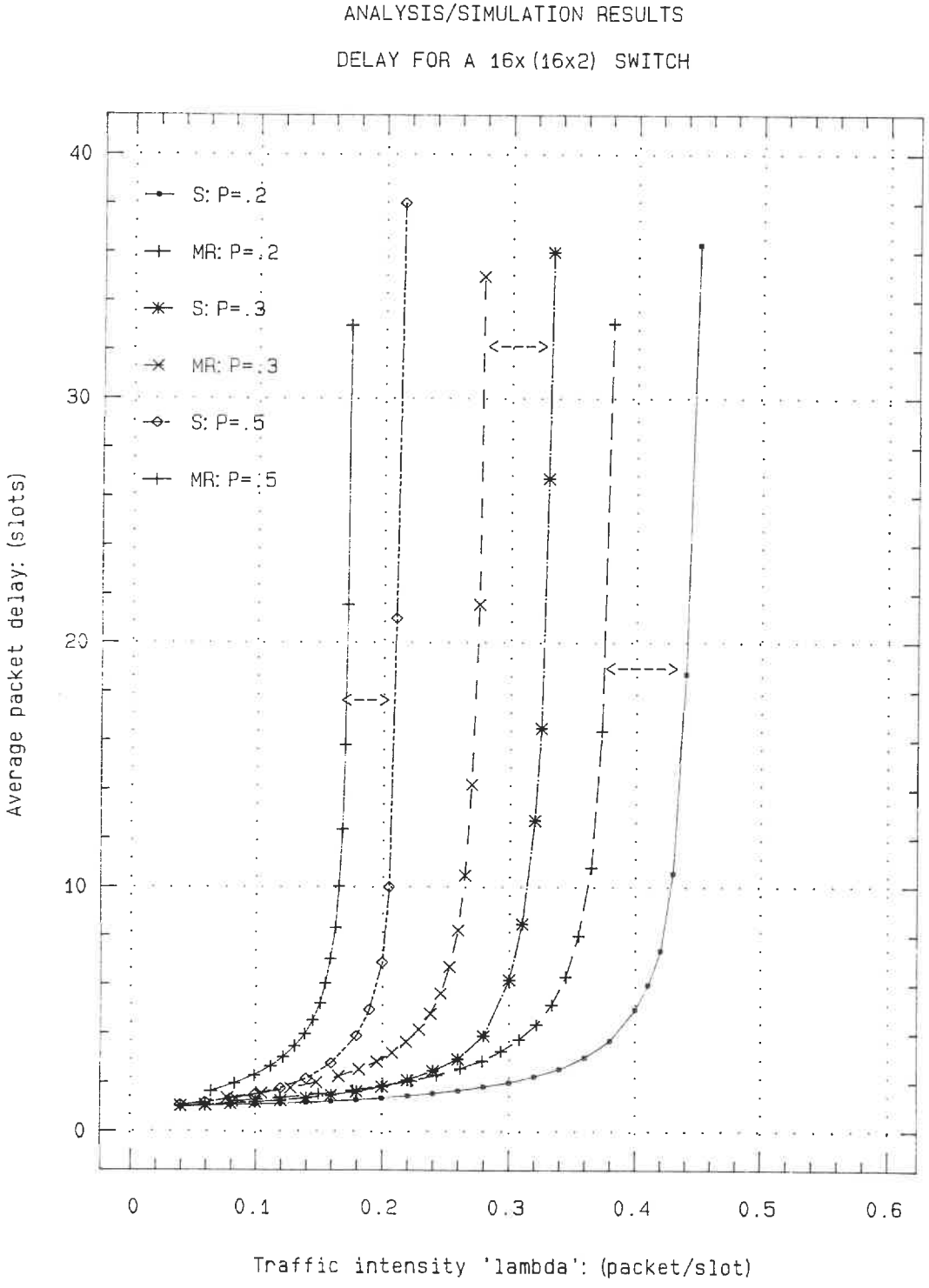

ANALYSIS/SIMULATION RESULTS
DELAY FOR A 16x(16x2) SWITCH
S: P=.2
MR: P=.2
S: P=.3
MR: P=.3
S: P=.5
MR: P=.5
Average packet delay: (slots)
0
10
20
30
40
0
0.1
0.2
0.3
0.4
0.5
0.6
Traffic intensity 'lambda': (packet/slot)

Figure 5.14

6. Conclusion

This thesis has studied a model which has permitted us to evaluate the performance of Broadband Packet Switch (BPS) in a multicast environment. These performance results were obtained from analysis and simulation. The analysis was developed for different multicast models with increasing degrees of refinement. The analytical results of the more refined model were then compared to simulation results. An approximation was also proposed for multicast switches containing multiple output ports at each output trunk.

The results obtained for square and distribution switches seem to show that the analysis gives a lower bound on packet delay. This lower bound, which is probably due to the independence assumption between input ports that was adopted in the analysis, becomes very tight as the switch size is increased. The approximation method for grouped output port switches that was proposed in section 5.3 gives an upper bound on packet delay. Again this upper bound is expected to become very tight as the switch size is increased.

Future research on multicast packet switches should include consideration of traffic classes such as reserved and unreserved packets. Reserved packets can be used to

emulate circuit switching for services that cannot tolerate the queuing delay of unreserved packets. One should also consider translating the analysis program from C to Pascal in order to get results for larger switches.

REFERENCES

[1] J.J. Kulzer, W.A. Montgomery, "Statistical switching architecture for future services," Proceeding of the International Switching Symposium (ISS), Florence, May 1984.

[2] J.S. Turner, "New directions in communications (or which way to the information age)," IEEE Commun. Mag., vol 24, Oct. 1986, pp. 8-15.

[3] P.Gonet, P. Adams, J.P. Goudreuse, "Asynchronous time division switching: The way to flexible broadband communication networks," IEEE 1986 Proceeding of the International Zurich Seminar on Digital Commun., Zurich, Mar. 1986, pp. 141-148.

[4] L.T. Wu, S.H. Lee, T.T. Lee, "Dynamic TDM - A packet approach to broadband networking," Proc. IEEE ICC '87 , Seattle, WA, June 1987.

[5] B. Schaffer, "Synchronous and asynchronous transfer modes in the future broadband ISDN," Proc. IEEE Globecom '88, Fort Lauderdale, FL., Nov. 1988, pp. 1552-1558.

[6] B. Eklundh, I. Gard, G. Leijonhufvud, "A layered architecture for ATM networks," Proc. IEEE Globecom '88, Fort Lauderdale, FL., Nov. 1988, pp. 409-414.

[7] J.Y. Hui, E. Arthurs, "A broadband packet switch for integrated transport," IEEE J. Select. Areas Commun., vol. SAC-5, No. 8, Oct. 1987, pp. 1264-1273.

[8] A. Huang, S. Knauer, "Starlite: A wideband digital switch," Proc. IEEE Globecom '84, pp. 121-125.

[9] K.E. Batcher, "Sorting networks and their applications," AFIPS Proc. Spring Joint Comput. Conf., 1968, pp. 307-314.

[10] T.T. Lee, "Nonblocking copy networks for multicast packet switching," IEEE J. Select. Areas Commun., vol. 6, No. 9, Dec. 1988, pp. 1455-1467.

[11] T.T. Lee, R. Boorstyn, E. Arthurs, "The architecture of a multicast broadband packet switch," Proc. IEEE Infocom '88, New Orleans, LA., pp. 1-8.

[12] J.S. Turner, "Design of a broadcast packet switching network," Proc. IEEE Infocom '86, pp. 667-675.

[13] J.F. Hayes, "Modeling and Analysis of Computer Communication Networks," Plenum Press, 1984.

[14] L. Kleinrock, "Queuing Systems Volume 1 : Theory," New York: Wiley-Interscience, 1975.

[15] M.J. Karol, M.G. Hluchyj, S.P. Morgan, "Input vs. output queuing on a space-division packet switch," IEEE Trans. Commun., vol. COM-35, Dec.1987, pp. 1347-1356.

[16] S.M. Ross, "Introduction to probability models," 3rd edition, Academic Press, 1985.

[17] H. Kobayashi, "Modeling and Analysis: An introduction to system performance evaluation methodology," Addison Wesley, 1978.

APPENDIX A

P.G.F. of the modified binomial distribution

For a random variable M following a modified binomial distribution with parameters NO and P, we have the following probability mass function

$$\text{Prob}(M = i) = \binom{NO}{i} \frac{P^i (1-P)^{NO-i}}{1 - (1-P)^{NO}} \quad ;i = 1,\ldots,NO \qquad (A.1)$$

The P.G.F. of M is define as

$$M(z) \triangleq E[\, z^M \,] = \sum_{i=1}^{NO} z^i \, \text{Prob}(M = i) \qquad (A.2)$$

We then have

$$M(z) = \frac{1}{1 - (1-P)^{NO}} \sum_{i=1}^{NO} \binom{NO}{i} P^i z^i (1-P)^{NO-i} \qquad (A.3)$$

$$M(z) = \frac{1}{1 - (1-P)^{NO}} \left[\sum_{i=0}^{NO} \binom{NO}{i} P^i z^i (1-P)^{NO-i} - (1-P)^{NO} \right] \qquad (A.4)$$

$$M(z) = \frac{1}{1 - (1-P)^{NO}} \left[\left[Pz + (1-P) \right]^{NO} - (1-P)^{NO} \right] \qquad (A.5)$$

where $[Pz + (1-P)]^{NO}$ is the probability generating function of the binomial distribution.

APPENDIX B

Integration of the P.G.F. of the multinomial distribution

We want to integrate the following expression

$$\text{Prob (first L or more } | \text{ } M_T, \text{ K)} = \tag{B.1}$$

$$\int_0^1 dz_1 \int_0^1 dz_2 \ldots \int_0^1 dz_L \left[\sum_{j=1}^{L} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T}$$

By first integrating over z_L from 0 to 1 we get

$$\int_0^1 \left[\sum_{j=1}^{L} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T} dz_L$$

$$= \frac{NO}{(M_T + 1)} \left[\sum_{j=1}^{L} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T + 1} \Bigg|_0^1 \tag{B.2}$$

$$= \frac{NO}{(M_T + 1)} \left\{ \left[\sum_{j=1}^{L-1} \frac{1}{NO} z_j + \left(1 - \frac{(L-1)}{NO} \right) \right]^{M_T + 1} \right. \tag{B.3}$$

$$\left. - \left[\sum_{j=1}^{L-1} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T + 1} \right\}$$

We can repeat the previous step for z_{L-1}

$$\frac{NO}{(M_T+1)} \int_0^1 \left\{ \left[\sum_{j=1}^{L-1} \frac{1}{NO} z_j + \left(1 - \frac{(L-1)}{NO} \right) \right]^{M_T+1} \right.$$

$$\left. - \left[\sum_{j=1}^{L-1} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T+1} \right\} dz_{L-1}$$

$$= \frac{NO^2}{(M_T+1)(M_T+2)} \left\{ \left[\sum_{j=1}^{L-2} \frac{1}{NO} z_j + \left(1 - \frac{(L-2)}{NO} \right) \right]^{M_T+2} \right. \quad \text{(B.4)}$$

$$- 2 \left[\sum_{j=1}^{L-2} \frac{1}{NO} z_j + \left(1 - \frac{(L-1)}{NO} \right) \right]^{M_T+2}$$

$$\left. + \left[\sum_{j=1}^{L-2} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T+2} \right\}$$

For z_{L-2} we can guess the resulting term

$$\frac{NO^3}{(M_T+1)(M_T+2)(M_T+3)} \left\{ \left[\sum_{j=1}^{L-3} \frac{1}{NO} z_j + \left(1 - \frac{(L-3)}{NO} \right) \right]^{M_T+3} \right. \quad \text{(B.5)}$$

$$- 3 \left[\sum_{j=1}^{L-3} \frac{1}{NO} z_j + \left(1 - \frac{(L-2)}{NO} \right) \right]^{M_T+3}$$

$$+ 3 \left[\sum_{j=1}^{L-3} \frac{1}{NO} z_j + \left(1 - \frac{(L-1)}{NO} \right) \right]^{M_T+3}$$

$$\left. + \left[\sum_{j=1}^{L-3} \frac{1}{NO} z_j + \left(1 - \frac{L}{NO} \right) \right]^{M_T+3} \right\}$$

After $n < L$ steps we have

$$\prod_{j=1}^{n} \frac{(NO)^n}{(M_T + j)} \sum_{i=0}^{n} \binom{n}{i} (-1)^i \left[\sum_{j=1}^{L-n} \frac{1}{NO} z_j + \left(1 - \frac{(L-n+i)}{NO} \right) \right]^{M_T + n} \tag{B.6}$$

We then get the final result after L steps

$$\text{Prob (first L or more } | \text{ } M_T, K) = \tag{B.7}$$

$$\prod_{j=1}^{L} \frac{(NO)^L}{(M_T + j)} \sum_{i=0}^{L} \binom{L}{i} (-1)^i \left[1 - i/NO \right]^{M_T + L} \quad ; L \leq K$$

APPENDIX C

P.G.F. of the residual of a discrete random variable

The residual probability mass function of any discrete random variable X is given by equation (C.1)

$$\text{Prob}(R = j) = \frac{1}{\bar{X}} \sum_{i=j}^{NO} \text{Prob}\ (X = i) \quad ;j = 1,..,NO \quad ;i \geq j \qquad (C.1)$$

The probability generating function R(z) is then given by

$$R(z) \triangleq \sum_{j=1}^{\infty} z^j\ \text{Prob}(R = j) \qquad (C.2)$$

We can develop an expression for it

$$R(z) = \frac{1}{\bar{X}} \sum_{j=1}^{\infty} z^j \sum_{i=j}^{NO} \text{Prob}(X = i) \qquad (C.3)$$

$$= \frac{1}{\bar{X}} \sum_{j=1}^{\infty} z^j \left[\sum_{i=0}^{NO} \text{Prob}(X = i) - \sum_{i=0}^{j-1} \text{Prob}(X = i) \right] \qquad (C.4)$$

From the total probability law we have

$$R(z) = \frac{1}{\bar{X}} \left[\sum_{j=1}^{\infty} z^j - \sum_{j=1}^{\infty} z^j \sum_{i=0}^{j-1} \text{Prob}(X = i) \right] \qquad (C.5)$$

$$= \frac{1}{\bar{X}} \left[z \sum_{j=1}^{\infty} z^{j-1} - z \sum_{j=1}^{\infty} z^{j-1} \sum_{i=0}^{j-1} \text{Prob}(X = i) \right] \quad (C.6)$$

If we define k = j-1 we have

$$R(z) = \frac{1}{\bar{X}} \left[z \sum_{k=0}^{\infty} z^{k} - z \sum_{k=0}^{\infty} z^{k} \sum_{i=0}^{k} \text{Prob}(X = i) \right] \quad (C.7)$$

From the properties of the z-Transform we get

$$R(z) = \frac{1}{\bar{X}} \left[\frac{z}{(1-z)} - \frac{z\, X(z)}{(1-z)} \right] \quad (C.8)$$

$$= \frac{z}{\bar{X}} \left[\frac{1 - X(z)}{(1-z)} \right] \quad (C.9)$$

where X(z) is the P.G.F. of the random variable X. If X follows a modified binomial distribution with parameter NO and P, we get the following expressions for R(z)

$$R(z) = \frac{z\,[1-(1-P)^{NO}]}{NO\ P\ (1-z)} \left[1 - \frac{[Pz + (1-P)^{NO}] - (1-P)^{NO}}{[1 - (1-P)^{NO}]} \right] \quad (C.10)$$

$$= \frac{z}{NO\ P\ (1-z)} \left[1 - [\, Pz + (1-P)]^{NO} \right] \quad (C.11)$$

where

$$\bar{X} = \frac{NO\ P}{[\ 1-\ (1-P)^{NO}]} \qquad (C.12)$$

It is easy to show that for a binomial random variable Y, we obtain the same result. That is, the residual of a modified binomial distribution is equal to the residual of a binomial distribution.

APPENDIX D

Analysis programs

This appendix contains the following programs and "header" files.

MULTI1.C : Analysis program for the Bernouilli traffic distribution model (B_model) developed in section 3.3.

MULTI3.C : Analysis program for the multinomial traffic distribution model (MM_model) developed in section 3.4.

MULTI4.C : Analysis program for the mutinomial traffic distribution model with residual service time (MR_model) developed in section 3.5. It also supports the grouped output ports model (section 3.6).

SIZE.H : "Header" file include in each program to define the switch parameters.

```
/*****************************************************************/
/*                                                               */
/*      MULTI1.C :                                               */
/*                                                               */
/*      Program to compute the probabilities related to the      */
/*      MULTICAST SWITCH analysis. The switch has NI inputs      */
/*      and NO outputs. Packets arrive at the inputs at a        */
/*      rate = L packets per second. Each packet generates a     */
/*      random number of copies according to a BINOMIAL          */
/*      distribution with parameters PEE and NO.                 */
/*      The choosing of a particular output port is a BERNOUILLI*/
/*      trial with parameter PEE. Time is sliced in slots and    */
/*      the service time corespond to the number of slots        */
/*      required to transmit all the copies generated .          */
/*      Conflicts at outputs are resolved by random selection.   */
/*      Copies that have not gain acces to an output port have   */
/*      to be retransmit.                                        */
/*                                                               */
/*****************************************************************/


#include <size.h>

main()

{

/***************/
/* VARIABLES   */
/***************/

double  rho,pee;
short   i,j;

/*************/
/*  CORE     */
/*************/

pee = PEEMIN;
while ( pee <= PEEMAX )
    {
    rho = RHOMIN;
    while ( rho <= RHOMAX )
        {
        compu(rho,pee);
        if ( rho < .95 )
            {
            rho = rho + RHOINC;
            }
        else
            {
            rho = rho + (RHOINC/5.0);
```

```
            }
        }

    pee = pee + PEEINC;
    }
}

/*****************************************************************/
/*     compu.c                                                   */
/*                                                               */
/*      Fonction to compute the service time and delay of the    */
/*      multicast switch.                                        */
/*****************************************************************/

compu(rho,pee)


/*************/
/* PARAMETERS*/
/*************/

double  rho,pee;

        {

        /*************/
        /* variables */
        /*************/

        short   i,l,j,k,n;
        short   expo;
        double  para_lng,delay,ser_square,lamda;
        double  rho_fact,pro_fact,ser_time;
        double  power(),product(),binomi(),odd_par,even_par;
        double  bin_fact,even_sum,odd_sum,sum;
        static  double  p_l_mo[KMAX+1],p_o_l[KMAX+1][KMAX+1];
        static  double  p_l_k[KMAX+1][KMAX+1],q_n_k[NMAX+1][KMAX+1];
        static  double  p_k,q_n[NMAX+1],tq_n,tp_l_k;


        /*************/
        /* corps     */
        /*************/

        /*********************************************************/
        /* Print  the context information of te calculation      */
        /*********************************************************/


printf("\n\n NI= %2d  NO= %2d  KMAX= %2d  NMAX= %2d RHO= %e PEE= %e",
                                        NI,NO,KMAX,NMAX,rho,pee);
        printf("\n\n");
```

```
/*********************************************************/
/* compute for all vales of "l"                          */
/*          PROB [first l or more throught]              */
/*********************************************************/

    for ( l = 0; l <= KMAX; l++ )
        {
        p_l_mo[l] = 0.0;

        /* compute the summation                         */

        for ( i = 0; i <= NI-1; i++ )
            {

            /* get the binomial factor                   */

            bin_fact = binomi(i,bin_fact,NI-1);

            /* compute the "RHO" products                */

            para_lng = rho;
            rho_fact = power( para_lng,i );
            para_lng = 1.0 - rho;
            expo = (NI - 1)-i;
            rho_fact = rho_fact * power(para_lng,expo);

            /* compute the product factor                */

            para_lng = 1.0 - pee;
            expo = i+1;
            para_lng = power(para_lng,expo);
            para_lng = (1.0-para_lng)/(expo*pee);
            pro_fact = power(para_lng,l);

            /* add to the summation                      */

            p_l_mo[l] = p_l_mo[l] +
                        (bin_fact *rho_fact * pro_fact);
            }


        /*
        printf ("\n\n l = %2d  P[first l or more] = %e",
                                          l,p_l_mo[l]);
        printf("\n\n");
        */
        }

    /*****************************************************/
    /* compute PROB [ first "l" throught|k]              */
```

```
        /*****************************************************/

    for ( k=0; k <= KMAX; k++ )
        {
        for ( l=0; l <= k; l++ )
            {
            /* compute the summation                          */

            sum = 0.0;
            even_sum = 0.0;
            odd_sum = 0.0;
            for ( j=0; j <= k-l; j++ )
                {

                /* get the binomial factor          */

                bin_fact = binomi(j,bin_fact,k-l);

                /* compute the even/odd summations  */

                if ( j % 2 == 0 )
                    {
                   even_sum = even_sum + (bin_fact * p_l_mo[l+j]);
                    sum = sum + (bin_fact * p_l_mo[l+j]);
                    }
                else
                    {
                    odd_sum = odd_sum + (bin_fact * p_l_mo[l+j]);
                    sum = sum - (bin_fact * p_l_mo[l+j]);
                    }

                }

            p_o_l[l][k] = even_sum - odd_sum;

            /*
printf ("\n\n l = %2d  k = %2d sum = %25.15e p[first l|k] = %25.15e",
                                                  l,k,sum,p_o_l[l][k]);
            printf("\n\n");
            */
            }


        /*****************************************************/
        /* compute PROB [ l (all possible sets) |k ]         */
        /*****************************************************/

        tp_l_k = 0.0;
        for (l=0; l<=k; l++ )
            {
```

```
        /* get the binomial factor                          */
        bin_fact = binomi(l,bin_fact,k);
        p_l_k[l][k] = bin_fact * p_o_l[l][k];
        tp_l_k = tp_l_k + p_l_k[l][k];
        /*
        printf ("\n\n l = %2d  k = %2d  p[l all sets|k] = %e",
                                        l,k,p_l_k[l][k]);
       printf("\n\n");
        */
       }
    /*
    printf ("\n\n k = %d tp_l_k = %e",k,tp_l_k);
    */
    }

/*********************************************************/
/* Compute the probability of the number of slots        */
/* required to transmit the k packets condition on k     */
/*********************************************************/


for ( n = 0; n <= NMAX; n++ )
    {
    for ( k = 0; k <= KMAX; k++ )
        {
        if ( k == 0 && n == 0 )
            {
            q_n_k[n][k] = 1.0;
            }
        else if ( n == 0 && k > 0 || k == 0 && n > 0 )
            {
            q_n_k[n][k] = 0.0;
            }
        else if ( n == 1 )
            {
            q_n_k[n][k] = p_l_k[k][k];
            }
        else
            {
            q_n_k[n][k] = 0.0;
            for ( l = 0; l <= k; l++ )
                {
                q_n_k[n][k] = q_n_k[n][k] +
                              (p_l_k[l][k] * q_n_k[n-1][k-l]);

                }
            }
```

```
        /*
        printf("\n\n  n = %2d  k = %2d  q_n_k = %e",
                                  n,k,q_n_k[n][k]);
        */

        }
    /*
    printf("\n\n");
    */
    }

/*************************************************************/
/*      Compute the number of slots required by avera-       */
/*   ging over all possible values of "k" the number         */
/*   copies generated                                        */
/*************************************************************/

tq_n = 0.0;
for ( n = 0; n <= NMAX; n++ )
    {
   q_n[n] = 0.0;
    for ( k = 0; k <= KMAX; k++ )
        {
        /* Compute the prob. of k copies generated      */

        bin_fact = binomi(k,bin_fact,NO);
        para_lng = pee;
        pro_fact = power(para_lng,k);
        para_lng = 1.0 - para_lng;
        pro_fact = pro_fact * power (para_lng,NO-k);
        p_k = bin_fact * pro_fact;

        /* Compute the prob. of n slots required        */

        q_n[n] = q_n[n] + (p_k * q_n_k[n][k]);
        }
    tq_n = tq_n + q_n[n];
    /*
    printf ("\n\n  n = %2d   q_n = %e",n,q_n[n]);
    */
    }
/*
printf ("\n\n tq_n = %e",tq_n);
*/

/*************************************************************/
/* Compute  the average service time and the average         */
/* delay                                                     */
/*************************************************************/

ser_time = 0.0;
ser_square = 0.0;
```

```
      for ( n = 0; n <= NMAX; n++)
          {
          ser_time = ser_time + ((double)n * q_n[n] );
          ser_square = ser_square + ((double)(n * n) * q_n[n] );
          }
      /*
      printf ("\n\n average service time = %e",ser_time);
      printf ("\n\n");
      printf ("\n\n average (service time)**2 = %e",ser_square);
      printf ("\n\n");
      */
      lamda = (rho / ser_time);
      /*
      printf ("\n\n lamda = %e msg/slot",lamda);
      printf ("\n\n");
      */
      delay = ser_time + ((lamda*ser_square)/(2.0*(1.0-rho)));
printf ("\n\n lamda =%e serv =%e delay =%e",lamda,ser_time,delay);
      printf("\n\n");
      printf ("\n\n");
      }
```

```
/*****************************************************************/
/*                                                               */
/*      Fonction power() qui calcule le resultat du parametre    */
/*      fact eleve a la puissance l.                             */
/*                                                               */
/*      PARAMETRES:     fact    parametre a elever a la puiss.   */
/*                      l       exposant                         */
/*****************************************************************/

double  power ( fact,l )

/***************/
/* PARAMETRES */
/***************/

double          fact;
short           l;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          result;
        short           i;

        /************/
        /* CORPS    */
        /************/

        result = 1.0;
        if ( l != 0 )
            {

            for ( i=1; i<=l; i++ )
                {
                result = result * fact;
                }
            }
        /*
        printf ("\n\n entry = %e power/result = %e",fact,result);
        */
        return (result);
        }


/*****************************************************************/
/*                                                               */
/*      Fonction product() utilisee pour calculer un produit     */
/*      avec variable allant d'une valeur minimal a une valeur   */
/*      maximal.                                                 */
```

```
/*                                                                    */
/*      PARAMETRES:     fact    facteur a multiplier                  */
/*                                                                    */
/*                      l       indice maximal du produit             */
/*                                                                    */
/**********************************************************************/

double  product ( fact,l )

/**************/
/* PARAMETRES */
/**************/

double          fact;
short           l;

        {

        /**************/
        /* VARIABLES  */
        /**************/

        double          result;
        short           i;


        /*************/
        /* CORPS     */
        /*************/

        result = 1.0;
        for ( i = 1; i <= l; i++ )
            {
            result = result * ( fact + i );
            }
        /*
        printf ("\n\n product/result = %e",result);
        */
        return (result);
        }

/**********************************************************************/
/*                                                                    */
/*      Fonction binomi() utilisee pour calculer un coeficient        */
/*      binomial quelconque .                                         */
/*                                                                    */
/*                                                                    */
/*      PARAMETRES:     ind_act = indice actuel du coeficient         */
/*                      pre_fact= coeficient calculer precedem-       */
/*                                ment avec l'indice (ind_act-1)      */
/*                      ind_max = indice maximal                      */
/*                                                                    */
```

```
/*****************************************************************/

double  binomi (ind_act,pre_fact,ind_max)


/***************/
/* PARAMETRES */
/***************/

short           ind_act;
double          pre_fact;
short           ind_max;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          bin_fact;


        /**************/
        /* CORPS      */
        /**************/


        /* compute the binomial factor                    */

        if ( ind_act == 0)
             {
             bin_fact = 1.0;
             }
        else
             {
       bin_fact = pre_fact * ((double)ind_max - (double)ind_act +1.0)
                                                /(double)ind_act;
             }
        /*
        printf("\n\n bin_fact = %e",bin_fact);
        */
        return (bin_fact);
        }
```

```
/*****************************************************************/
/*                                                               */
/*      MULTI3.C :                                               */
/*                                                               */
/*      Program to compute the probabilities related to the      */
/*      MULTICAST SWITCH analysis. The switch has NI inputs      */
/*      and NO outputs. Packets arrive at the inputs at a        */
/*      POISSON rate = L packets per second. Each packet         */
/*      generates a MODIFIED-BINOMIAL number of copies. The      */
/*      copies from all input ports are pooled together and      */
/*      distributed to the output ports according to a           */
/*      MULTINOMIAL distribution.                                */
/*      Time is sliced in slots and the service time corespond   */
/*      to the number of slots required to transmit all the      */
/*      copies generated .                                       */
/*      Conflicts at outputs are resolved by random selection.   */
/*      Copies that have not gain access to an output port have  */
/*      to be retransmit.                                        */
/*      The program computes the probabilities for different     */
/*      values of PEE = prob. of sending a copy to an output     */
/*      port and for different values of RHO = load to the       */
/*      system.                                                  */
/*                                                               */
/*****************************************************************/

#include        <size.h>

static  double  conv[2][MTMAX+1];       /* convolution array       */
static  double  fonct[MTMAX+1];

static double   conv_ave;
static double   conv_var;
static double   conv_tot;

main()

{

/***************/
/* VARIABLES   */
/***************/

double  rho,pee;
short   i,j;

/*************/
/*  CORE     */
/*************/

pee = PEEMIN;
while ( pee <= PEEMAX )
    {
```

```
    rho = RHOMIN;
    while ( rho <= RHOMAX )
        {
        compu(rho,pee);
        if (rho < .95)
            {
            rho = rho + RHOINC;
            }
        else
            {
            rho = rho + (RHOINC/5.0);
            }
        }
    pee = pee + PEEINC;
    }
}

/*****************************************************************/
/*                                                               */
/*      Fonction to compute the service time and delay of the    */
/*      multicast switch.                                        */
/*****************************************************************/

compu(rho,pee)


/*************/
/* PARAMETERS*/
/*************/

double  rho,pee;

        {

        /*************/
        /* variables */
        /*************/

        short   i,l,j,k,n,mt,expo,ind_res;
        double  para_lng,delay,ser_square,lamda,sum2_fact;
        double  rho_fact,pro_fact,ser_time,bi[NO+1];
        double  power(),product(),binomi(),sum_fact,bin2_fact;
        double  bin_fact,even_sum,odd_sum,glo_fact,prob_fact;
        double  p_l_mo[KMAX+1],p_o_l[KMAX+1][KMAX+1];
        double  p_l_k[KMAX+1][KMAX+1],q_n_k[NMAX+1][KMAX+1];
        double  p_k,q_n[NMAX+1],pl_mo_mt[KMAX+1][MTMAX+1];
        double  bi_ave,bi_var,bi_tot;


        /*************/
        /* corps     */
        /*************/
```

```
    /*********************************************************/
    /* Print  the context information of te calculation     */
    /*********************************************************/

    printf("\n\n NI= %2d  NO= %2d  NMAX= %2d  RHO= %e PEE= %e",
                                        NI,NO,NMAX,rho,pee);
    printf("\n\n");


    /*********************************************************/
    /* compute for all possible values of "mt"              */
    /*          PROB [first l throught or more|mt]          */
    /*********************************************************/

for ( mt = 0; mt <= MTMAX; mt++ )
    {
    for ( l = 0; l <= KMAX; l++ )
        {
       pl_mo_mt[l][mt] = 0.0;

        /* compute the global factor        */

        glo_fact = power ((double)NO,l );
        glo_fact = glo_fact / product (mt,l );

        /* compute the summation            */

        expo = mt + l;
        even_sum = 0.0;
        odd_sum = 0.0;

        for ( i = 0; i <= l; i++ )

            {
            /* compute the binomial factor  */

            bin_fact = binomi(i,bin_fact,l);

            /* compute one of the partial summation with    */
            /* the coresponding probability factor          */

            if ( i % 2 == 0 )
                {
                /* even summation           */

                prob_fact = 1.0 - ((double)i/(double)NO);
                prob_fact = power ( prob_fact,expo );
                even_sum = even_sum + (bin_fact * prob_fact);
                }
            else
```

```
                {
                /* odd summation              */

                prob_fact = 1.0 - ((double)i/(double)NO);
                prob_fact = power ( prob_fact,expo );
                odd_sum = odd_sum + (bin_fact * prob_fact);
                }

            }

        pl_mo_mt[l][mt] = glo_fact * ( even_sum - odd_sum );

        /*
        printf ("\n\n  l = %2d  mt =  %2d  P(l or more|mt) = %e",
                                    l,mt,pl_mo_mt[l][mt]);
        printf ("\n\n");
        */
        }
    }

/*************************************************************/
/* compute the PROB [ first l or more ] by averaging over    */
/* all possible values of "mt" the number of interfering     */
/* copies from the NI-1 other input ports                    */
/*************************************************************/

/* Compute the MODIFIED BINOMIAL number of copies generated */
/* by one input.                                             */

    bi[0] =0.0 + (1.0 - rho);
    bi_ave = 0.0;
    bi_tot = 0.0;
    bi_var = 0.0;
    bin_fact = 1.0;
    for ( i = 1; i <= NO; i ++ )
        {
        bin_fact = binomi (i,bin_fact,(short)NO);
        pro_fact = power (pee,i);
        pro_fact = pro_fact / (1.0 - power(1.0-pee,(short)NO));
        para_lng = 1.0 - pee;
       bi[i] = bin_fact * pro_fact * power(para_lng,(short)NO-i);
        bi[i] = rho * bi[i];
        bi_ave = bi_ave + bi[i] * (double)i;
        bi_var = bi_var + bi[i] * (double)(i*i);
        bi_tot = bi_tot + bi[i];
        }
    bi_var = bi_var - ( bi_ave * bi_ave );


#ifdef TEST

    /* print the results                                     */
```

```
    for ( i = 0; i <= NO; i++ )
        {
        printf ("\n\n bi(%d) = %e",i,bi[i]);
        }
    printf ("\n\n bi_ave = %e bi_var = %e bi_tot = %e",
                                        bi_ave,bi_var,bi_tot);
#endif


    /* Compute the convolution to get the distribution of the total*/
    /* number of conflicting copies from NI-1 input ports.         */

    convo (NO,&bi[0],NI-1);
    ind_res = (NI) %2;        /* indicates which array holds results*/

#ifdef TEST

    /* print the results                                           */

    for ( i = 0; i <= MTMAX; i++ )
        {
        printf ("\n\n conv(%d) = %e",i,conv[ind_res][i]);
        }
    printf ("\n\n conv_ave = %e conv_var = %e conv_tot = %e",
                                    conv_ave,conv_var,conv_tot);
#endif

        /* compute the PROB [first 1 or more]          */

            for ( l = 0; l <= KMAX; l++ )
                {
               p_l_mo[l] = 0.0;
                for ( mt = 0; mt <= MTMAX; mt++ )
                    {
                    p_l_mo[l] = p_l_mo[l] +
                                pl_mo_mt[l][mt] * conv[ind_res][mt];
                    }
                /*
              printf("\n\n l = %2d k = %2d p[first 1 or more] = %e",
                                    l,p_l_mo[l]);
                printf ("\n\n);
                */
                }

            /*****************************************************/
            /* compute PROB [ first "1" throught|k]              */
            /*****************************************************/

        for ( k = 0; k <= KMAX; k++ )
```

```
{
for ( l=0; l <= k; l++ )
    {

    /* compute the summation                          */

    even_sum = 0.0;
    odd_sum = 0.0;
    for ( j=0; j <= k-l; j++ )
        {

        /* get the binomial factor                    */

        bin_fact = binomi(j,bin_fact,k-l);

        /* compute the even/odd summations            */

        if ( j % 2 == 0 )
            {
            even_sum = even_sum +
                    (bin_fact * p_l_mo[l+j]);
            }
        else
            {
            odd_sum = odd_sum +
                    (bin_fact * p_l_mo[l+j]);
            }

        }

    p_o_l[l][k] = even_sum - odd_sum;
    /*
    printf ("\n\n l = %2d  k = %2d  p[first l|k] = %e",
                                    l,k,p_o_l[l][k]);
    printf("\n\n");
    */
    }

/*****************************************************/
/* compute PROB [ l (all possible sets) |k ]         */
/*****************************************************/

for (l=0; l<=k; l++ )
    {

    /* get the binomial factor                        */

    bin_fact = binomi(l,bin_fact,k);

    p_l_k[l][k] = bin_fact * p_o_l[l][k];
    /*
  printf ("\n\n l = %2d  k = %2d  p[l all sets|k] = %e",
```

```
                                                    l,k,p_l_k[l][k]);
        */
        }
    }
/*
printf("\n\n");
*/

/*********************************************************/
/* Compute the probability of the number of slots        */
/* required to transmit the k packets condition on k     */
/*********************************************************/


for ( n = 0; n <= NMAX; n++ )
    {
    for ( k = 0; k <= KMAX; k++ )
        {
        if ( k == 0 && n == 0 )
            {
           q_n_k[n][k] = 1.0;
           }
        else if ( n == 0 && k > 0 || k == 0 && n > 0 )
            {
            q_n_k[n][k] = 0.0;
            }
        else if ( n == 1 )
            {
            q_n_k[n][k] = p_l_k[k][k];
            }
        else
            {
            q_n_k[n][k] = 0.0;
            for ( l = 0; l <= k; l++ )
                {
                q_n_k[n][k] = q_n_k[n][k] +
                              (p_l_k[l][k] * q_n_k[n-1][k-l]);

                }
            }
        /*
        printf("\n\n  n = %2d  k = %2d  q_n_k = %e",
                                  n,k,q_n_k[n][k]);
        */
        }
    /*
    printf("\n\n");
    */
    }

/*********************************************************/
```

```
/*      Compute the number of slots required by avera-  */
/*   ging over all possible values of "k" the number    */
/*   copies generated by our target input port. This    */
/*  number of copies follows a MODIFIED-BINOMIAL dist.  */
/*******************************************************/

q_n[0] = 0.0;
for ( n = 1; n <= NMAX; n++ )
    {
    q_n[n] = 0.0;
   for ( k = 1; k <= KMAX; k++ )
        {
        /* compute the prob of k copies generated       */

        bin_fact = binomi(k,bin_fact,NO);
        pro_fact = power(pee,k);
        pro_fact = pro_fact / (1.0 - power ((1.0-pee),NO));
        pro_fact = pro_fact * power((1.0-pee),NO-k);
        p_k = bin_fact * pro_fact;

        /* compute the prob of n slots required         */

        q_n[n] = q_n[n] + (p_k * q_n_k[n][k]);
        }
    /*
    printf ("\n\n  n = %2d   q_n = %e",n,q_n[n]);
    */
    }

/*******************************************************/
/* Compute  the average service time and the average    */
/* delay                                                */
/*******************************************************/

ser_time = 0.0;
ser_square = 0.0;
for ( n = 0; n <= NMAX; n++)
    {
    ser_time = ser_time + ( n * q_n[n] );
    ser_square = ser_square + ((n * n) * q_n[n] );
    }
/*
printf ("\n\n average service time = %e",ser_time);
printf ("\n\n");
printf ("\n\n average (service time)**2 = %e",ser_square);
printf ("\n\n");
*/
lamda = (rho / ser_time);
/*
printf ("\n\n lamda = %e msg/slot",lamda);
printf ("\n\n");
*/
```

```
        delay = ser_time + ((lamda*ser_square)/(2.0*(1.0-rho)));
printf ("\n\n lamda =%e service =%e delay =%e",lamda,ser_time,delay);
        printf("\n\n");
        }
```

```
/*****************************************************************/
/*                                                               */
/*      Fonction power() qui calcule le resultat du parametre    */
/*      fact eleve a la puissance l.                             */
/*                                                               */
/*      PARAMETRES:     fact    parametre a elever a la puiss.   */
/*                      l       exposant                         */
/*****************************************************************/

double  power ( fact,l )

/***************/
/* PARAMETRES */
/***************/

double          fact;
short           l;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          result;
        short           i;

        /************/
        /* CORPS    */
        /************/

        result = 1.0;
        if ( l != 0 )
            {

            for ( i=1; i<=l; i++ )
                {
                result = result * fact;
                }
            }
        /*
        printf ("\n\n entry = %e power/result = %e",fact,result);
        */
        return (result);
        }


/*****************************************************************/
/*                                                               */
/*      Fonction product() utilisee pour calculer un produit     */
/*      avec variable allant d'une valeur minimal a une valeur   */
/*      maximal.                                                 */
```

```
/*                                                                    */
/*      PARAMETRES:     fact    facteur a multiplier                  */
/*                                                                    */
/*                      l       indice maximal du produit             */
/*                                                                    */
/**********************************************************************/

double  product ( fact,l )

/***************/
/* PARAMETRES */
/***************/

short           fact;
short           l;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          result;
        short           i;


        /**************/
        /* CORPS     */
        /**************/

        result = 1.0;
        for ( i = 1; i <= l; i++ )
            {
            result = result * ( fact + i );
            }
        /*
        printf ("\n\n product/result = %e",result);
        */
        return (result);
        }

/**********************************************************************/
/*                                                                    */
/*      Fonction binomi() utilisee pour calculer un coeficient        */
/*      binomial quelconque .                                         */
/*                                                                    */
/*                                                                    */
/*      PARAMETRES:     ind_act = indice actuel du coeficient         */
/*                      pre_fact= coeficient calculer precedem-       */
/*                                ment avec l'indice (ind_act-1)      */
/*                      ind_max = indice maximal                      */
/*                                                                    */
```

```
/*****************************************************************/

double  binomi (ind_act,pre_fact,ind_max)


/***************/
/* PARAMETRES */
/***************/

short           ind_act;
double          pre_fact;
short           ind_max;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          bin_fact;


        /**************/
        /* CORPS     */
        /**************/


        /* compute the binomial factor                  */

        if ( ind_act == 0)
            {
            bin_fact = 1.0;
            }
        else
            {
   bin_fact = pre_fact * ((double)ind_max - (double)ind_act +1.0)
                                                /(double)ind_act;
            }
        return (bin_fact);
        }

/*****************************************************************/
/*                                                               */
/*      Fonction convo() used for computing the n-fold           */
/*      convolution of a fonction                                */
/*                                                               */
/*****************************************************************/
convo ( ind_max,adr_fct,degree)

/***************/
/* PARAMETRES */
/***************/
```

```
short   ind_max;
double  *adr_fct;
short   degree;


    {

    /***************/
    /* VARIABLES   */
    /***************/

    short               i,j,k,pre_ind,ind_act,ind_pre;


    /*************/
    /* CORPS     */
    /*************/

    /* Get a working copy of the fonction and initialise the    */
    /* convolution first array to the function values           */

    for ( j = 0; j <= (ind_max*degree); j++ )
        {
        if ( j <= ind_max )
            {
            conv[0][j] = *adr_fct;
            fonct[j] = *adr_fct;
            adr_fct += 1;
            }
        else
            {
            conv[0][j] = 0.0;
            fonct[j] = 0.0;
            }
        }

    /* Compute the n-fold convolution where n = degree             */

    conv_ave = 0.0;
    conv_tot = 0.0;
    conv_var = 0.0;

    for ( i = 1; i <= (degree-1); i++ )
        {
        ind_act = i % 2;
        ind_pre = (i+1) % 2;
        for ( j = 0; j <= (ind_max*degree); j++ )
            {
            conv[ind_act][j] = 0.0;
            for ( k = 0; k <= j; k++ )
                {
```

```
                conv[ind_act][j] = conv[ind_act][j] +
                        (conv[ind_pre][k] * fonct[j-k]);
                }
            if ( i == (degree-1) )
                {
                conv_ave = conv_ave + (conv[ind_act][j] * (double)j);
            conv_var = conv_var + (conv[ind_act][j] * (double)(j*j));
                conv_tot = conv_tot + conv[ind_act][j];
                }
            }
        conv_var = conv_var - ( conv_ave * conv_ave );
        }
return;
    }
```

```
/*****************************************************************/
/*                                                               */
/*      MULTI4.C :                                               */
/*                                                               */
/*      Program to compute the probabilities related to the      */
/*      MULTICAST SWITCH analysis. The switch has NI inputs      */
/*      and NO outputs. Packets arrive at the inputs at a        */
/*      POISSON rate = L packets per second. Each packet         */
/*      generates a MODIFIED-BINOMIAL number of copies. The      */
/*      copies from all input ports are pooled together and      */
/*      distributed to the output ports according to a           */
/*      MULTINOMIAL distribution. The evaluation of the traffic  */
/*      interfering with our targeted input port is done by      */
/*      computing the distribution of the residual number of     */
/*      copies coming from each interfering input port and       */
/*      then by performing the NI-1 fold convolution to get the  */
/*      total conflicting traffic.                               */
/*      Time is sliced in slots and the service time corespond   */
/*      to the number of slots required to transmit all the      */
/*      copies generated .                                       */
/*      Conflicts at outputs are resolved by random selection.   */
/*      Copies that have not gain access to an output port have  */
/*      to be retransmit. We also compute the cases where mul-   */
/*      tiple output ports are grouped at one output trunk.      */
/*      The program computes the probabilities for different     */
/*      values of PEE = prob. of sending a copy to an output     */
/*      port and for different values of RHO = load to the       */
/*      system.                                                  */
/*                                                               */
/*****************************************************************/

#include        <size.h>

static  double  conv[2][MTMAX+1];       /* convolution array       */
static  double  fonct[MTMAX+1];

static double   conv_ave;
static double   conv_var;
static double   conv_tot;

main()

{

/***************/
/* VARIABLES   */
/***************/

double  rho,pee;
short   i,j;

/*************/
```

```
/*  CORE     */
/*************/

pee = PEEMIN;
while ( pee <= PEEMAX )
    {
    rho = RHOMIN;
    while ( rho <= RHOMAX )
        {
        compu(rho,pee);
        if (rho < .95)
            {
            rho = rho + RHOINC;
            }
        else
            {
            rho = rho + (RHOINC/5.0);
            }
        }
    pee = pee + PEEINC;
    }
}

/*****************************************************************/
/*                                                               */
/*      Fonction to compute the service time and delay of the    */
/*      multicast switch.                                        */
/*****************************************************************/

compu(rho,pee)


/*************/
/* PARAMETERS*/
/*************/

double  rho,pee;

        {

        /*************/
        /* variables */
        /*************/

        short   i,l,j,k,n,rt,expo,ind_res;
        double  para_lng,delay,ser_square,lamda,sum2_fact;
        double  rho_fact,pro_fact,ser_time,bi[KMAX+1],re[KMAX+1];
        double  power(),product(),binomi(),sum_fact,bin2_fact;
        double  bin_fact,even_sum,odd_sum,glo_fact,prob_fact;
        double  p_l_mo[KMAX+1],p_o_l[KMAX+1][KMAX+1];
        double  p_l_k[KMAX+1][KMAX+1],q_n_k[NMAX+1][KMAX+1];
        double  p_k,q_n[NMAX+1],pl_mo_rt[KMAX+1][MTMAX+1];
```

```
    double  bi_ave,bi_var,bi_tot,re_ave,re_var,re_tot;

    /*************/
    /* corps     */
    /*************/

    /*********************************************************/
    /* Print  the context information of te calculation      */
    /*********************************************************/

    printf("\n\n NI=%2d NO=%2d KMAX=%2d NMAX=%2d RHO=%e PEE=%e",
                              NI,NO,KMAX,NMAX,rho,pee);
    printf("\n\n");


    /*********************************************************/
    /* compute for all possible values of "rt"               */
    /*          PROB [first l throught or more|rt]     */
    /*********************************************************/

for ( rt = 0; rt <= MTMAX; rt++ )
    {
    for ( l = 0; l <= KMAX; l++ )
        {
       pl_mo_rt[l][rt] = 0.0;

        /* compute the global factor        */

        glo_fact = (double)S * (double)NO;
        glo_fact = power (glo_fact,l );
        glo_fact = glo_fact / product (rt,l );

        /* compute the summation            */

        expo = rt + l;
        even_sum = 0.0;
        odd_sum = 0.0;

        for ( i = 0; i <= l; i++ )

            {
            /* compute the binomial factor  */

            bin_fact = binomi(i,bin_fact,l);

            /* compute one of the partial summation with    */
            /* the coresponding probability factor          */

            if ( i % 2 == 0 )
                {
```

```
                /* even summation           */

                prob_fact = 1.0 - ((double)i/(double)NO);
                prob_fact = power ( prob_fact,expo );
                even_sum = even_sum + (bin_fact * prob_fact);
                }
            else
                {
                /* odd summation            */

                prob_fact = 1.0 - ((double)i/(double)NO);
                prob_fact = power ( prob_fact,expo );
                odd_sum = odd_sum + (bin_fact * prob_fact);
                }

            }

        pl_mo_rt[l][rt] = glo_fact * ( even_sum - odd_sum );

        /*
        printf ("\n\n  l = %2d  rt =  %2d  P(l or more|rt) = %e",
                                    l,rt,pl_mo_rt[l][rt]);
        printf ("\n\n");
        */
        }
    }

/*************************************************************/
/* compute the PROB [ first l or more ] by averaging over    */
/* all possible values of "rt" the number of interfering     */
/* copies from the NI-1 other input ports                    */
/*************************************************************/

/* Compute the BINOMIAL distribution of the number of copies */
/* generated by one input. We use the BINOMIAL distribution  */
/* instead of the MODIFIED-BINOMIAL since the RESIDUAL LIFE   */
/* of both distribution is the same.                         */

bi_ave = 0.0;
bi_tot = 0.0;
bi_var = 0.0;
for ( i =0; i <= KMAX; i++ )
    {
    bin_fact = binomi (i,bin_fact,(short)KMAX);
    pro_fact = power (pee,i);
    para_lng = 1.0 - pee;
    bi[i] = bin_fact * pro_fact * power (para_lng,(short)KMAX-i);
    bi_ave = bi_ave + bi[i] * (double)i;
    bi_var = bi_var + bi[i] * (double)(i*i);
    bi_tot = bi_tot + bi[i];
    }
bi_var = bi_var - ( bi_ave * bi_ave );
```

```
#ifdef TEST

    /* print the results                                                */

    for ( i = 0; i <= KMAX; i++ )
        {
        printf ("\n\n bi(%d) = %e",i,bi[i]);
        }
    printf ("\n\n bi_ave = %e bi_var = %e bi_tot = %e",
                                          bi_ave,bi_var,bi_tot);

#endif

    /* Compute the residual number of conflicting copies from an    */
    /* interfering input port as seen by our target input port      */

    re[0] = 0.0 + (1.0 - rho);
    re_ave = 0.0;
    re_tot = 0.0;
    re_var = 0.0;
    for ( i = 1; i <= KMAX; i++ )
        {
        re[i] = 0.0;
        for ( j = 0; j <= i-1; j++ )
            {
            re[i] = re[i] + bi[j];
            }
        re[i] = rho * (1.0 - re[i]) / ((double)KMAX * pee);
        re_ave = re_ave + re[i] * (double)i;
        re_var = re_var + re[i] * (double)(i*i);
        re_tot = re_tot + re[i];
        }
    re_var = re_var - ( re_ave * re_ave );

#ifdef TEST

    /* print the results                                                */

    for ( i = 0; i <= KMAX; i++ )
        {
        printf ("\n\n re(%d) = %e",i,re[i]);
        }
    printf ("\n\n re_ave = %e re_var = %e re_tot = %e",
                                          re_ave,re_var,re_tot);

#endif

    /* Compute the convolution to get the distribution of the total*/
    /* number of conflicting copies from NI-1 input ports.          */

    convo (KMAX,&re[0],NI-1);
```

```
    ind_res = (NI) % 2;       /* indicates which array holds results*/
#ifdef TEST
    /* print the results                                              */
    for ( i = 0; i <= MTMAX; i++ )
        {
        printf ("\n\n conv(%d) = %e",i,conv[ind_res][i]);
        }
    printf ("\n\n conv_ave = %e conv_var = %e conv_tot = %e",
                                        conv_ave,conv_var,conv_tot);
#endif
        /* compute the PROB [first l or more]            */
            for ( l = 0; l <= KMAX; l++ )
                {
               p_l_mo[l] = 0.0;
                for ( rt = 0; rt <= MTMAX; rt++ )
                    {
                    p_l_mo[l] = p_l_mo[l] +
                                  pl_mo_rt[l][rt] * conv[ind_res][rt];
                   }
                /*
               printf("\n\n l = %2d k = %2d p[first l or more] = %e",
                                        l,p_l_mo[l]);
                printf ("\n\n);
                */
                }
            /*****************************************************/
            /* compute PROB [ first "l" throught|k]              */
            /*****************************************************/
        for ( k = 0; k <= KMAX; k++ )
            {
            for ( l=0; l <= k; l++ )
                {
                /* compute the summation                         */
                even_sum = 0.0;
                odd_sum = 0.0;
                for ( j=0; j <= k-l; j++ )
                    {
                    /* get the binomial factor                   */
                    bin_fact = binomi(j,bin_fact,k-l);
```

```
            /* compute the even/odd summations          */

            if ( j % 2 == 0 )
                {
                even_sum = even_sum +
                        (bin_fact * p_l_mo[l+j]);
                }
            else
                {
                odd_sum = odd_sum +
                        (bin_fact * p_l_mo[l+j]);
                }

            }

        p_o_l[l][k] = even_sum - odd_sum;
        /*
        printf ("\n\n l = %2d  k = %2d  p[first l|k] = %e",
                                        l,k,p_o_l[l][k]);
        printf("\n\n");
        */
        }

    /*******************************************************/
    /* compute PROB [ l (all possible sets) |k ]           */
    /*******************************************************/

    for (l=0; l<=k; l++ )
        {

        /* get the binomial factor                         */

        bin_fact = binomi(l,bin_fact,k);

        p_l_k[l][k] = bin_fact * p_o_l[l][k];
        /*
      printf ("\n\n l = %2d  k = %2d  p[l all sets|k] = %e",
                                        l,k,p_l_k[l][k]);
        */
        }
   }
/*
printf("\n\n");
*/

/*******************************************************/
/* Compute the probability of the number of slots      */
/* required to transmit the k packets condition on k   */
/*******************************************************/


for ( n = 0; n <= NMAX; n++ )
```

```
    {
    for ( k = 0; k <= KMAX; k++ )
        {
        if ( k == 0 && n == 0 )
            {
           q_n_k[n][k] = 1.0;
           }
        else if ( n == 0 && k > 0 || k == 0 && n > 0 )
            {
            q_n_k[n][k] = 0.0;
            }
        else if ( n == 1 )
            {
            q_n_k[n][k] = p_l_k[k][k];
            }
        else
            {
            q_n_k[n][k] = 0.0;
            for ( l = 0; l <= k; l++ )
                {
                q_n_k[n][k] = q_n_k[n][k] +
                              (p_l_k[l][k] * q_n_k[n-1][k-l]);

                }
            }
        /*
        printf("\n\n  n = %2d  k = %2d  q_n_k = %e",
                                  n,k,q_n_k[n][k]);
        */
        }
    /*
    printf("\n\n");
    */
    }


/*********************************************************/
/*      Compute the number of slots required by avera-   */
/*   ging over all possible values of "k" the number     */
/*   copies generated by our target input port. This     */
/*  number of copies follows a MODIFIED-BINOMIAL dist.   */
/*********************************************************/

q_n[0] = 0.0;
for ( n = 1; n <= NMAX; n++ )
    {
    q_n[n] = 0.0;
   for ( k = 1; k <= KMAX; k++ )
        {
        /* compute the prob of k copies generated       */

        bin_fact = binomi(k,bin_fact,KMAX);
```

```
            pro_fact = power(pee,k);
            pro_fact = pro_fact / (1.0 - power ((1.0-pee),KMAX));
            pro_fact = pro_fact * power((1.0-pee),KMAX-k);
            p_k = bin_fact * pro_fact;

            /* compute the prob of n slots required          */

            q_n[n] = q_n[n] + (p_k * q_n_k[n][k]);
            }
        /*
        printf ("\n\n  n = %2d   q_n = %e",n,q_n[n]);
        */
        }

    /*********************************************************/
    /* Compute  the average service time and the average     */
    /* delay                                                 */
    /*********************************************************/

    ser_time = 0.0;
    ser_square = 0.0;
    for ( n = 0; n <= NMAX; n++)
        {
        ser_time = ser_time + ( n * q_n[n] );
        ser_square = ser_square + ((n * n) * q_n[n] );
        }
    /*
    printf ("\n\n average service time = %e",ser_time);
    printf ("\n\n");
    printf ("\n\n average (service time)**2 = %e",ser_square);
    printf ("\n\n");
    */
    lamda = (rho / ser_time);
    /*
    printf ("\n\n lamda = %e msg/slot",lamda);
    printf ("\n\n");
    */
    delay = ser_time + ((lamda*ser_square)/(2.0*(1.0-rho)));
printf ("\n\n lamda =%e service =%e delay =%e",lamda,ser_time,delay);
    printf("\n\n");
    }
```

```
/*****************************************************************/
/*                                                               */
/*      Fonction power() qui calcule le resultat du parametre    */
/*      fact eleve a la puissance l.                             */
/*                                                               */
/*      PARAMETRES:     fact    parametre a elever a la puiss.   */
/*                      l       exposant                         */
/*****************************************************************/

double  power ( fact,l )

/***************/
/* PARAMETRES */
/***************/

double          fact;
short           l;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          result;
        short           i;

        /************/
        /* CORPS   */
        /************/

        result = 1.0;
        if ( l != 0 )
            {

            for ( i=1; i<=l; i++ )
                {
                result = result * fact;
                }
            }
        /*
        printf ("\n\n entry = %e power/result = %e",fact,result);
        */
        return (result);
        }


/*****************************************************************/
/*                                                               */
/*      Fonction product() utilisee pour calculer un produit     */
/*      avec variable allant d'une valeur minimal a une valeur   */
/*      maximal.                                                 */
```

```
/*                                                                    */
/*      PARAMETRES:     fact    facteur a multiplier                  */
/*                                                                    */
/*                      l       indice maximal du produit             */
/*                                                                    */
/**********************************************************************/

double  product ( fact,l )

/***************/
/* PARAMETRES */
/***************/

short           fact;
short           l;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          result;
        short           i;


        /*************/
        /* CORPS     */
        /*************/

        result = 1.0;
        for ( i = 1; i <= l; i++ )
            {
            result = result * ( fact + i );
            }
        /*
        printf ("\n\n product/result = %e",result);
        */
        return (result);
        }

/**********************************************************************/
/*                                                                    */
/*      Fonction binomi() utilisee pour calculer un coeficient        */
/*      binomial quelconque .                                         */
/*                                                                    */
/*                                                                    */
/*      PARAMETRES:     ind_act = indice actuel du coeficient         */
/*                      pre_fact= coeficient calculer precedem-       */
/*                                ment avec l'indice (ind_act-1)      */
/*                      ind_max = indice maximal                      */
/*                                                                    */
```

```
/*****************************************************************/

double  binomi (ind_act,pre_fact,ind_max)


/***************/
/* PARAMETRES */
/***************/

short           ind_act;
double          pre_fact;
short           ind_max;

        {

        /***************/
        /* VARIABLES  */
        /***************/

        double          bin_fact;


        /*************/
        /* CORPS     */
        /*************/


        /* compute the binomial factor                    */

        if ( ind_act == 0)
            {
            bin_fact = 1.0;
            }
        else
            {
            bin_fact = pre_fact * ((double)ind_max - (double)ind_act +
                                                   /(double)ind_act;
            }
        return (bin_fact);
        }

/*****************************************************************/
/*                                                               */
/*      Fonction convo() used for computing the n-fold           */
/*      convolution of a fonction                                */
/*                                                               */
/*****************************************************************/
convo ( ind_max,adr_fct,degree)

/***************/
/* PARAMETRES */
```

```
/***************/

short   ind_max;
double  *adr_fct;
short   degree;


    {

    /***************/
    /* VARIABLES   */
    /***************/

    short               i,j,k,pre_ind,ind_act,ind_pre;


    /**************/
    /* CORPS      */
    /**************/

    /* Get a working copy of the fonction and initialise the    */
    /* convolution first array to the function values           */

    for ( j = 0; j <= (ind_max*degree); j++ )
        {
        if ( j <= ind_max )
            {
            conv[0][j] = *adr_fct;
            fonct[j] = *adr_fct;
            adr_fct += 1;
            }
        else
            {
            conv[0][j] = 0.0;
            fonct[j] = 0.0;
            }
        }

    /* Compute the n-fold convolution where n = degree              */

    conv_ave = 0.0;
    conv_tot = 0.0;
    conv_var = 0.0;

    for ( i = 1; i <= (degree-1); i++ )
        {
        ind_act = i % 2;
        ind_pre = (i+1) % 2;
        for ( j = 0; j <= (ind_max*degree); j++ )
            {
            conv[ind_act][j] = 0.0;
            for ( k = 0; k <= j; k++ )
```

```
                {
                conv[ind_act][j] = conv[ind_act][j] +
                        (conv[ind_pre][k] * fonct[j-k]);
                }
            if ( i == (degree-1) )
                {
                conv_ave = conv_ave + (conv[ind_act][j] * (double)j);
            conv_var = conv_var + (conv[ind_act][j] * (double)(j*j));
                conv_tot = conv_tot + conv[ind_act][j];
                }
            }
        conv_var = conv_var - ( conv_ave * conv_ave );
        }
return;
    }
```

```
/*****************************************************************/
/*                        SIZE.H                                 */
/* Fichier entete specifiant les parametres qui definissent      */
/* la grandeur du commutateur analyse                            */
/*                                                               */
/*                                                               */
/*****************************************************************/

/*
#define         TEST    1
*/
#define         NI      16
#define         NO      16
#define         S       2
#define         KMAX    16
#define         NMAX    256
#define         MTMAX   (NI-1)*NO
#define         RHOMAX  1.0
#define         PEEMAX  1.0
#define         RHOMIN  0.1
#define         PEEMIN  0.1
#define         RHOINC  0.05
#define         PEEINC  0.1
```

APPENDIX E

Simulation programs

This appendix contains the following programs and "header" files.

SIM1.C : Simulation program of the unicast switch. This program is used for validation purposes.

SIM1.H : "Header" file containing the data structures definitions for SIM1.C.

SIM3.C : Simulation program for the multicast packet switch. It also implememts the grouped output ports.

SIM3.H : "Header" file containing the data structures definitions for SIM3.C.

SIZE.H : "Header" file defining the switch parameters.

```
/*****************************************************************/
/*                                                               */
/*      sim1.c:                                                  */
/*                                                               */
/*      Simulation programme for the evaluation of performance of */
/*      a multicast switch                                       */
/*                                                               */
/*****************************************************************/

#include        "size.h"
#include        "sim1.h"

/* Tableau contenant les queues pour chaque port d'entree        */

static  short   tran_lim = IN_SIZE - 1;
static  struct  IN_QUEU tab_inqueu[NI] = 0;

/* Tableau contenant les queues imaginaires pour chaque port de */
/* sorties.                                                      */

static struct   OU_QUEU tab_ouqueu[NO] = 0;

/* Tableaux contenant les "SEEDS" pour la generation des nombres */
/* aleatoires                                                    */

static unsigned in_seed[NI] = 0;
static unsigned ou_seed[NO] = 0;

/* Horloge qui tient le temps en nombre de slot                  */

static long int sys_clock = 0L;

/* Performance data report                                       */

static  struct  PERF_BOX perf_box;

/* Error counter                                                 */

static  short   err_cnt = 0;
```

```
/*****************************************************************/
/*                                                               */
/*      main:                                                    */
/*                                                               */
/*      Main program that calls the simulation routine for       */
/* different values of "lambda" and "pee". Each head-of-queue    */
/* packet generates a random number of copies by first choosing  */
/* its primary output port according to a uniform distribution and*/
/* the other output ports by performing independent Bernouilli   */
/* trials with probability P. A unicast switch is simulated by   */
/* setting P = 0                                                 */
/*****************************************************************/
main ()

    {

    /****************/
    /*  variables   */
    /****************/

    short       run_ind,num_run,i;
    long int    max_time;
    double      pee,lambda,rho;

    /****************/
    /*  program     */
    /****************/

    rho = 0.0;
    max_time = SIM_TIME;
    pee = PEEMIN;
    while ( pee <= PEEMAX )
        {
        lambda = LAMBMIN;
        while ( lambda <= LAMBMAX )
            {
            if ( rho >= MAX_LOAD )
                {
                break;
                }

            /*******************************************************/
            /* Print the simulation parameters                     */
            /*******************************************************/

            printf ("\n\n");
printf ("\n\n simulation para :max_time = %d lambda = %e pee = %e",
                                        max_time,lambda,pee);

     /*********************************************************/
     /* Call the simulator routine for all batch run  and ...    */
     /* print the results                                        */
```

```
    /*************************************************************/

            for ( run_ind = 0; run_ind <= NUM_RUN - 1; run_ind++ )
                {

                simuli (max_time,run_ind,lambda,pee);
/*
                printf ("\n\n batch = %d ser_time = %e delay = %e",
                run_ind,perf_box.batch[run_ind].ave_serv,
                perf_box.batch[run_ind].ave_delay);
*/
                num_run = run_ind + 1;
                }

            /* get the average result of all batch run            */

            sim_stat (FCT_RES,num_run,lambda,0);
   printf ("\n\n Average results: service = %e delay = %e rho = %e",
                      perf_box.service,perf_box.delay,perf_box.rho);

#ifdef  SIM_TEST
            for ( i = 0; i <= MAX_PAQ - 1; i += 4 )
                {
printf ("\n\n distribution: D(%d)=%e  D(%d)=%e  D(%d)=%e D(%d)=%e",
                i,perf_box.dist[i],i+1,perf_box.dist[i+1],i+2,
                perf_box.dist[i+2],i+3,perf_box.dist[i+3]);
                }
         printf ("\n\n distribution: D(%d)=%e",i,perf_box.dist[i]);

            printf ("\n\n distr. average = %e  total prob. = %e",
                                perf_box.dis_ave,perf_box.tot_prob);
#endif
            rho = perf_box.rho;
            if (rho > .85)
                {
                lambda = lambda + (LAMBINC/5.0);
                }
            else
                {
                lambda = lambda + LAMBINC;
                }
            }
        pee = pee + PEEINC;
        rho = 0.0;
        }
    }
```

```
/*****************************************************************/
/*                                                               */
/*      simuli:                                                  */
/*                                                               */
/*      Simulation routine called by the main program. It simulate*/
/*      the multicast switch for a specific value of:            */
/*                                                               */
/*                max_time = the lenght of the simulation (slots)*/
/*                run_ind = the run number                       */
/*                lambda = average packet arrival rate           */
/*                pee = the multicast probability                */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              long int  max_time                               */
/*              short     run_ind                                */
/*              double    lambda                                 */
/*              double    pee                                    */
/*                                                               */
/*      OUTPUT VALUE:                                            */
/*                                                               */
/*                                                               */
/*****************************************************************/

simuli (max_time,run_ind,lambda,pee)

/****************/
/*  parameter   */
/****************/

long int        max_time;
short           run_ind;
double          lambda;
double          pee;

    {

    /****************/
    /*  variables   */
    /****************/

    short       i,j,k,ind_in,ind_ou;
    short       num_arr,err_code;
    short       poisson(),enter_queu(),exit_queu();
    float       mth$random(),ran_num,prob_int,cum_prob;


    /****************/
    /*  program     */
    /****************/

    /*****************************************************************/
```

```
/* Initialise the clock and the seeds for RANDOM NUMBERS ...  */
/* ...generation,the statitics and the input and output queues*/
/************************************************************/

err_cnt = 0;
ran_num = 0.0;

sim_stat (FCT_INI,run_ind,0.0,0);

for (i = 0; i <= NI-1; i++)
    {
    in_seed[i] = (unsigned)(seed1[run_ind]*(i+1));
    tab_inqueu[i].pt_in = &tab_inqueu[i].li_tran[0];
    tab_inqueu[i].pt_ou = &tab_inqueu[i].li_tran[0];
    tab_inqueu[i].num_tran = 0;
    }

for (j = 0; j <= NO-1; j++)
    {
    ou_seed[j] = (unsigned)(seed2[run_ind]*(j+1));
    tab_ouqueu[j].num_cop = 0;
    tab_ouqueu[j].busy_cnt = 0;
    for ( k = 0; k <= NI-1; k++ )
        {
        tab_ouqueu[j].ind_cop[k] = 0;
        }
    }



/************************************************************/
/* Initialise the clock and simulate for the desired number of*/
/* slots                                                      */
/************************************************************/

sys_clock = 0L;
while ( sys_clock <= max_time )
    {

    /************************************************************/
    /* Generate packets arrivals at input port(POISSON)         */
    /************************************************************/

    for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
        {
        num_arr = poisson(&in_seed[ind_in],lambda);
        if ( num_arr > 0 )
            {
            /* packet arrival(s) at input port -> update queue*/

            for ( i = 1; i <= num_arr; i++ )
                {
```

```
                err_code = enter_queu(ind_in);
                if (err_code != SUCCES )
                    {
                    /* queu is full -> consider as an error   */

                    sim_err ( err_code );
                    break;
                    }
                }
            }
        }

    /*************************************************************/
    /* Generate traffic distribution if new HOL(head of line)  */
    /* packet                                                    */
    /*************************************************************/

    for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
        {

        if (tab_inqueu[ind_in].pt_ou->num_cop == -1 &&
                                tab_inqueu[ind_in].num_tran > 0 )
            {
            /* New HOL packet -> update input queue information*/
            /* ... and generate the copies                     */

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d serv_beg = %d",ind_in,
                            tab_inqueu[ind_in].pt_ou,sys_clock);
#endif

            tab_inqueu[ind_in].pt_ou->ser_beg = sys_clock;
            tab_inqueu[ind_in].pt_ou->num_cop = 0;


            /* Choose the primary output port                  */

            ran_num = mth$random(&in_seed[ind_in]);
            prob_int = 1.0 / (float) NO;
            cum_prob = prob_int;

            for ( ind_ou = 0; ind_ou <= NO-1; ind_ou++ )
                {
                if ( ran_num < cum_prob )
                    {
                    /* copy generated for that output -> update*/
                    /* ... output and input queues             */

                    tab_inqueu[ind_in].pt_ou->pri_out = ind_ou;
                    tab_inqueu[ind_in].pt_ou->num_cop += 1;
                    tab_ouqueu[ind_ou].num_cop += 1;
```

```
                    tab_ouqueu[ind_ou].ind_cop[ind_in] = 1;
                    break;
                    }
                else
                    {
                    /* Try the next output port -> change      */
                    /* interval                                */

                    cum_prob = cum_prob + prob_int;
                    }
                }

            /* Choose the secondary output ports               */

            for ( ind_ou = 0; ind_ou <= NO-1; ind_ou++ )
                {
                ran_num = mth$random(&in_seed[ind_in]);
                if ( ran_num < pee &&
                    tab_inqueu[ind_in].pt_ou->pri_out != ind_ou)
                    {
                    /* copy generated for that output -> update*/
                    /* ... output and input queues              */

                    tab_inqueu[ind_in].pt_ou->num_cop += 1;
                    tab_ouqueu[ind_ou].num_cop += 1;
                    tab_ouqueu[ind_ou].ind_cop[ind_in] = 1;
                    }
                }

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d num_cop = %d",ind_in,
        tab_inqueu[ind_in].pt_ou,tab_inqueu[ind_in].pt_ou->num_cop);
#endif

                }
            }

        /*********************************************************/
        /* Contention resolution at all output ports             */
        /*********************************************************/

        /* MODIF: Evaluate traffic distribution at output        */

        sim_stat (FCT_DIS,run_ind,lambda,0);

        /* END MODIF                                             */

        for ( ind_ou = 0; ind_ou <= NO-1; ind_ou++ )
            {
            /* determine the probability intervalles size depending*/
            /* ... on the number of copies in contention and obtain*/
```

```
        /* ... a random number                                          */

        if ( tab_ouqueu[ind_ou].num_cop != 0 )
             {
             /* Output queue not empty                                  */

             prob_int = 1.0/(double)tab_ouqueu[ind_ou].num_cop;
             ran_num = mth$random(&ou_seed[ind_ou]);
             cum_prob = prob_int;
             tab_ouqueu[ind_ou].busy_cnt +=1;

             /* Scan for copies from particular input port      */

             for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
                  {
                  if ( tab_ouqueu[ind_ou].ind_cop[ind_in] == 1 )
                       {
                       /* Found a copy from input port "ind_in"  */

                       if ( ran_num < cum_prob )
                            {
                            /* This copy is chosen "lucky" ->update*/
                            /* ...output and input queues and exit */

                            tab_ouqueu[ind_ou].ind_cop[ind_in] = 0;
                            tab_ouqueu[ind_ou].num_cop -=1;
                            tab_inqueu[ind_in].pt_ou->num_cop -=1;
                            break;
                            }
                       else
                            {
                           /* update the probability valid interval.   */
                           /*...and look again for the lucky input port*/

                            cum_prob = cum_prob + prob_int;
                            }
                       }
                  }

#ifdef  SIM_DEBUG

printf ("\n\n ou_queu = %d num_cop = %d",
                                  ind_ou,tab_ouqueu[ind_ou].num_cop);

#endif

                  }
             }

        /*************************************************************/
        /* Update system clock                                       */
        /*************************************************************/
```

```
    sys_clock += 1;

    /*************************************************************/
    /* Test for service completion at input queues and cumulate*/
    /* ... the statistics if nececessary.                        */
    /*************************************************************/

    for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
        {
        if (tab_inqueu[ind_in].num_tran > 0 &&
                        tab_inqueu[ind_in].pt_ou->num_cop == 0 )
            {
       /* service completed -> update input queue,cumulate ... */
       /* the statistics and update transaction output pointer */

            exit_queu( ind_in,run_ind );
            }
        }
    }

/*************************************************************/
/* Output the simulation results for the received values of ..*/
/* ... "lambda" and "pee"                                    */
/*************************************************************/

sim_stat (FCT_REP,run_ind,lambda,0);
return;
}
```

```
/*****************************************************************/
/*                                                               */
/*      poisson:                                                 */
/*                                                               */
/*          Routine that generates packet arrivals according to  */
/*          a POISSON distribution. The method use is derived    */
/*          from the Von-Neumann algorithm. The algorithm assum  */
/*        an arrival rate equal to 1 and a time interval equal   */
/*        to "lambda". But fortunately this is the same as having */
/*        a time interval equal to 1 and an average arrival rate */
/*        equal to "lambda".                                     */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*        - address of the seed to supply to the random number   */
/*          generator.                                           */
/*                                                               */
/*        - average packet arrival rate (lambda)                 */
/*                                                               */
/*      RETURN VALUE:                                            */
/*                                                               */
/*        - number of packet arrivals                            */
/*                                                               */
/*****************************************************************/
short poisson (adr_seed,lambda)

/****************/
/*  parameters  */
/****************/

unsigned int    *adr_seed;
double          lambda;

    {

    /****************/
    /*  variables   */
    /****************/

    short       i,num_arr;
    float       mth$random(),mth$exp(),ran_num,expo;
    double      pro_num,mlambda;

    /****************/
    /*  program     */
    /****************/


    /*****************************************************************/
    /* Get the exponential ( prob of no arrival) for the received .*/
    /* ... value of lambda                                           */
    /*****************************************************************/
```

```
    mlambda = -lambda;
    expo = mth$exp(&mlambda);

    /*****************************************************************/
    /* Determine the number of arrivals                              */
    /*****************************************************************/

    pro_num = 1.0;
    for ( i = 1; i <= MAX_ARR; i++ )
        {
        ran_num = mth$random(adr_seed);
        pro_num = pro_num * ran_num;
        if ( pro_num < expo )
            {
            /* number of arrivals is determine                       */

            num_arr = i-1;
            break;
            }
        }
    return(num_arr);
    }

/*****************************************************************/
/*                                                               */
/*      enter_queu:                                              */
/*                                                               */
/*              Routine that takes care of managing the queue for  */
/*              packets arrivals.                                */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              - input queu indicator                           */
/*                                                               */
/*****************************************************************/
short enter_queu(ind_in)

/****************/
/*  parameters  */
/****************/

short ind_in;

    {

    /****************/
    /*  variables   */
    /****************/

    short       err_code;
```

```
    /*****************/
    /*  program      */
    /*****************/


    if ( tab_inqueu[ind_in].num_tran == IN_SIZE )
        {
        /* Input queue is full                              */

        err_code = QUEU_FULL;
        }
    else
        {
        /* Input queue not full -> save entry info. ... */
        /* ... and update queue                             */

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d arr_time = %d",ind_in,
                            tab_inqueu[ind_in].pt_in,sys_clock);
#endif

        tab_inqueu[ind_in].pt_in->arr_time = sys_clock;
        tab_inqueu[ind_in].pt_in->num_cop = -1;  /* New HOL */
        tab_inqueu[ind_in].pt_in->pri_out = -1;
        if ( tab_inqueu[ind_in].pt_in++ ==
                        &tab_inqueu[ind_in].li_tran[tran_lim] )
            {
          tab_inqueu[ind_in].pt_in = &tab_inqueu[ind_in].li_tran[0];
            }
        tab_inqueu[ind_in].num_tran += 1;
        err_code = SUCCES;
        }
return(err_code);
    }


/*****************************************************************/
/*                                                               */
/*      exit_queu:                                               */
/*                                                               */
/*              Routine that takes care of input queu when packets*/
/*              exit.                                            */
/*                                                               */
/*      INPUT PARAMETER:                                         */
/*                                                               */
/*              - input queu indicator                           */
/*              - batch indicator                                */
/*                                                               */
/*****************************************************************/
short exit_queu( ind_in,run_ind )
```

```
/****************/
/*  parameters  */
/****************/

short ind_in;
short run_ind;

    {

    /*****************/
    /*  variables    */
    /*****************/

    short       err_code;

    /*****************/
    /*  program      */
    /*****************/


    tab_inqueu[ind_in].pt_ou->ser_end = sys_clock;

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d ser_end = %d",ind_in,
        tab_inqueu[ind_in].pt_ou,tab_inqueu[ind_in].pt_ou->ser_end);
#endif

    sim_stat (FCT_CUM,run_ind,0.0,tab_inqueu[ind_in].pt_ou );

    if ( tab_inqueu[ind_in].pt_ou++ ==
                        &tab_inqueu[ind_in].li_tran[tran_lim] )
        {
        tab_inqueu[ind_in].pt_ou = &tab_inqueu[ind_in].li_tran[0];
        }
    tab_inqueu[ind_in].num_tran -= 1;
    err_code = SUCCES;
    return(err_code);
    }


/*****************************************************************/
/*                                                               */
/*      sim_stat:                                                */
/*                                                               */
/*        Routine that manage all the performance statistics of  */
/*        the simulation.                                        */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*        - function code                                        */
/*        - pointer to an input transaction structure            */
```

```
/*                                                                    */
/**********************************************************************/
sim_stat (fct_code,run_ind,lambda,adr_tran)

/****************/
/*  parameters  */
/****************/

short   fct_code;
short   run_ind;
double  lambda;
struct  IN_TRAN *adr_tran;

    {

    /****************/
    /*  variables   */
    /****************/

        float   num_run;
        short   i,j,num_cop;


    /****************/
    /*  program     */
    /****************/

    switch ( fct_code )

        {
        case 0:                 /* Initialisation                */

        perf_box.batch[run_ind].num_paq = 0;
        perf_box.batch[run_ind].cum_serv = 0;
        perf_box.batch[run_ind].cum_delay = 0;
        for ( i = 0; i <= MAX_PAQ; i++ )
            {
            perf_box.batch[run_ind].paq_dis[i] = 0.0;
            }
        break;

        case 1:                 /* cumm. the statistics          */

        if ( sys_clock > WARM_UP )
            {
            perf_box.batch[run_ind].num_paq +=1;
            perf_box.batch[run_ind].cum_serv =
                        perf_box.batch[run_ind].cum_serv +
                          (adr_tran->ser_end - adr_tran->ser_beg);
            perf_box.batch[run_ind].cum_delay =
                        perf_box.batch[run_ind].cum_delay +
                          (adr_tran->ser_end - adr_tran->arr_time);
```

```
        }
    break;

    case 2:                 /* report statistics                */

    perf_box.batch[run_ind].ave_serv =
                    (double)perf_box.batch[run_ind].cum_serv /
                    (double)perf_box.batch[run_ind].num_paq;
    perf_box.batch[run_ind].ave_delay =
                    (double)perf_box.batch[run_ind].cum_delay /
                    (double)perf_box.batch[run_ind].num_paq;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.batch[run_ind].paq_dis[i] =
        perf_box.batch[run_ind].paq_dis[i] /
                                (double)(SIM_TIME - WARM_UP);
        }

    break;

    case 3:   /* compute the final result by averaging over    */
              /* all batch results                             */

    num_run = run_ind;          /* convert to float            */
    perf_box.service = 0.0;
    perf_box.delay = 0.0;
    perf_box.rho = 0.0;
    perf_box.dis_ave = 0.0;
    perf_box.tot_prob = 0.0;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.dist[i] = 0.0;
        }

    for ( i = 0; i <= NUM_RUN - 1; i++ )
        {
perf_box.service = perf_box.service + perf_box.batch[i].ave_serv;
  perf_box.delay = perf_box.delay + perf_box.batch[i].ave_delay;
        for ( j = 0; j <= MAX_PAQ; j++ )
            {
            perf_box.dist[j] = perf_box.dist[j] +
                                    perf_box.batch[i].paq_dis[j];
            }
        }
    perf_box.service = perf_box.service / num_run;
    perf_box.delay = perf_box.delay / num_run;
    perf_box.rho = lambda * perf_box.service;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.dist[i] = perf_box.dist[i] / num_run;
        perf_box.dis_ave = perf_box.dis_ave + perf_box.dist[i] *
                                                    (double)i;
```

```
            perf_box.tot_prob = perf_box.tot_prob + perf_box.dist[i];
             }
        break;

        case 4:  /* get data for evaluation of distribution of   */
                 /* number of packets in conflict                */

        if ( sys_clock > WARM_UP )
            {
            num_cop = 0;
            for ( i = 0; i <= NO - 1; i++ )
                {
                num_cop = num_cop + tab_ouqueu[i].num_cop;
                }
            perf_box.batch[run_ind].paq_dis[num_cop] += 1.0;
            }
        break;
        }
    return;
    }


/*****************************************************************/
/*                                                               */
/*      sim_err:                                                 */
/*                                                               */
/*              Error handling routine                           */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              - Error identification code                      */
/*                                                               */
/*****************************************************************/
sim_err (err_code)

/****************/
/*  parameters  */
/****************/

short   err_code;

    {

    /****************/
    /*  variables   */
    /****************/



    /****************/
    /*  program     */
    /****************/
```

```
err_cnt += 1;
if ( err_cnt <= MAX_ERR )
    {
    printf ("\n\n error code = %d",err_code);
    }
return;
}
```

```
/*****************************************************************/
/*                                                               */
/*      sim1.h:                                                  */
/*                                                               */
/*                                                               */
/*      Fichier contenant les variables associees au commutateur */
/*      simmule ainsi que les structures de donnees              */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/* Constant definition                                           */
/*****************************************************************/

/*
#define         SIM_TEST        1
#define         SIM_DEBUG       1
*/
#define         MAX_ERR         2
#define         MAX_RUN         10 /* Maximum number of batch run */
#define         MAX_ARR         16 /* Maximum number of arrival ..*/
                                   /* ... in a slot time         */
#define         MAX_PAQ         NI*NO /* max # packets at output  */

/*****************************************************************/
/* Function codes                                                */
/*****************************************************************/

#define         FCT_INI 0               /* Initialisation          */
#define         FCT_CUM 1               /* Cummulation of statistics */
#define         FCT_REP 2               /* Compute,print perf. report*/
#define         FCT_RES 3               /* Compute final results     */
#define         FCT_DIS 4               /* data for evaluation of #  */
                                        /* packet in conflict        */

/*****************************************************************/
/* Error codes                                                   */
/*****************************************************************/

#define         SUCCES          1
#define         QUEU_FULL       -1

/*****************************************************************/
/* Structures definitions                                        */
/*****************************************************************/

struct  IN_TRAN         /* transaction pour chaque paquet aux ports */
                        /* d'entree                                 */
        {
        long int          arr_time;       /* arrival time          */

        long int          ser_beg;      /* beginning of service time */
```

```
        long int        ser_end;     /* end of service time       */
        short           num_cop;     /* # copies generated        */
        short           pri_out;     /* primary output port       */
        };


struct  IN_QUEU      /* queue circulaire de transactions a chaque  */
                     /* port d'entree                              */
        {
        short   num_tran;
        struct  IN_TRAN *pt_in;
        struct  IN_TRAN *pt_ou;
        struct  IN_TRAN li_tran[IN_SIZE];
        };


struct  OU_QUEU          /* queue de sortie (imaginaire) a chaque  */
                         /* port de sortie                         */
        {
        short            num_cop;     /* # copies in contention    */
        short            busy_cnt;    /* busy count of the queue   */
        short            ind_cop[NI]; /* ind. de copies provenant  */
                                      /* de chaque entree          */
        };

struct STAT_DATA     /* Performance statistics of the system for   */
                     /* each batch run                             */
        {
        long int        num_paq;     /* number of packets served   */
        long int        cum_serv;    /* cummulative service time   */
        long int        cum_delay;   /* cummulative delay          */
        double          ave_serv;    /* average service time       */
        double          ave_delay;   /* average delay              */
        double          paq_dis[MAX_PAQ+1];/* Distrib.of # packets */
                                      /* ... at output ports       */
        };


struct PERF_BOX         /* Mail box for performance results passing*/
        {
        struct STAT_DATA batch [MAX_RUN];
        double          service;
        double          delay;
        double          rho;
        double          dist[MAX_PAQ+1];
        double          dis_ave;
        double          tot_prob;
        };


unsigned        seed1[MAX_RUN] ={
                                 1735467,1671,97631,41,745,
```

```
                                89655987,506463,7,2871,47512809
                                };

unsigned        seed2[MAX_RUN] ={
                                217,41097839,1459361,6543,67,
                                932,46172583,9,523987,90651
                                };
```

```
/*****************************************************************/
/*                                                               */
/*      sim3.c:                                                  */
/*                                                               */
/*      Simulation programme for the evaluation of performance of */
/*      a multicast switch. Each HOL packet generates a modified  */
/*      binomial number of copies. The switch can have multiple   */
/*      output ports at each output trunk                        */
/*                                                               */
/*****************************************************************/

#include        "size.h"
#include        "sim3.h"

/* Tableau contenant les queues pour chaque port d'entree        */

static  short   tran_lim = IN_SIZE - 1;
static  struct  IN_QUEU tab_inqueu[NI] = 0;

/* Tableau contenant les queues imaginaires pour chaque port de  */
/* sortie                                                        */

static struct   OU_QUEU tab_ouqueu[NT] = 0;

/* Tableaux contenant les "SEEDS" pour la generation des nombres */
/* aleatoires                                                    */

static unsigned in_seed[NI] = 0;
static unsigned ou_seed[NT] = 0;

/* Horloge qui tient le temps en nombre de slot                  */

static long int sys_clock = 0L;

/* Performance data report                                       */

static  struct  PERF_BOX perf_box;

/* Error counter                                                 */

static  short   err_cnt = 0;
```

```
/*****************************************************************/
/*                                                               */
/*      main:                                                    */
/*                                                               */
/*      Main program that calls the simulation routine for       */
/*      different values of "lambda" and "pee"                   */
/*                                                               */
/*                                                               */
/*****************************************************************/
main ()

    {

    /****************/
    /*  variables   */
    /****************/

    short       run_ind,num_run,i;
    long int    max_time;
    double      pee,lambda,rho;

    /****************/
    /*  program     */
    /****************/

    rho = 0.0;
    max_time = SIM_TIME;
    pee = PEEMIN;
    while ( pee <= PEEMAX )
        {
        lambda = LAMBMIN;
        while ( lambda <= LAMBMAX )
            {
            if ( rho >= MAX_LOAD )
                {
                break;
                }

            /*******************************************************/
            /* Print the simulation parameters                     */
            /*******************************************************/

            printf ("\n\n");
printf ("\n\n simulation para :max_time = %d lambda = %e pee = %e",
                                        max_time,lambda,pee);

            /*******************************************************/
            /* Call the simulator routine for all batch run  and .*/
            /* print the results                                   */
            /*******************************************************/
```

```
            for ( run_ind = 0; run_ind <= NUM_RUN - 1; run_ind++ )
                {

                simuli (max_time,run_ind,lambda,pee);
/*
              printf ("\n\n batch = %d ser_time = %e delay = %e",
              run_ind,perf_box.batch[run_ind].ave_serv,
                perf_box.batch[run_ind].ave_delay);
*/
                num_run = run_ind + 1;
                }

            /* get the average result of all batch run                */

            sim_stat (FCT_RES,num_run,lambda,0);
printf ("\n\n Average results: service = %e delay = %e rho = %e",
                    perf_box.service,perf_box.delay,perf_box.rho);

#ifdef  SIM_TEST
            for ( i = 0; i <= MAX_PAQ - 1; i += 4 )
                {
printf ("\n\n distribution: D(%d)=%e  D(%d)=%e  D(%d)=%e D(%d)=%e",
                i,perf_box.dist[i],i+1,perf_box.dist[i+1],i+2,
                perf_box.dist[i+2],i+3,perf_box.dist[i+3]);
                }
       printf ("\n\n distribution: D(%d)=%e",i,perf_box.dist[i]);

            printf ("\n\n distr. average = %e  total prob. = %e",
                              perf_box.dis_ave,perf_box.tot_prob);
#endif
            rho = perf_box.rho;
            if (rho > .7)
                {
                lambda = lambda + (LAMBINC/2.0);
                }
            else
                {
                lambda = lambda + LAMBINC;
                }

            }
        pee = pee + PEEINC;
        rho = 0.0;
        }
    }
```

```
/*****************************************************************/
/*                                                               */
/*      simuli:                                                  */
/*                                                               */
/*      Simulation routine called by the main program. It simulate*/
/*      the multicast switch for a specific value of:            */
/*                                                               */
/*                max_time = the lenght of the simulation (slots) */
/*                lambda = average packet arrival rate           */
/*                pee = the multicast probability                */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              long int  max_time                               */
/*              short     run_ind                                */
/*              double    lambda                                 */
/*              double    pee                                    */
/*                                                               */
/*      OUTPUT VALUE:                                            */
/*                                                               */
/*                                                               */
/*****************************************************************/

simuli (max_time,run_ind,lambda,pee)

/****************/
/*  parameter   */
/****************/

long int        max_time;
short           run_ind;
double          lambda;
double          pee;

    {

    /****************/
    /*  variables   */
    /****************/

    short       i,j,k,ind_in,ind_ou;
    short       num_arr,err_code;
    short       poisson(),enter_queu(),exit_queu();
    float       mth$random(),ran_num,prob_int,cum_prob;


    /****************/
    /*  program     */
    /****************/

    /*****************************************************************/
    /* Initialise the clock and the seeds for RANDOM NUMBERS ...  */
```

```
/* ..generation,the statitics and the input and output queues */
/*************************************************************/

err_cnt = 0;
ran_num = 0.0;

sim_stat (FCT_INI,run_ind,0.0,0);

for (i = 0; i <= NI-1; i++)
    {
    in_seed[i] = (unsigned)(seed1[run_ind]*(i+1));
    tab_inqueu[i].pt_in = &tab_inqueu[i].li_tran[0];
    tab_inqueu[i].pt_ou = &tab_inqueu[i].li_tran[0];
    tab_inqueu[i].num_tran = 0;
    }

for (j = 0; j <= NT-1; j++)
    {
    ou_seed[j] = (unsigned)(seed2[run_ind]*(j+1));
    tab_ouqueu[j].num_cop = 0;
    tab_ouqueu[j].busy_cnt = 0;
    for ( k = 0; k <= NI-1; k++ )
        {
        tab_ouqueu[j].ind_cop[k] = 0;
        }
    }


/*************************************************************/
/* Initialise the clock and simulate for the desired number  */
/* of slots                                                  */
/*************************************************************/

sys_clock = 0L;
while ( sys_clock <= max_time )
    {

    /*********************************************************/
    /* Generate packets arrivals at input port(POISSON)      */
    /*********************************************************/

    for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
        {
        num_arr = poisson(&in_seed[ind_in],lambda);
        if ( num_arr > 0 )
            {
            /* packet arrival(s) at input port -> update queue*/

            for ( i = 1; i <= num_arr; i++ )
                {
                err_code = enter_queu(ind_in);
                if (err_code != SUCCES )
```

```
                    {
                    /* queu is full -> consider as an error   */

                    sim_err ( err_code );
                    break;
                    }
                }
            }
        }

        /*************************************************************/
        /* Generate traffic distribution if new HOL(head of line)  */
        /* packet                                                    */
        /*************************************************************/

        for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
            {

            if (tab_inqueu[ind_in].pt_ou->num_cop == -1 &&
                                tab_inqueu[ind_in].num_tran > 0 )
                {
                /* New HOL packet ->update input queue information*/
                /* ... and generate the copies                    */

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d serv_beg = %d",ind_in,
                        tab_inqueu[ind_in].pt_ou,sys_clock);
#endif

                tab_inqueu[ind_in].pt_ou->ser_beg = sys_clock;
                tab_inqueu[ind_in].pt_ou->num_cop = 0;


                /* Choose the output ports                        */

label1:         for ( ind_ou = 0; ind_ou <= NT-1; ind_ou++ )
                    {
                    ran_num = mth$random(&in_seed[ind_in]);
                    if ( ran_num < pee )
                        {
                        /* copy generated for that output ->update*/
                        /* ... output and input queues            */

                        tab_inqueu[ind_in].pt_ou->num_cop += 1;
                        tab_ouqueu[ind_ou].num_cop += 1;
                        tab_ouqueu[ind_ou].ind_cop[ind_in] = 1;
                        }
                    }

#ifdef  SIM_DEBUG
```

```
printf ("\n\n in_queu = %d tran_num = %d num_cop = %d",ind_in,
        tab_inqueu[ind_in].pt_ou,tab_inqueu[ind_in].pt_ou->num_cop);
#endif

                /* Takes care of the cases when no copie is        */
                /* generated                                       */

                if (tab_inqueu[ind_in].pt_ou->num_cop == 0 )
                    {
                    /* packet has generated 0 copy ->try again with*/
                    /* the same packet (modified binomial)          */

                    goto label1;
                    }
                }
            }

        /*************************************************************/
        /* Contention resolution at all output ports                 */
        /*************************************************************/

        /* MODIF: Evaluate traffic distribution at output           */

        sim_stat (FCT_DIS,run_ind,lambda,0);

        /* END MODIF                                                */

        for ( ind_ou = 0; ind_ou <= NT-1; ind_ou++ )
            {
      /* For each output trunk,select the winning copies by first */
      /* finding the probability intervalles size depending ...   */
      /* ... on the number of copies in contention and obtaining  */
      /* ... a random number                                      */

          for ( i = 1; i <= NP; i++ )
              {

              if ( tab_ouqueu[ind_ou].num_cop != 0 )
                  {
                  /* Output queue not empty                       */

               prob_int = 1.0/(double)tab_ouqueu[ind_ou].num_cop;
                  ran_num = mth$random(&ou_seed[ind_ou]);
                  cum_prob = prob_int;
                  tab_ouqueu[ind_ou].busy_cnt +=1;

                  /* Scan for copies from particular input port */

                  for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
                      {
                      if ( tab_ouqueu[ind_ou].ind_cop[ind_in] == 1)
                          {
```

```
                        /* Found a copy from input port "ind_in" */

                          if ( ran_num < cum_prob )
                              {
                              /* This copy is chosen "lucky" -> */
                              /* update output and input queues */
                              /* and exit                       */

                           tab_ouqueu[ind_ou].ind_cop[ind_in] = 0;
                             tab_ouqueu[ind_ou].num_cop -=1;
                           tab_inqueu[ind_in].pt_ou->num_cop -=1;
                              break;
                              }
                          else
                              {
                              /* update the probability valid ...*/
                              /* interval and look again for the */
                              /* lucky input port                */

                              cum_prob = cum_prob + prob_int;
                              }
                          }
                      }

                  }
#ifdef  SIM_DEBUG

printf ("\n\n ou_queu = %d num_cop = %d",
                            ind_ou,tab_ouqueu[ind_ou].num_cop);

#endif

            }

        /*********************************************************/
        /* Update system clock                                   */
        /*********************************************************/

        sys_clock += 1;

        /*********************************************************/
        /* Test for service completion at input queues and       */
        /* cumulate the statistics if nececessary.               */
        /*********************************************************/

        for ( ind_in = 0; ind_in <= NI-1; ind_in++ )
            {
            if (tab_inqueu[ind_in].num_tran > 0 &&
                        tab_inqueu[ind_in].pt_ou->num_cop == 0)
                {
```

```
            /* service completed -> update input queue,       */
            /* cumulate the statistics and update transaction */
            /* output pointer                                 */

            exit_queu( ind_in,run_ind );
            }
        }
    }

/*************************************************************/
/* Output the simulation results for the received values of ..*/
/* ... "lambda" and "pee"                                    */
/*************************************************************/

sim_stat (FCT_REP,run_ind,lambda,0);
return;
}
```

```
/*****************************************************************/
/*                                                               */
/*      poisson:                                                 */
/*                                                               */
/*        Routine that generates packet arrivals according to    */
/*        a POISSON distribution. The method use is derived      */
/*        from the Von-Neumann algorithm. The algorithm assume   */
/*        an arrival rate equal to 1 and a time interval equal   */
/*        to "lambda". But fortunately this is the same as having */
/*        a time interval equal to 1 and an average arrival rate */
/*        equal to "lambda".                                     */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*        - address of the seed to supply to the random number   */
/*          generator.                                           */
/*                                                               */
/*        - average packet arrival rate (lambda)                 */
/*                                                               */
/*      RETURN VALUE:                                            */
/*                                                               */
/*        - number of packet arrivals                            */
/*                                                               */
/*****************************************************************/
short poisson (adr_seed,lambda)

/****************/
/*  parameters  */
/****************/

unsigned int    *adr_seed;
double          lambda;

    {

    /****************/
    /*  variables   */
    /****************/

    short       i,num_arr;
    float       mth$random(),mth$exp(),ran_num,expo;
    double      pro_num,mlambda;

    /****************/
    /*  program     */
    /****************/


    /*****************************************************************/
    /* Get the exponential ( prob of no arrival) for the received ..*/
    /* ... value of lambda                                           */
    /*****************************************************************/
```

```
    mlambda = -lambda;
    expo = mth$exp(&mlambda);

    /*****************************************************************/
    /* Determine the number of arrivals                              */
    /*****************************************************************/

    pro_num = 1.0;
    for ( i = 1; i <= MAX_ARR; i++ )
        {
        ran_num = mth$random(adr_seed);
        pro_num = pro_num * ran_num;
        if ( pro_num < expo )
            {
            /* number of arrivals is determine                       */

            num_arr = i-1;
            break;
            }
        }
    return(num_arr);
    }

/*****************************************************************/
/*                                                               */
/*      enter_queu:                                              */
/*                                                               */
/*              Routine that takes care of managing the queue for */
/*              packets arrivals.                                */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              - input queu indicator                           */
/*                                                               */
/*****************************************************************/
short enter_queu(ind_in)

/****************/
/*  parameters  */
/****************/

short ind_in;

    {

    /****************/
    /*  variables   */
    /****************/

    short       err_code;
```

```
    /*****************/
    /*  program      */
    /*****************/


    if ( tab_inqueu[ind_in].num_tran == IN_SIZE )
        {
        /* Input queue is full                              */

        err_code = QUEU_FULL;
        }
    else
        {
        /* Input queue not full -> save entry info. ... */
        /* ... and update queue                             */

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d arr_time = %d",ind_in,
                          tab_inqueu[ind_in].pt_in,sys_clock);
#endif

        tab_inqueu[ind_in].pt_in->arr_time = sys_clock;
        tab_inqueu[ind_in].pt_in->num_cop = -1;  /* New HOL */
        tab_inqueu[ind_in].pt_in->pri_out = -1;
        if ( tab_inqueu[ind_in].pt_in++ ==
                        &tab_inqueu[ind_in].li_tran[tran_lim] )
            {
         tab_inqueu[ind_in].pt_in = &tab_inqueu[ind_in].li_tran[0];
            }
        tab_inqueu[ind_in].num_tran += 1;
        err_code = SUCCES;
        }
return(err_code);
    }


/*****************************************************************/
/*                                                               */
/*      exit_queu:                                               */
/*                                                               */
/*              Routine that takes care of input queu when packets*/
/*              exit.                                            */
/*                                                               */
/*      INPUT PARAMETER:                                         */
/*                                                               */
/*              - input queu indicator                           */
/*              - batch indicator                                */
/*                                                               */
/*****************************************************************/
short exit_queu( ind_in,run_ind )
```

```
/****************/
/*  parameters  */
/****************/

short ind_in;
short run_ind;

    {

    /****************/
    /*  variables   */
    /****************/

    short       err_code;

    /****************/
    /*  program     */
    /****************/


    tab_inqueu[ind_in].pt_ou->ser_end = sys_clock;

#ifdef  SIM_DEBUG

printf ("\n\n in_queu = %d tran_num = %d ser_end = %d",ind_in,
       tab_inqueu[ind_in].pt_ou,tab_inqueu[ind_in].pt_ou->ser_end);
#endif

    sim_stat (FCT_CUM,run_ind,0.0,tab_inqueu[ind_in].pt_ou );

    if ( tab_inqueu[ind_in].pt_ou++ ==
                        &tab_inqueu[ind_in].li_tran[tran_lim] )
        {
        tab_inqueu[ind_in].pt_ou = &tab_inqueu[ind_in].li_tran[0];
        }
    tab_inqueu[ind_in].num_tran -= 1;
    err_code = SUCCES;
    return(err_code);
    }


/*****************************************************************/
/*                                                               */
/*      sim_stat:                                                */
/*                                                               */
/*            Routine that manage all the performance statistics */
/*            of the simulation.                                 */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              - function code                                  */
/*              - pointer to an input transaction structure      */
```

```
/*                                                                    */
/**********************************************************************/
sim_stat (fct_code,run_ind,lambda,adr_tran)

/****************/
/*  parameters  */
/****************/

short   fct_code;
short   run_ind;
double  lambda;
struct  IN_TRAN *adr_tran;

    {

    /****************/
    /*  variables   */
    /****************/

        float   num_run;
        short   i,j,num_cop;


    /****************/
    /*  program     */
    /****************/

    switch ( fct_code )

        {
        case 0:                 /* Initialisation                */

        perf_box.batch[run_ind].num_paq = 0;
        perf_box.batch[run_ind].cum_serv = 0;
        perf_box.batch[run_ind].cum_delay = 0;
        for ( i = 0; i <= MAX_PAQ; i++ )
            {
            perf_box.batch[run_ind].paq_dis[i] = 0.0;
            }
        break;

        case 1:                 /* cumm. the statistics          */

        if ( sys_clock > WARM_UP )
            {
            perf_box.batch[run_ind].num_paq +=1;
            perf_box.batch[run_ind].cum_serv =
                        perf_box.batch[run_ind].cum_serv +
                          (adr_tran->ser_end - adr_tran->ser_beg);
            perf_box.batch[run_ind].cum_delay =
                        perf_box.batch[run_ind].cum_delay +
                          (adr_tran->ser_end - adr_tran->arr_time);
```

```
        }
    break;

    case 2:                 /* report statistics                */

    perf_box.batch[run_ind].ave_serv =
                    (double)perf_box.batch[run_ind].cum_serv /
                    (double)perf_box.batch[run_ind].num_paq;
    perf_box.batch[run_ind].ave_delay =
                    (double)perf_box.batch[run_ind].cum_delay /
                    (double)perf_box.batch[run_ind].num_paq;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.batch[run_ind].paq_dis[i] =
        perf_box.batch[run_ind].paq_dis[i] /
                                (double)(SIM_TIME - WARM_UP);
        }
    break;


    case 3:     /* compute the final result by averaging over */
                /* all batch results                          */

    num_run = run_ind;              /* convert to float       */
    perf_box.service = 0.0;
    perf_box.delay = 0.0;
    perf_box.rho = 0.0;
    perf_box.dis_ave = 0.0;
    perf_box.tot_prob = 0.0;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.dist[i] = 0.0;
        }

    for ( i = 0; i <= NUM_RUN - 1; i++ )
        {
  perf_box.service = perf_box.service + perf_box.batch[i].ave_serv;
  perf_box.delay = perf_box.delay + perf_box.batch[i].ave_delay;
        for ( j = 0; j <= MAX_PAQ; j++ )
            {
            perf_box.dist[j] = perf_box.dist[j] +
                                perf_box.batch[i].paq_dis[j];
            }
        }
    perf_box.service = perf_box.service / num_run;
    perf_box.delay = perf_box.delay / num_run;
    perf_box.rho = lambda * perf_box.service;
    for ( i = 0; i <= MAX_PAQ; i++ )
        {
        perf_box.dist[i] = perf_box.dist[i] / num_run;
        perf_box.dis_ave = perf_box.dis_ave + perf_box.dist[i] *
(double)i;
```

```
            perf_box.tot_prob = perf_box.tot_prob + perf_box.dist[i];
            }
        break;

        case 4:    /* get data for evaluation of distribution of  */
                        /* number of packets in conflict          */

        if ( sys_clock > WARM_UP )
            {
            num_cop = 0;
            for ( i = 0; i <= NT - 1; i++ )
                {
                num_cop = num_cop + tab_ouqueu[i].num_cop;
                }
            perf_box.batch[run_ind].paq_dis[num_cop] += 1.0;
            }
        break;
        }
    return;
    }


/*****************************************************************/
/*                                                               */
/*      sim_err:                                                 */
/*                                                               */
/*              Error handling routine                           */
/*                                                               */
/*      INPUT PARAMETERS:                                        */
/*                                                               */
/*              - Error identification code                      */
/*                                                               */
/*****************************************************************/
sim_err (err_code)

/****************/
/*  parameters  */
/****************/

short   err_code;

    {

    /****************/
    /*  variables   */
    /****************/



    /****************/
    /*  program     */
    /****************/
```

```
err_cnt += 1;
if ( err_cnt <= MAX_ERR )
    {
    printf ("\n\n error code = %d",err_code);
    }
return;
}
```

```
/*****************************************************************/
/*                                                               */
/*      sim3.h:                                                  */
/*                                                               */
/*      Fichier contenant les variables associees au commutateur */
/*      simmule ainsi que les structures de donnees              */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/* Constant definition                                           */
/*****************************************************************/
/*
#define         SIM_TEST        1
#define         SIM_DEBUG       1
*/
#define         MAX_ERR         4
#define         MAX_RUN         10  /* Maximum number of batch run*/
#define         MAX_ARR         16  /* Maximum number of arrival .*/
                                    /* ... in a slot time        */
#define         MAX_PAQ       NI*NT /* max # packets at output   */

/*****************************************************************/
/* Function codes                                                */
/*****************************************************************/

#define         FCT_INI 0            /* Initialisation           */
#define         FCT_CUM 1            /* Cummulation of statisti  */
#define         FCT_REP 2           /* Compute,print perf. report */
#define         FCT_RES 3           /* Compute final results     */
#define         FCT_DIS 4           /* data for evaluation of #  */
                                    /* packet in conflict        */

/*****************************************************************/
/* Error codes                                                   */
/*****************************************************************/

#define         SUCCES          1
#define         QUEU_FULL       -1

/*****************************************************************/
/* Structures definitions                                        */
/*****************************************************************/

struct  IN_TRAN         /* transaction pour chaque paquet aux ports */
                        /* d'entree                                 */
        {
        long int          arr_time;       /* arrival time           */

        long int          ser_beg;   /* beginning of service time   */
        long int          ser_end;   /* end of service time         */
        short             num_cop;   /* # copies generated          */
```

```
        short           pri_out;   /* active flag that indicates  */
                                   /* if no output ports are chosen */
        };


struct  IN_QUEU     /* queue circulaire de transactions a chaque  */
                    /* port d'entree                              */
        {
        short   num_tran;
        struct  IN_TRAN *pt_in;
        struct  IN_TRAN *pt_ou;
        struct  IN_TRAN li_tran[IN_SIZE];
        };


struct  OU_QUEU         /* queue de sortie (imaginaire) a chaque  */
                        /* port de sortie                         */
        {
        short           num_cop;        /* # copies in contention */
        short           busy_cnt;       /* busy count of the queue*/
        short           ind_cop[NI];   /* ind. de copies provenant*/
                                        /* de chaque entree       */
        };

struct STAT_DATA        /* Performance statistics of the system   */
                        /* for each batch run                     */
        {
        long int        num_paq;    /* number of packets served   */
        long int        cum_serv;   /* cummulative service time   */
        long int        cum_delay;      /* cummulative delay      */
        double          ave_serv;       /* average service time   */
        double          ave_delay;      /* average delay          */
        double          paq_dis[MAX_PAQ+1];/* Distribution of #   */
                                /* packet at output ports         */
        };


struct PERF_BOX      /* Mail box for performance results passing  */
        {
        struct STAT_DATA batch [MAX_RUN];
        double          service;
        double          delay;
        double          rho;
        double          dist[MAX_PAQ+1];
        double          dis_ave;
        double          tot_prob;
        };


unsigned        seed1[MAX_RUN] ={
                                1735467,1671,97631,41,745,
                                89655987,506463,7,8651,4789031
```

```
                                };

unsigned        seed2[MAX_RUN] ={
                                217,41097839,1459361,6543,67,
                                932,76172583,9,4327,345565
                                };
```

```
/*****************************************************************/
/*                                                               */
/*      size.h:                                                  */
/*                                                               */
/*      Fichier contenant les variables associees au commutateur */
/*      simmule.                                                 */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/* Constant definition                                           */
/*****************************************************************/


#define         NI      16      /* number of input port          */
#define         NT      16      /* number of output trunk        */
#define         NP      2       /* number of port per trunk      */
#define         NO      NT*NP   /* number of output port         */
#define         PEEMIN  0.2
#define         PEEMAX  1.0
#define         LAMBMIN .04
#define         LAMBMAX 1.0
#define         LAMBINC .02
#define         PEEINC  .1
#define         NUM_RUN 8       /* number of batch run desired   */
#define         SIM_TIME 8000   /* simulation duration in slots  */
#define         WARM_UP  400    /* warm up duration in slots     */
#define         MAX_LOAD 1.0    /* maximum load to the systeme   */
#define         IN_SIZE  256    /* input queue size              */
```