Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT
Modeling and Performance of Token Bus LAN with Petri Nets

by
Zhenggang Pan

A token bus local area network (LAN) is analyzed by Generalized Stochastic Petri Nets (GSPN). The GSPN models of both a single station and LAN for four types of service schemes are obtained and their liveness property is proved. The network performance parameters comprise throughput and delay. The performance analysis for both symmetric and asymmetric single-service systems is conducted for varying load. In order to analyze a token bus LAN with a large number of stations, an approximation method is developed to resolve the state space explosion problem. A token bus LAN with twenty-one stations is used to show the approximation method.

The contributions of this thesis are 1) modeling token bus LAN using GSPN, 2) performance evaluation of five-station token bus with symmetric and asymmetric service cases, and 3) approximate performance evaluation of a token bus LAN with a large number of stations.

MODELING AND PERFORMANCE OF TOKEN BUS LAN WITH PETRI NETS

by
Zhenggang Pan

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science
Department of Electrical and Computer Engineering
May 1992

APPROVAL PAGE

Modeling and Performance of Token Bus LAN with Petri Nets

by
Zhenggang Pan

Dr. MengChu Zhou, Thesis Advisor
Assistant Professor
Department of Electrical and Computer Engineering, NJIT

Dr. Daniel Chao, Committee Member
Assistant Professor
Department of Computer and Information Science, NJIT

Dr. Anthony Robbi, Committee Member
Associate Professor
Department of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Zhenggang Pan

Degree: Master of Science in Electrical Engineering

Date: May, 1992

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering, New Jersey Institute of Technology, Newark, NJ, 1992
- Bachelor of Science in Electrical Engineering, Nanjing Institute of Posts and Telecommunications, Nanjing, People's Republic of China, 1982

Major: Electrical Engineering

This thesis is dedicated to

my mother

ACKNOWLEDGMENT

I wish to express my sincere gratitude to my thesis advisor Dr. MengChu Zhou for his ingenious guidance and encouragement.

Special thanks to Dr. Daniel Chao and Dr. Anthony Robbi for serving as members of the committee.

I would like to thank my wife for her love and support.

I also wish to thank my relatives, because of whom I was able to study in the United States.

And finally, a thank you to my schoolmate Mr. J. M. Ma for his help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Over the past decade, complex and diverse computer communication networks have been established. A computer communication network is a system of interconnected computers and other devices capable of exchanging information. A generic network component (i.e., computer or device) is refered to as a *node* or *station.* A station belonging to a network is capable of communicating with any other station in the same network. A local area network (LAN) is a computer communication network within a small area, which is characterized by an interstation distance in the order of magnitude of a few kilometers. Typically, a LAN is owned by a single organization such as a hospital, a university, a manufacturing plant, or an office.

IEEE has produced several LAN standards collectively known as IEEE 802 which include CSMA/CD, token bus, and token ring. These standards include the protocols of the medium access control (MAC) sublayer based on ISO OSI (Open Systems Interconnection) Reference Model as shown in Figure 1.1. This model is composed of seven layers. The IEEE 802 standards only concern the lowest two layers (Data Link layer and Physical layer). The structure of Data Link layer (layer 2) of OSI model is shown in Figure 1.2. The MAC sublayer is located at the bottom of the Data Link layer. The token bus standard is called IEEE standard 802.4 [1] published by IEEE standards Committee in 1985. It has been widely implemented throughout industry, including MAP (Manufacturing Automation Protocol) network architectures used in the manufacturing industry. Actually, a MAP network is a broadband token bus LAN [2].

The increasing computer communication networks pose challenging evaluation problems due to their user requirements and complexity. Performance evaluation of local area networks is an important topic, since it allows choices to be made in terms of

many factors affecting performance. It allows assessment of the different alternatives available in local area network designs in order to optimize certain potential benefits, while minimizing their associated costs. For example, a MAP network allows certain options such as setting various timers and buffer size at the different layers of the OSI reference model. Potential benefits (e.g., performance) include improved response time, better serviceability, etc.

Network performance models usually belong to two major categories: analytical and simulation.

Analytical methods are based on mathematical models that characterize the system under study. The models are usually queueing models and Petri net models. However, while queueing models have been used extensively for the evaluation of computing system, their application to systems that exhibit concurrency, synchronization, fault tolerance, and degradable performance is not straightforward. Furthermore, queueing network cannot easily represent fault-related behavior, since the probabilistic nature of their structures (e.g., branching probabilities) is fixed. The analysis of asymmetric network is complicated and difficult by using queueing models [3]. On the other hand, Petri nets, especially generalized stochastic Petri nets (GSPN) [4], thereof are much better suited to the modeling of systems which exhibit such properties. In particular, the use of timed "transitions" with probabilistic timing permits, via different interpretations of tokens, simultaneous representation of characteristics related to both performance and reliability [5].

Some other advantages of modeling systems with the generalized stochastic Petri net include the following: 1) the exact results can be easily obtained for the class of systems with Poisson arrivals and exponentially distributed service time. These results can be used to validate approximate solutions when these become available. 2) GSPN results can be obtained interactively, as opposed to simulation results that take much longer time.

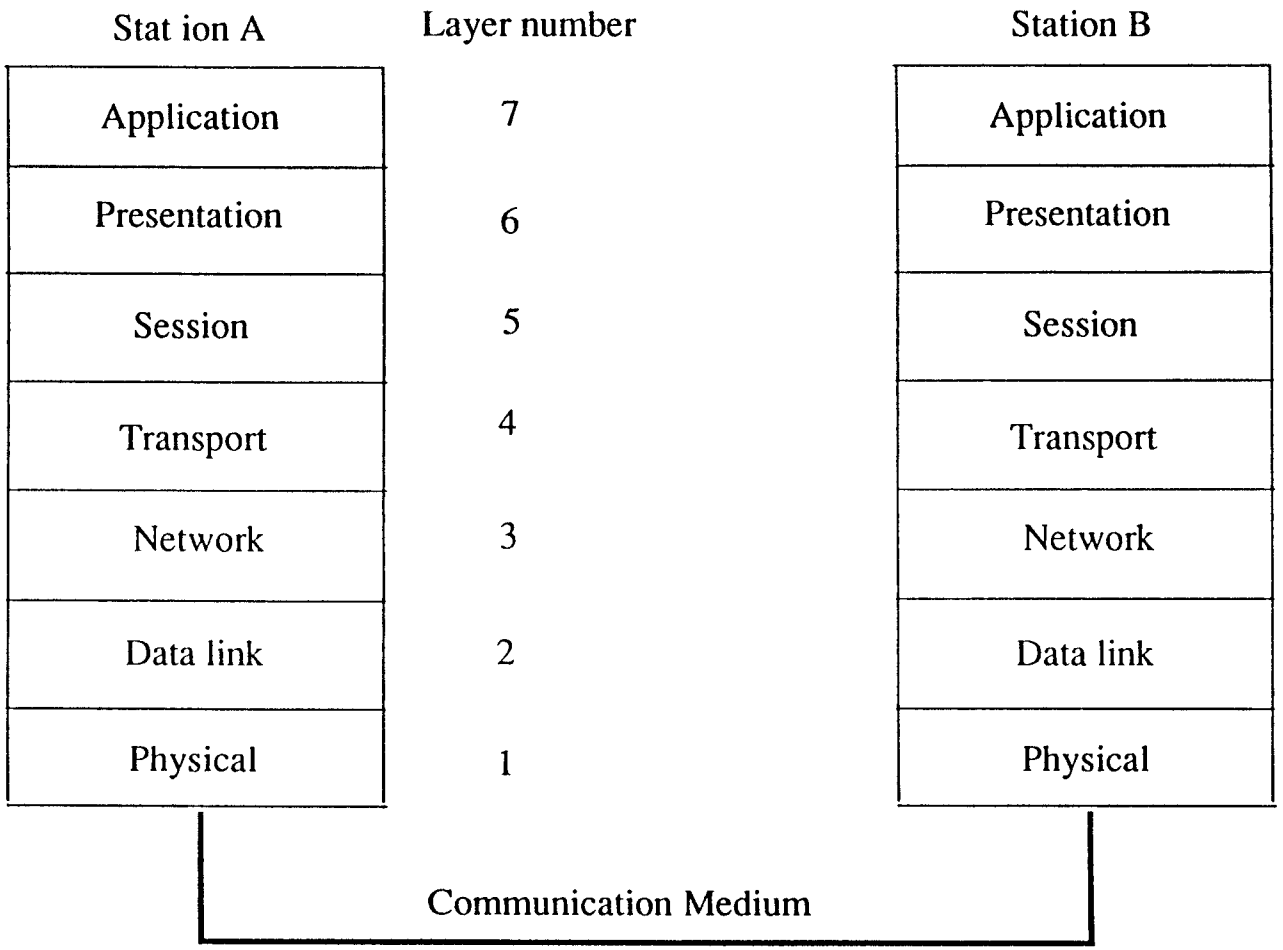

Figure 1.1 OSI reference model for a communication system

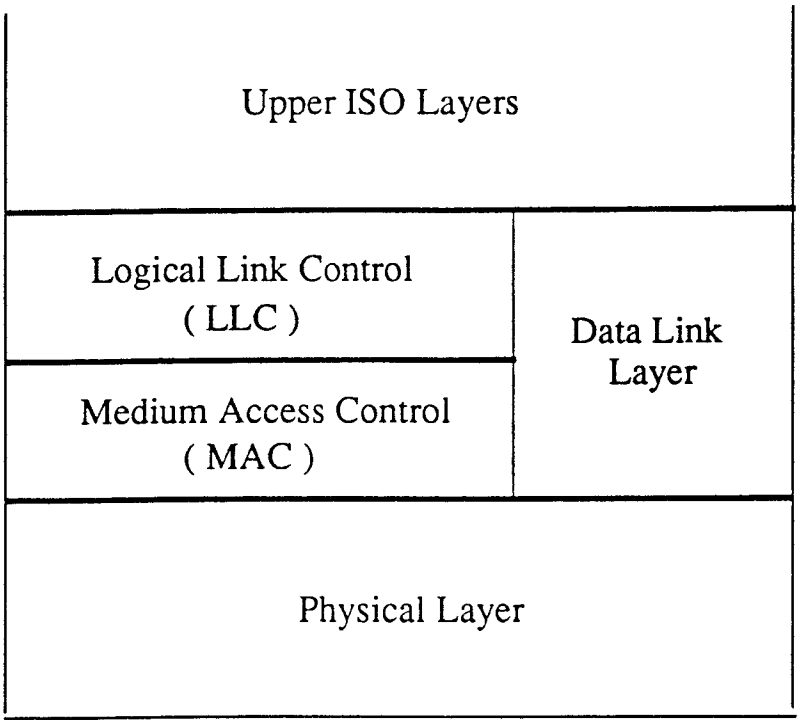

Figure 1.2 IEEE 802 sublayer structure

GSPN can be viewed as a way of specifying, generating, and solving Markov models. As such, all the capabilities of Markov chains are available. However, the state space of the resultant Markov model is generally large even for a small number of stations. Thus, an approximate method of modeling networks is needed to obtain a compact GSPN model to reduce the state space. This method is important for a GSPN to model a LAN with a large number of stations. The computer-aided analysis of GSPN models is useful to performance evaluation of networks. A stochastic Petri net package called the SPNP [6] is utilized to automatically generate and solve the Markov models. Besides obtaining the steady-solution, SPNP can also obtain transient solutions and carry out parametric sensitivity analysis [7].

Simulation models are somewhat similar to analytical models. Analytical and simulation models differ, in that simulation models include model extensions for obtaining the solution through the use of a computer program that behaves like the system under study. By studying the performance of the program, one can infer the performance of the simulated system.

This thesis is motivated by the increasing importance of Petri net theory's application in the area of computer-communication networks and by the importance of modeling and performance evaluation of token bus LAN.

Accordingly, the objectives of this thesis are as follows:

1. To provide basic concept of a local area network (LAN) and detail about the token bus LAN;

2. To provide an introduction to generalized stochastic Petri nets (GSPN):

3. To model token bus LAN with generalized stochastic Petri nets;

4. To present and analyze the performance of the token bus LAN;

5. To present an approximation method and results for token bus LAN with a large number of stations.

This thesis is organized as follows. The next chapter provides basic concepts and properties of Petri nets, introduction to stochastic Petri nets and generalized stochastic Petri nets, and Petri net applications in network modeling and analysis. Chapter 3 provides a brief description of IEEE 802.4 protocol, token bus network and its service types. Chapter 4 provides the GSPN models, modeling methods and detail modeling procedure of token bus LAN. Chapter 5 provides the performance analysis based on GSPN models of token bus LAN. In Chapter 6, an approximation method is provided to reduce the state space requirement of GSPN for modeling token bus LAN with a large number of stations. Finally, Chapter 7 provides a summary of the major points discussed in the thesis.

CHAPTER 2
PETRI NETS IN NETWORK MODELING AND ANALYSIS

2.1 Generalized Stochastic Petri Nets

Petri nets are formal graph models that are well suited for representing the flow of information and control in systems that exhibit concurrency and synchronization characteristics [8]. However, the concept of time is not explicitly given in the original definition of Petri nets. For performance evaluation of dynamic systems, it is necessary and useful to introduce time delays associated with transitions and/or places in their net models. Such a Petri net model is known as a (deterministic) *timed Petri net* if the delays are deterministically given, or as a *stochastic Petri net (SPN)* if the delays are probabilistically specified with exponential distribution [9]. Based on stochastic Petri net (SPN), generalized stochastic petri net (GSPN) is obtained by allowing transitions to belong to two different classes: *immediate* transitions and *timed* transitions with exponential distribution.

2.1.1 GPSN Definition and Properties

A GSPN can be defined as an eight-tuple [10]:

$Z = (P, T, I, O, m, H, F, P_r)$

where

$P = \{p_1, p_2, ..., p_n\}$, $n > 0$, and is a finite set of places;

$T = \{t_1, t_2, ..., t_s\}$, $s > 0$, and is a finite set of transitions with $P \cup T \neq \emptyset$, $P \cap T = \emptyset$;

I: $P \times T \rightarrow N$ and is an input function where $N = \{0, 1, 2, ...\}$;

O: $P \times T \rightarrow N$ and is an output function;

m: $P \rightarrow N$ and is a marking whose i^{th} component is the number of tokens in the i^{th} place. An initial marking is denoted by m_0; and

H: $P \times T \rightarrow N$ and is an inhibitor function;

F: $T \rightarrow (0, \infty)$ is a vector whose i^{th} component is the exponential firing rate if the i^{th} transition is timed or otherwise undefined or ∞ if the i^{th} transition is immediate.

P_r: $P \rightarrow R^{\Sigma_{p \in P}|p^*|}$, such that $\Sigma_{t \in p^*} P_r(t) = 1$ and $P_r(t) \geq 0$, $\forall p \in P$, where $p^* = \{t \in p^{\cdot}: t \text{ is an immediate transition } \}$, and $p^{\cdot} = \{t \in T : I(p,t) \neq 0\}$.

In this definition, the first five tuples define an ordinary Petri net [11], [12]. Within it, the places are represented by circles and the transitions are represented by bars.

A place may contain *tokens* (represented by dots). The *marking* of a place is the number of tokens which the place contains. The marking of the Petri net is a vector that specifies the marking of each place in the net. A place is defined to be an *input place* of a transition if an arc exists from the place to the transition. Similarly, a place is defined to be an *output place* of a transition if an arc exists from the transition to the place. An integer $d > 1$ (default value 1), called its *multiplicity*, is associated with each arc.

A transition is *enabled* if each of its input places contains as many tokens as the multiplicity of the corresponding arc for an ordinary Petri net. An enabled transition can *fire*. When a transition fires, a number of tokens are removed from each of its input places equal to the multiplicity of the corresponding arc and it deposits in each of its output places as many tokens as the multiplicity of the corresponding arc.

H is an inhibitor function from places to transitions. An *inhibitor arc* connects a place to a transition, and is represented by a line terminating in a small circle rather than an arrow head. It functions to prevent a transition from firing under certain markings. Thus a transition may fire if each of its normal input places contains at least as many tokens as the multiplicity of the connecting arc and each of its inhibitor input places contains fewer tokens than its multiplicity. Each firing generates a new

marking of the net. The token number of inhibitor input place remains unchanged.

In a GSPN, there are two kinds of transitions. *Immediate* transitions fire in zero time once they are enabled. *Timed* transitions fire after an exponentially distributed random enabling time. In the figures in this thesis immediate transitions are represented by solid bar, and timed transitions by hollow bar.

If several immediate transitions have input arcs from the same place, these transitions must have different probabilities because they cannot fire at the same time. Several immediate transitions can fire simultaneously if their input arcs from different places.

The *reachability set* of a Petri net for a given initial marking is the set of all states (or markings) that can be generated from the initial state by a sequence of transition firings.

A GSPN has following behavioral properties which are the same as an ordinary Petri net [13]:

Safeness A place in a Petri net is safe if the number of tokens in that place never exceed one. A Petri net is safe if all of its places are safe.

Boundedness Boundedness is a generalization of safeness of a net with the situation that the places can hold a particular number of tokens. A place is k-bounded, if the number of tokens in that place cannot exceed an integer k. A Petri net is defined to be k-bounded if every place in it is k-bounded.

Liveness A transition is live, if and only if for any number in the reachability set there is a firing sequence whose firing enables the transition. A Petri net is live if each of its transition is live. Liveness of a Petri net implies freedom from deadlock.

2.1.2 An Example of GSPN

An example of a generalized stochastic Petri net is shown in Fig.2.1 to illustrate the

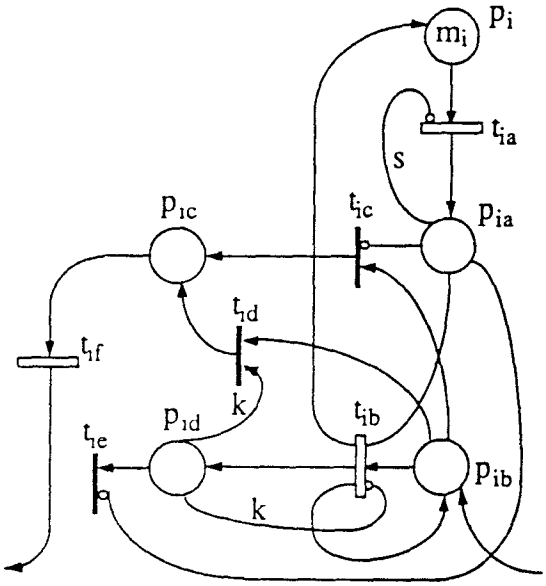

Figure 2.1 Single-station GSPN model for limited service

concept of GSPN. This GSPN model is a station model of token bus LAN with limited service in which a station keeps sending packets until either the queue is emptied or the number K of packets are sent.

In the Figure 2.1, eight tuples in GSPN definition are represented as follows.

A set of places: $P = \{p_i, p_{ia}, p_{ib}, p_{ic}, p_{id}\}$;

A set of transitions:

$T = \{t_{ia}, t_{ib}, t_{ic}, t_{id}, t_{ie}, t_{if}\}$;

An input function:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

An output function:

$$O = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

An inhibitor function:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A marking:

$$m = \begin{bmatrix} m_i \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

A firing rate vector:

$$F = \begin{bmatrix} \lambda \\ \mu \\ \infty \\ \infty \\ \infty \\ \gamma \end{bmatrix}$$

An arc probability vector:

$$P_r = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

It is noticed that the immediate transitions t_{ic} and t_{id} have input arcs from the same place, p_{ib}, but these two transitions cannot fire at the same time. The immediate transitions t_{ic} and t_{ie} fire simutanously, but the input arcs come from different places p_{ib} and p_{id}. Therefore, the probabilities of these three immediate transitions are equal to one.

2.2 Review of Petri Nets in Communications

This section reviews various previous efforts to analyze communication system using Petri nets. Previous research on Petri net application in the communication area falls into two basic categories: protocol specification and verification; network modeling and performance.

- Protocol specification and verification:

  Diaz [14] presented Petri net models for each of the following protocols, and verified their liveness.

  1. Alternating bit protocol
  2. Packet switching call establishment protocol
  3. The subscribers and the CCITT No 7 protocol; Connection-Disconnection of entities
  4. The X.21 interface protocol

  Sajkowski [15] presented a new approach to the verification of a communication protocol modelled as a discrete-event system. This approach is based on the analysis of a communication protocol considered as a time-driven system. The verification technique combines time constraints base projection and the examination of the safeness of certain places in a timed Petri net model.

  Diaz [16] surveyed the applicability of Petri nets for protocols, as well as for service specification and validation. At the specification level, different classes of nets are introduced, and emphasis is given to the modular specification of a protocol layer. At the validation level, the analysis techniques implemented in the CAD package OGIVE/OVIDE are used in order to prove safety and progress properties of a protocol layer.

Juanole [17] concerned the formal specification of the gateway connecting a LAN and a remote computer through ISDN, using Petri nets.

Mukherjee [18] investigated the performance of flow control and error control protocols in computer communication using GSPN. The go-back-n and the selective repeat protocols were considered for error recovery. The sliding-window protocol was considered for flow control.

- Network modeling and performance

  Gressier [19] presented a stochastic Petri net (SPN) model of Ethernet very close to design specifications. This model, in spite of row stochastic approximations and assumptions gave accurate results for network loads less than 50% of transmission medium capacity.

  Marsan [20] used deterministic and stochastic Petri nets (DSPN) models to develop models of several fiber optics local area network architectures which were Expressnet, D-net, Fasnet, U-net and Token ring, and gave some comparative results.

  Marsan [21] developed two timed transition Petri net (TTPN) models of a six-station LAN with linear topology, in which a finite number of stations located randomly along the bus channel and access method based on a 1-persistent CSMA/CD protocol. The first model contains a very detailed representation of the LAN behavior, modeling each station individually, but the model is intractable from an analytical point of view. Then the second model was a compact model that is a DSPN, and hence permits an analytical approach to its solution.

CHAPTER 3
TOKEN BUS LOCAL AREA NETWORK

This chapter is concerned with the detail of token bus local area network (LAN) and its service types.

3.1 Token Bus Principle

As depicted in Figure 3.1 a physical bus interconnects stations using *token bus scheme.* All stations connected to the bus cooperate in the use of the shared channel. The basic idea behind the channel access mechanism involves the concept of the *right to use the channel.* Basically, only the station having the right to use the channel is allowed to send messages. The right to use the channel is referred to as the *token* which is a special control frame. Each time a station acquires the token, it can transmit data frames for a certain amount of time, then it must pass the token to the next station. If the frames are short enough, several consecutive frames may be sent. If a station has no data, it passes the token immediately to the next one upon receiving it. Since only one station at a time holds the token, collisions do not occur.

The token is passed from station to station in a cyclic fashion, thus defining a *logical ring* which is shown in Figure 3.1. Therefore, the logical ring determines the sequence for passing the token from station to station. As far as the logical ring is concerned, each station knows only the identity (i.e., the address) of the station preceding it and the one following it. Naturally, a station must know the identity of all remaining stations for the purpose of communicating with them, but it has no idea about their physical location in the logical ring. Although any station is capable of sending and receiving data, stations not included in the logical ring cannot send data, since they are never given the token. These stations are called *listen only* stations (e.g., station 6 and 7 in Figure 3.1), because they can receive data but cannot send them [22].

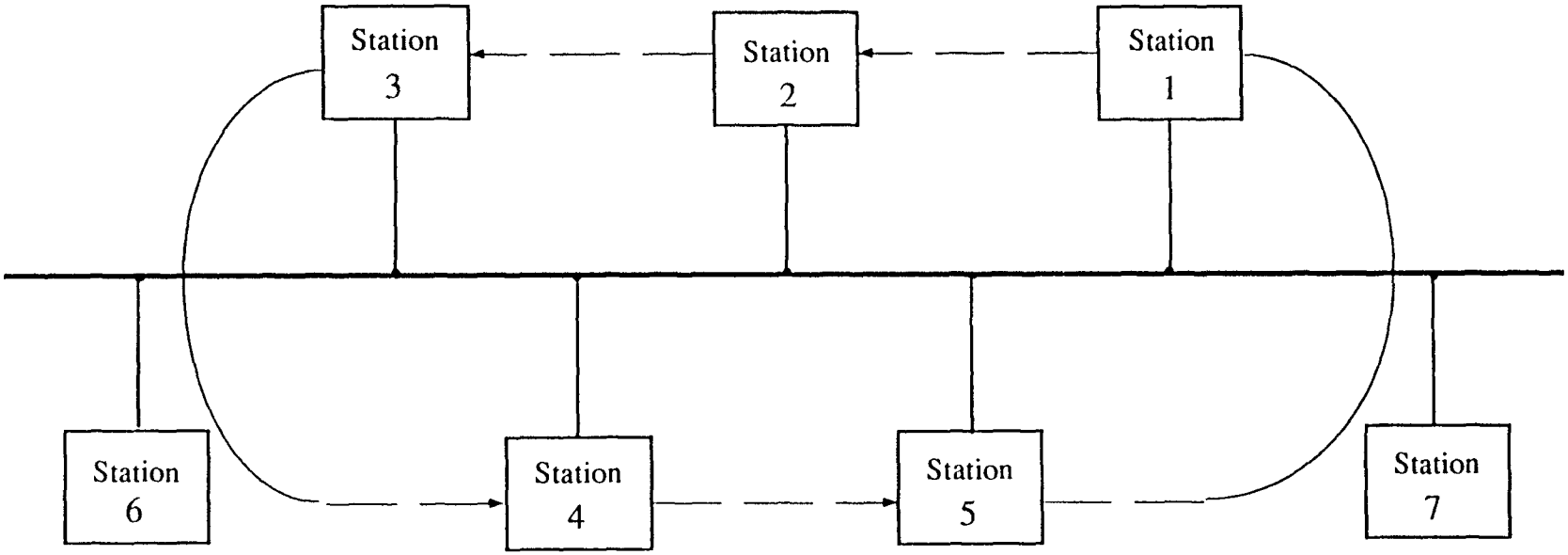

Figure 3.1 A token bus LAN

An important point to realize is that the physical order in which the stations are connected to the cable is not important. The logical order is independent of the physical order. Since the cable is inherently a broadcast medium, each station receives each frame, discarding those not addressed to it. When a station passes the token, it sends a token frame specifically addressed to its logical neighbor in the ring, irrespective of where that station is physically located on the cable. That is, during normal steady state operation, the right to access the medium passes from one station to another. The medium access control (MAC) sublayer provides sequential access to the shared bus medium in a logically circular fashion. This MAC sublayer determines when a station has the right to access the shared medium by recognizing and accepting the token from its predecessor station. For the physical layer, the token bus uses the 75-ohm broadband coaxial cable used for cable television. Both single and dual cable systems are allowed, with or without headends. Three different analog modulation schemes are permitted: phase continuous frequency shift keying, phase coherent frequency shift keying, and multilevel duobinary ampliude modulated phase shift keying. Speeds of 1, 5, and 10 Mbps are possible [22], [23].

3.2 Protocol Description

Token bus protocol, IEEE standard 802.4, is one of IEEE 802 LAN standard series. It is concerned with medium access control (MAC) sublayer [1].

3.2.1 MAC sublayer structure

As depicted in Figure 3.2, MAC sublayer is mainly composed of five elements:

1. Interface Machine (IFM)
2. Access Control Machine (ACM)
3. Receive Machine (RxM)

4. Transmit Machine (TxM)

5. Regenerative Repeater Machine (RRM)

The function of each machine is as follows:

- *Interface Machine*: This machine acts as an interface and buffer between the LLC and MAC sublayer. It interprets all incoming service primitives from the LLC sublayer and generates appropriate outgoing primitives. It handles the queueing of service requests and performs the address recognition function.

- *Access Control Machine*: This machine cooperates with the ACM of all other stations in the logical ring. As an option, the MAC handles messages with priorities. The ACM is also responsible for initialization and maintenance of the logical ring, including admission of new stations, failure detection, and recovery, and handling other failures in the token bus network.

- *Receive Machine*: This machine accepts symbols from the physical layer, assembles them into frames, performs frame validation, and passes the frames to the ACM and IFM. The RxM accomplishes this by recognizing the delimiters for the start of a frame (i.e., the start delimiter, SD) and the end of the frame (i.e., the end delimiter, ED), checking the frame check sequence (FCS), and validating the frame structure. The RxM also identifies and indicates the reception of noise bursts, and bus quiet conditions.

- *Transmit Machine*: This machine accepts a data frame from the ACM and transmits it as a sequence of symbols, in the proper format to the physical layer. The TxM builds a MAC protocol data unit by prefacing each frame with the required preamble and SD, and appending the FCS and ED.

- *Regenerative Repeater Machine*: This machine is an optional MAC component present only in special *repeater stations*, e.g., in a broadband or a head-end

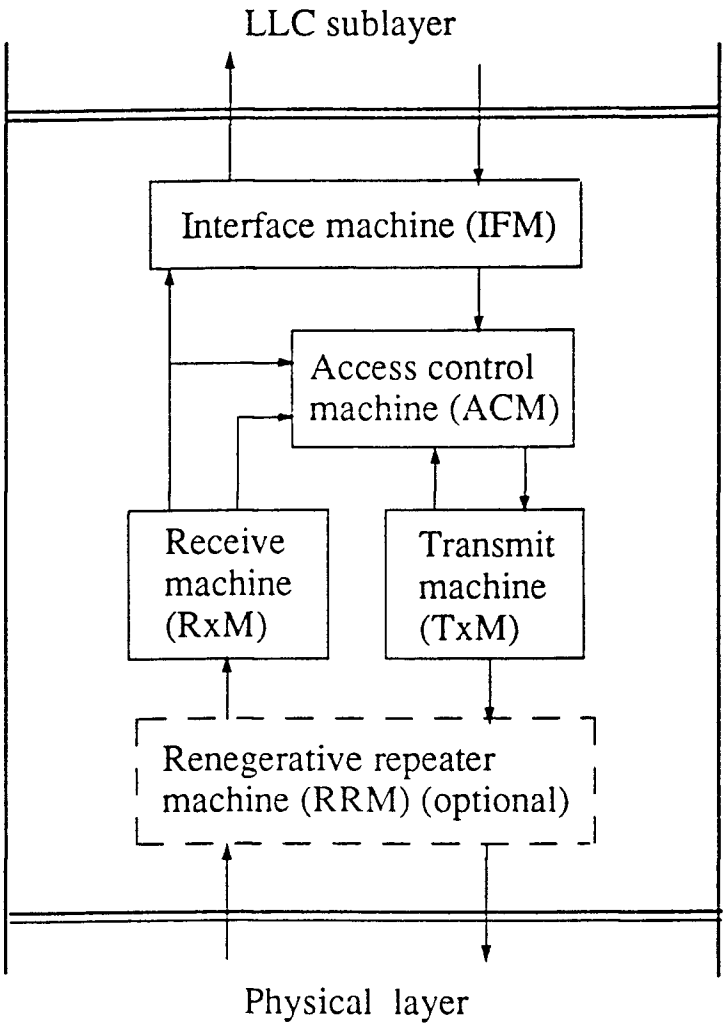

Figure 3.2 Functional configuration of the MAC sublayer

demodulator. In such stations the RRM repeats the incoming atomic symbol stream, from the physical layer, back to the physical layer for retransmission.

Of all these five machines, the ACM is both the most critical and the most complex. It is the key control mechanism for the token-bus method. The IFM and RxM participate heavily in the operation of the MAC sublayer protocol.

3.2.2 Priority Mechanisms

The IEEE 802.4 standard provides an optional priority mechanism. The priority of each frame is indicated when the LLC sublayer submits a data frame to be transmitted to the MAC sublayer. The MAC sublayer offers four levels of priority classes, called *access classes.* The access classes are named 0, 2, 4, and 6, with 6 corresponding to the highest priority and 0 to the lowest.

The priority scheme works as follows: each station is internally being divided into four substations, one at each priority level. As input comes into the MAC sublayer from the above, the data are checked for priority and routed to one of the four substations. Thus each substation maintains its timer with stipulated time known as *Target Rotation Counter* (TRTC). When a station recieves the token it begins transmitting the frames of priority class 6. When it is done, the token is passed internally to the priority 4 substation, which may then transmit until its timer expires, at which point the token is passed internally to priority 2 substation. This process is repeated until all its frames have been sent or its timer has expired. Again, when the station does not have any frames to send, it simply passes the token to its successor.

In this thesis, it is assumed that the protocol implements the highest priority class (class 6) only. In another word, the model assumes a single priority for all data packets.

3.2.3 Frame Format

The token bus frame format is shown in Figure 3.3. The function of each component

in the frame is following. *Preamble* is used to synchronize the receiver's clock.

Starting delimiter (SD) and *Ending delimiter* (ED) fields are used to mark the frame boundaries. Both of these fields contain analog encoding of symbols other than 0s and 1s, so that they cannot occur accidentally in the user data.

Frame control (FC) field is used to distinguish data frames from control frames. For the former, it carries the frame's priority. It can also carry an indicator requiring the destination station to acknowledge correct or incorrect receipt ot the frame. For the latter, the *Frame control* field is used to specify the frame type. The allowed types include token passing and various ring maintenance frames, including the mechanism for letting new stations enter the ring, the mechanism for allowing stations to leave the ring, and so on.

Destination address (DA) field identifies the station to which the frame is destined. This may be 2 or 6 bytes depending on the number of bits used for addressing. All addressing on a given LAN shall be of the same length.

Source address (SA) field identifies the station which originates the frame and has the same format and length as the destination address in a given frame.

Data field may be up to 8182 bytes long when 2-byte addresses are used, and up to 8174 bytes long when 6-byte addresses are used. The data field can contain a LLC protocol data unit, which is used to exchange LLC information between LLC entities or a MAC management data frame which is used to exchange management information between MAC management entities or a value specific to one of the MAC control frames.

The *Frame check sequence*(FCS) field which is a 32 bit frame checking sequence is used to detect transmission errors.

The **token frame** is a special MAC control frame which gives a station the right to transmit data. This frame has the same structure as shown in the Fig. 3-3 except that the DA is the successor's address in the logical ring and that the data field is

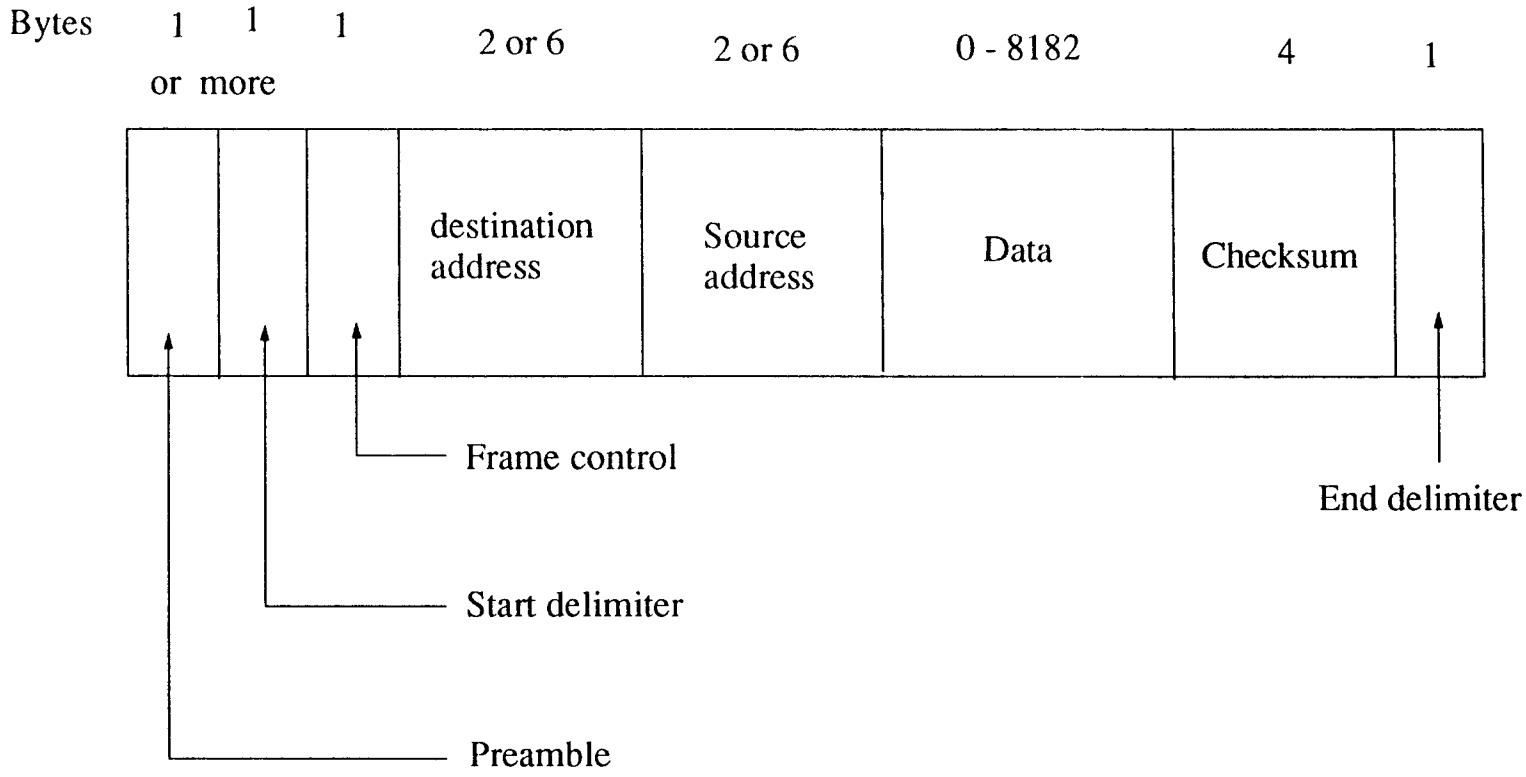

Figure 3.3 IEEE 802.4 frame format

null.

3.3 Service Types

The four service types of token bus LAN are considered as the following [23]:

Single service: Each station has only single-buffer queue, and can send only one packet when each station acquires the *token*.

Exhaustive service: When a station receives the *token*, it continues to send packets until the station queue is empty. Then it passes the *token* to the next station.

Gated service: The station sends only those packets found at the queue when the *token* is received.

Limited service: A station continues to send packets until either 1) the queue empties, or 2) the first fixed number K of packets are sent, whichever comes first.

CHAPTER 4
GSPN MODELING OF TOKEN BUS LAN

4.1 Modelling Method

Since in a moderstely sized communication system, the complexity of design at the implementation level of detail may be unreasonable, researchers are seeking methods for the progressive synthesis of Petri net models. Previous research on Petri net design methods falls into two basic categories: bottom-up and top-down [11], [12].

Bottom-up approaches begin with the construction of subnets for component processed, and proceed to the final net by merging and/or linking all these subnets.

Top-down synthesis is characterized by the stepwise refinement of an aggregate Petri net model. Each successive refinement contains increasing detail until the implementation level is reached.

System decomposition and modular composition are two keys to both approaches. The bottom-up method is used to model the token bus LAN as follows:

Step 1: Model the details of the token bus operation for each station.

Step 2: Link five submodels as a token bus LAN model based on the order of the logical ring.

4.2 Model Assumptions

The following general assumptions have been made to simplify model design and operation:

- The LAN has five stations connected to a shared communication channel based on the IEEE 802.4 specifications.
- The acknowledgement mechanisms between stations are not offered to the data link layer in these models.

- The stations under study are already in the logical ring. No new stations join the logic ring and no station drop from the logic ring. The station always knows its successor and its predecessor in the logical ring.
- The protocol implements the highest priority class (class 6) only [23].
- Customers arrival is modeled as a Poisson process.
- The length of package is of Poisson distribution. The mean package length is 4096 bytes.
- Channel capacity is 10Mbits/sec. Bus length is 1 Km. The length of *token* is fixed as 100 bits.

4.3 GSPN Model Design

In this section, the bottom-up method is used to model operation of token bus LAN according to four types of services. First of all, the behaviors of each station must be modeled for each type of service. Then, the GSPN models of LAN can be obtained by combining each GSPN model of stations.

4.3.1 Modeling a Single Station

Single-Service Type

As shown in Figure 4.1, the single station model consists of one immediate transition with an inhibitor arc, three timed transitions and four places.

The immediate transition (t_{ic}) has zero time delay. The timed transitions: t_{ia} has a firing rate λ which is the arrival rate of packets; t_{ib} has a firing rate μ which is the service rate of the channel; t_{id} has a firing rate γ which is the token rotation rate. Place p_i represents the condition that station i is idle; place p_{ia} represents the condition that station i has generated a packet; place p_{ib} represents the condition that station i has acquired the token and starts to transmit a packet; place p_{ic} represents

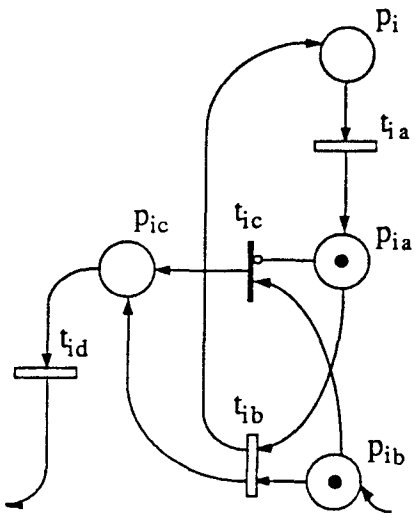

Figure 4.1 Single-station GSPN model for single service

the condition that station i is passing the *token* to the next station. When station i obtains the *token*, only one packet can be sent before station i passes the *token*.

Exhaustive-Servive Type

The single station model is shown in Figure 4.2. The difference between this service and previous one is that whenever the *token* arrives at a station it cannot leave the station until all packets of the station have been sent. this difference is modeled by adding an inhibitor arc from place p_{ia} to transition t_{ia} with the multiplicity s which indicates the buffer size of station i, and by changing the output arc from transition t_{ib} to place p_{ib} instead of p_{ic}. Whenever station i gets the *token*, there is a token in place p_{ib}. if station i already has packets (i.e. place p_{ia} has tokens), the timed transition t_{ib} is enabled and keeps firing until place p_{ia} (buffer) has no token (all packets have been sent). The immediate transition t_{ic} then fires and a token goes to place p_{ic}. The station i starts to pass the *token* on. The number of tokens, m_i, in place p_i is usually larger than the buffer size, s, implying that station i always has packets to arrive.

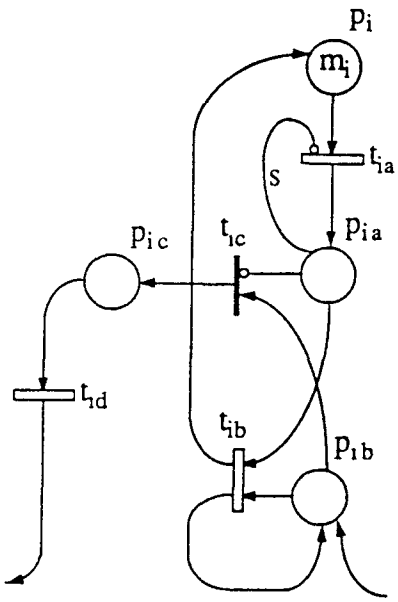

Figure 4.2 Single-station GSPN model for exhaustive service

Gated-Service Type

In this case, on acquiring the *token*, station i serves only the packets which arrive at the station before the *token* arrives. As shown in Figure 4.3, the single station model of gated-service type has made little difference from the model of exhaustive-service type by adding an inhibitor arc from p_{ib} to t_{ia}. Whenever the *token* arrives at station i (a token in place p_{ib}), the transition is disenabled by the inhibitor arc. No more packets can come into the station buffer (place p_{ia}). The gated function is implemented.

Limited-Service Type

In this service case, station i, on receiving the token, either transmits all packets that it finds at the station if the number of packets is less than k or k of them. The single station model is shown in Figure 4.4. The place p_{id} acts as a counter that ensures that at most k packets are transmitted when the token resides at this station. An inhibitor arc to transition t_{ib} originates from place p_{id}. The multiplicity of the arc

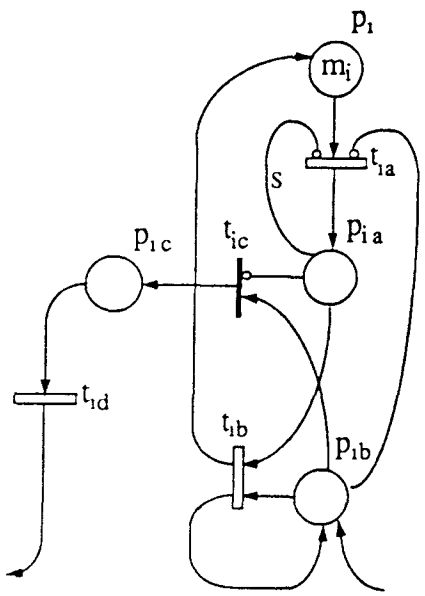

Figure 4.3 Single-station GSPN model for gated service

is k, which means that the timed transition t_{ib} will fire if and only if there are less than k tokens in place p_{id}, at same time, one token in place p_{ib} and at least one token in place p_{ia}. When the number of tokens in place p_{id} reaches k, the timed transition t_{ib} is disabled while the immediate transition t_{id} is enabled. When station i has sent all packets whose number is less than k, i.e., there is no more token in place p_{ia} before the number of tokens in place p_{id} reaches k, immediate transition t_{ic} and t_{ie} fire, and the tokens in place p_{id} are discarded by transition t_{ie}.

4.3.2 Modeling LAN

Based on the bottom-up method, the models of a token bus LAN can be obtained by connecting each station model according to the order of the logical ring. The GSPN models of LAN with five stations for four service cases are described next.

Single-Service Case

The LAN model is shown in Figure 4.5. This model is obtained by linking each

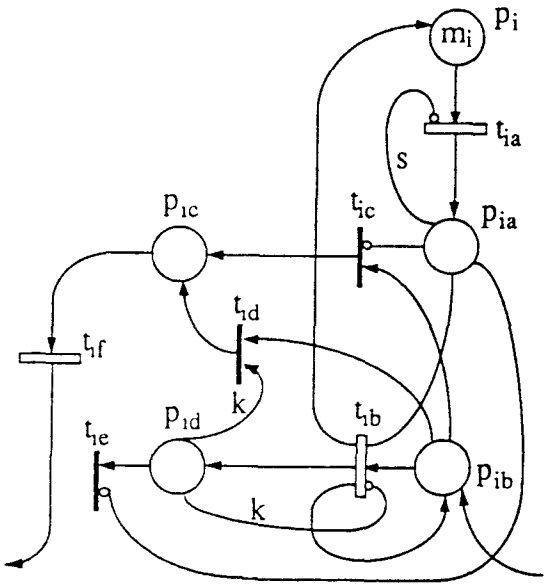

Figure 4.4 Single-station GSPN model for limited service

single-station model based on the order of the logical ring. Places p_1 to p_5 represent the condition that stations 1 to 5 are idle; the places p_{1b} to p_{5b} represent the condition that stations have already received the token and start to transmit packets. The place p_{1c} to p_{5c} represent the condition that the token is passing to the next station. Currently, stations 1, 3, and 5 are idle and station 1 has received the *token* but no packet arrive at it. Stations 2 and 4 have generated a packet but have to wait for the *token* to come. Considering the current marking in Figure 4.5, since p_{1b} contains the *token*, the immediate transition t_{1c} is enabled. It fires immediately and a token is deposited to place p_{1c}. This models the situation that station 1, on receiving the token and finding no packet to transmit, immediately passes the token to station 2. Since station 2 has a packet (a token in p_{2a}), as soon as the token reaches to station 2, the timed transition t_{2b} is enabled. Its firing causes a token to be deposited to place p_2 and another token to be deposited in place p_{2c}. This represents that station 2 returns to the idle state and starts to pass the *token* to station 3.

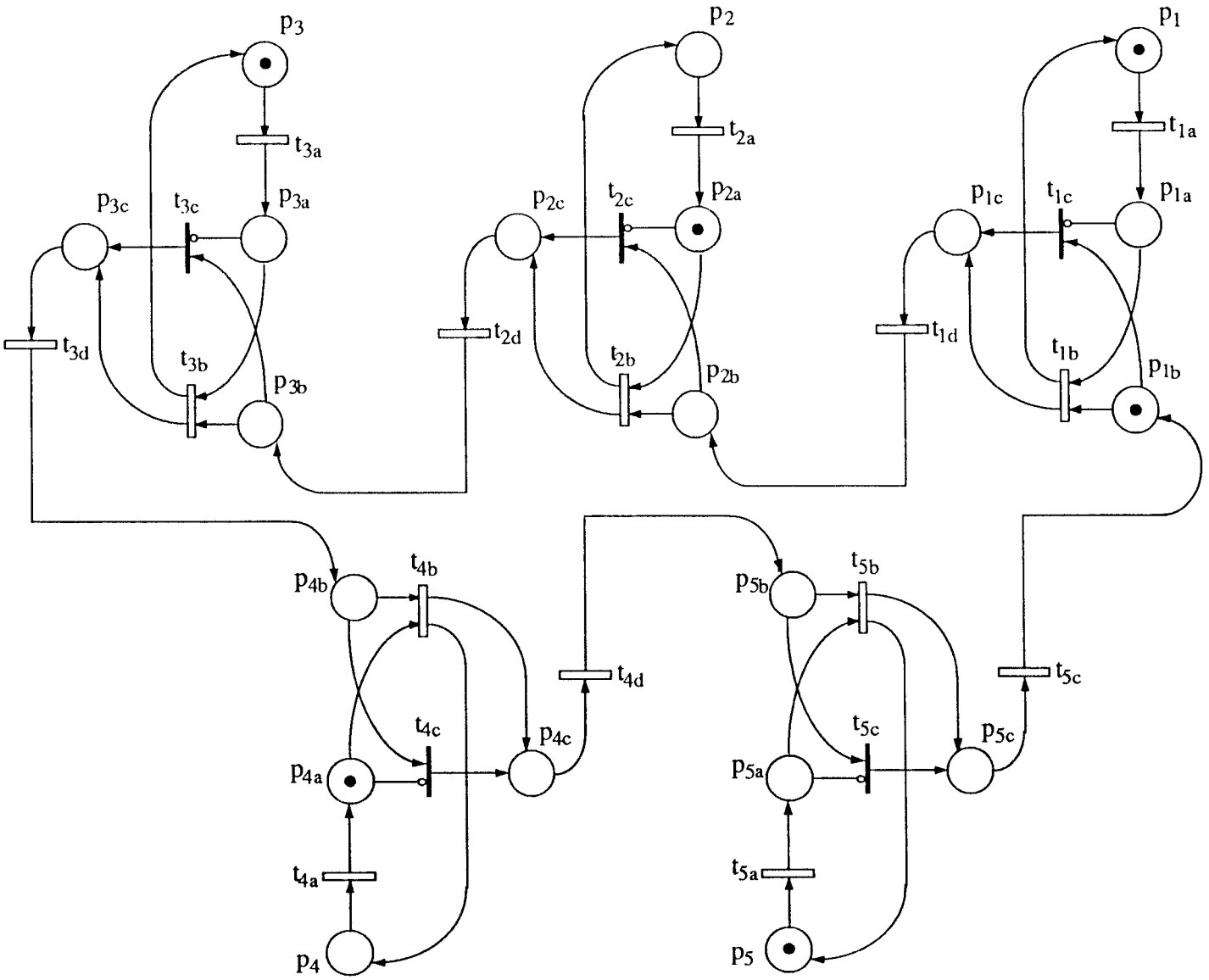

Figure 4.5 GSPN model for single-service (symmetric) token bus LAN

Exhaustive-Service Case

In this service case, the *token* cannot leave a station until all packets in that station have been transmitted. Whenever the *token* leaves a station the number of packets in that station is zero. Figure 4.6 shows the GSPN model for an exhaustive-service token bus LAN. The multiplicity of the inhibitor arc is s which represnts the buffer size of a station. Note that as long as $s < m_i$ holds, the value of m_i does not affect the results. This means that the model is good for an infinite-population system, in which packets always come to stations. In Figure 4.6, station 1 must immediately pass the *token* to station 2 because there is no packet in station 1 at the moment when the *token* reaches the station.

Gated-Service Case

The model of LAN with this service is shown in Figure 4.7. In this service case, after acquiring the *token*, a station sends only those packets that arrive at that station prior to the *token* arrival. This function is implemented by adding an inhibitor arc from places p_{ib} to transitions t_{ia}, $i = 1, 2, 3, 4, 5$. As soon as a station acquires the *token*, t_{ia} is disenabled and no more packets can come into that station. For the current marking in Figure 4.7, the station 1 will send all four packets in place p_{1a}, then t_{1c} fires and a token is deposited to p_{1c}. At this time, t_{1d} is enabled to fire, passing the *token* to station 2.

Limited-Service Case

In this service case, a station, on receiving the *token*, transmits at most k packets. The GSPN model for this type of service is shown in Figure 4.8. For the current marking, the *token* is in station 1. If $k = 3$, station 1 can send only three of packets in place p_{1a}, and then has to pass the *token* to the next station. If $k > 4$, when station 1 has transmitted four packets in p_{1a} no more packets come in. The station then passes the *token* to the next. The counter number k and buffer size s can be different at each station.

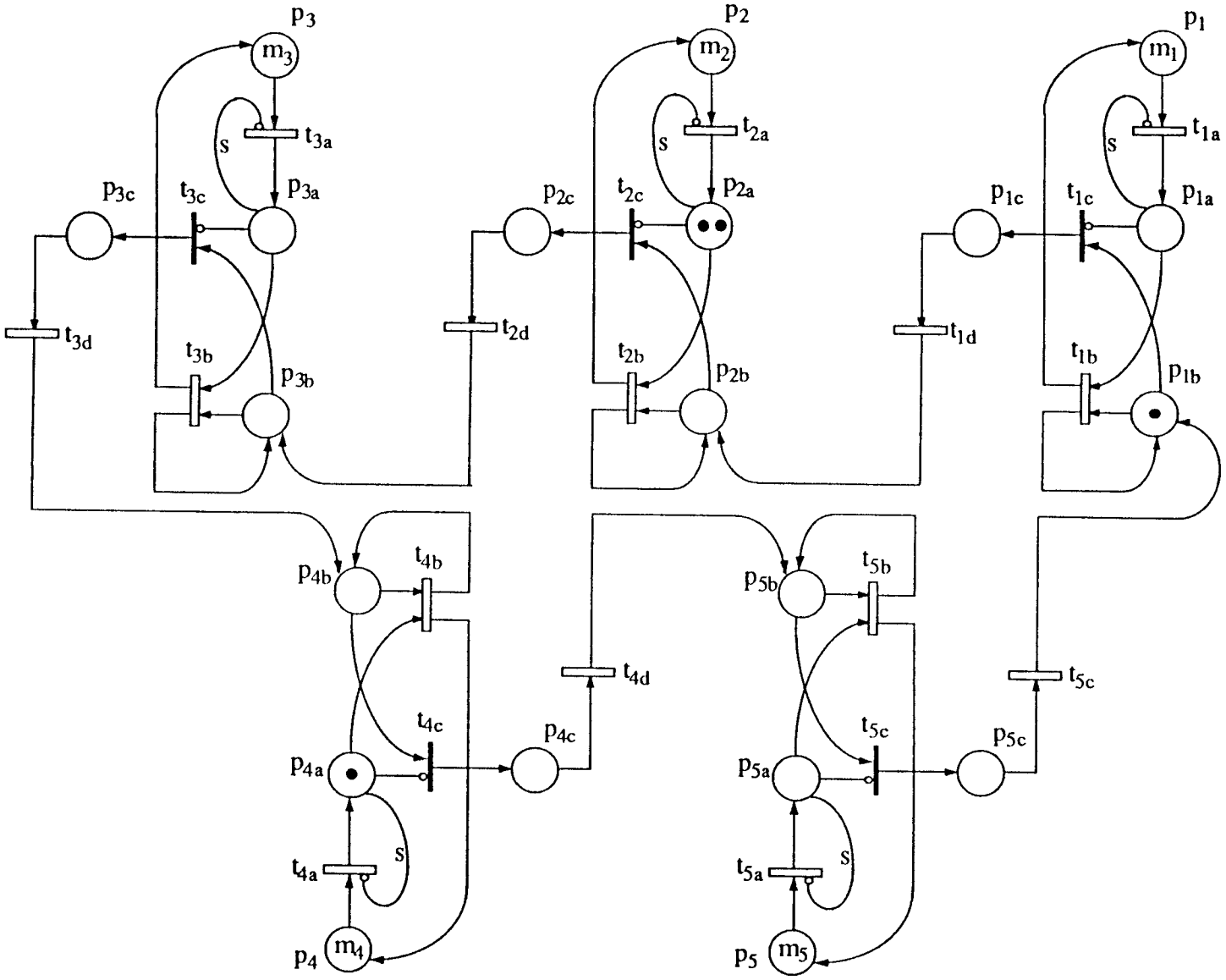

Figure 4.6 GSPN model for exhaustive-service token bus LAN

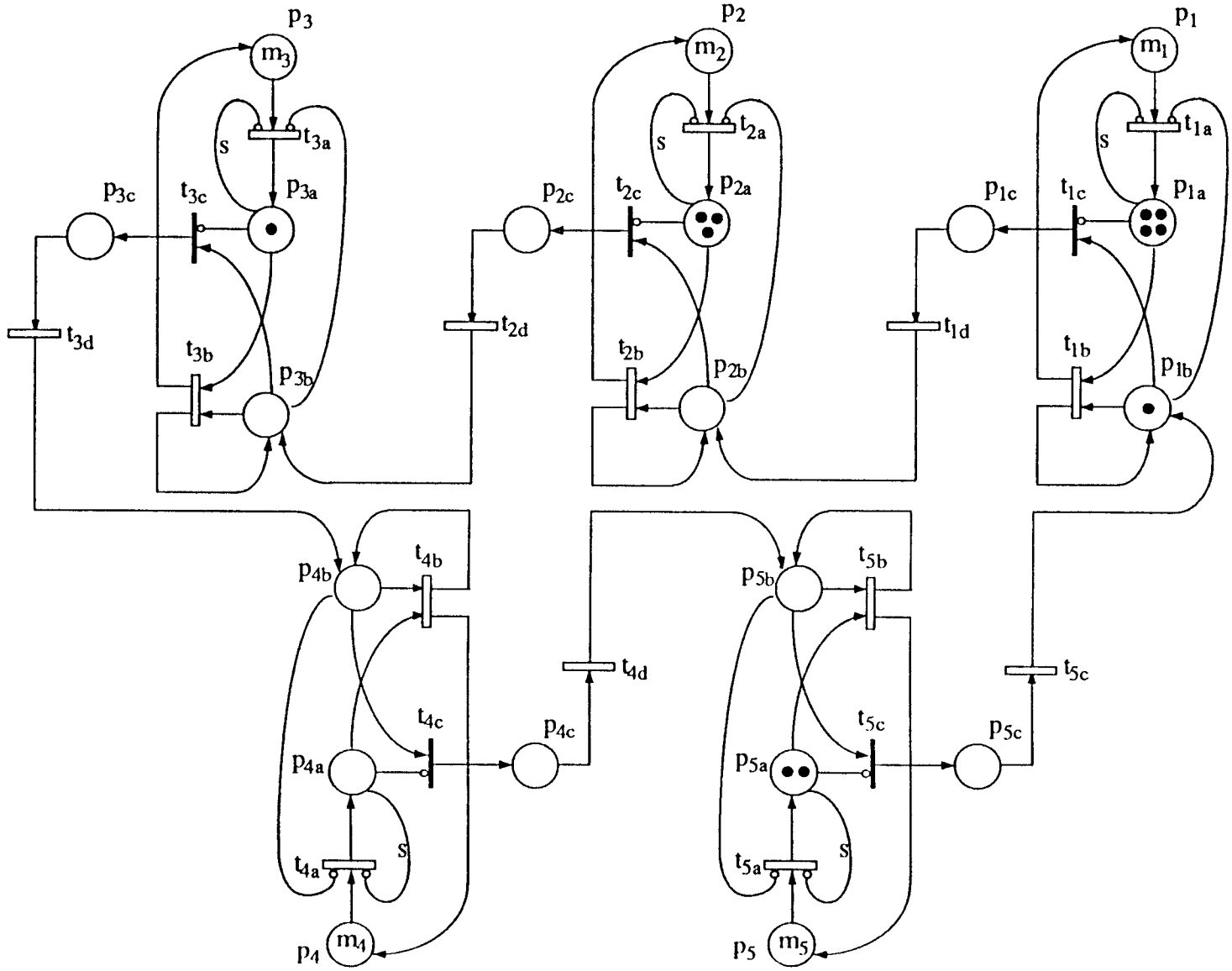

Figure 4.7 GSPN model for gated-service token bus LAN

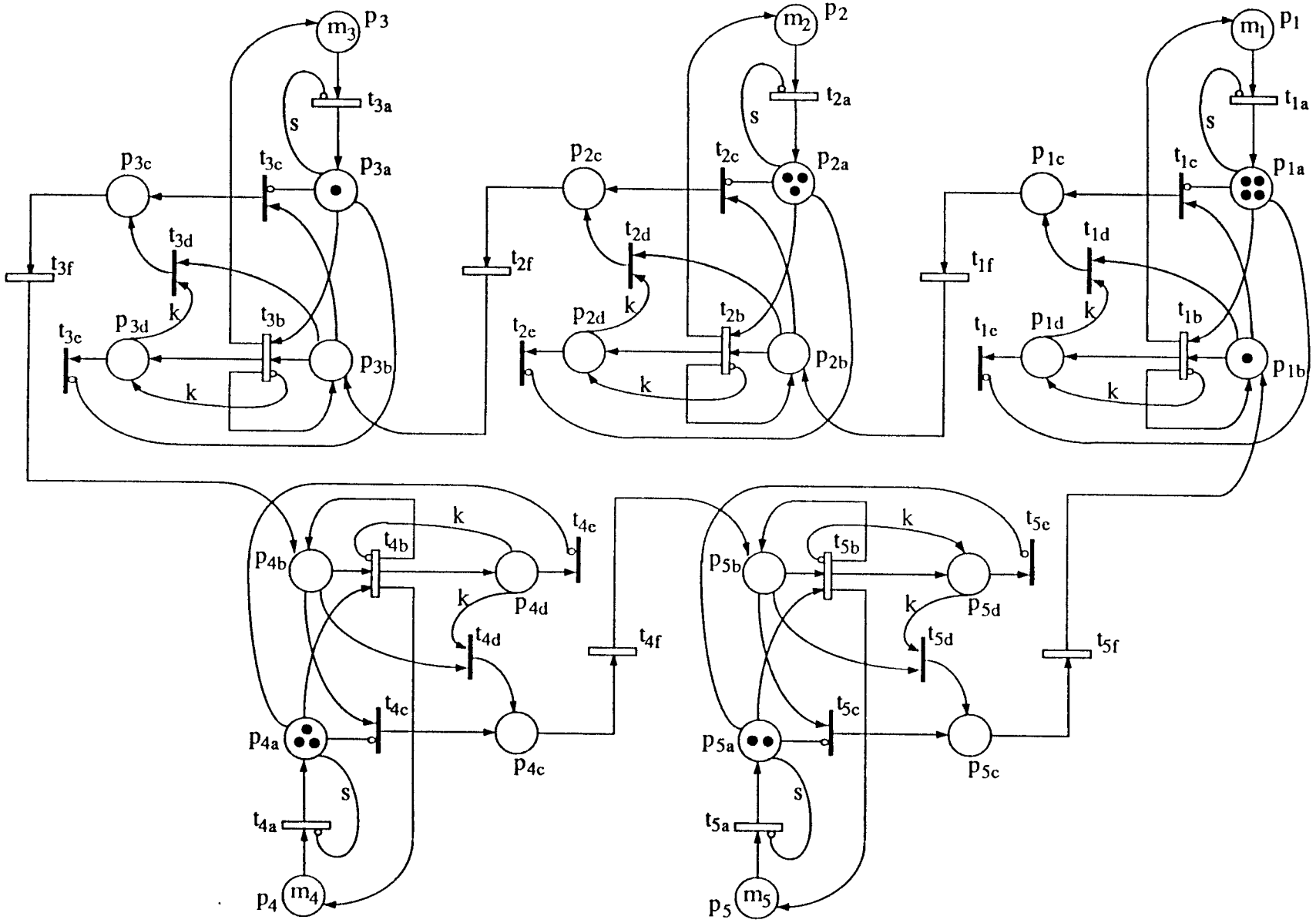

Figure 4.8 GSPN model for limited-service token bus LAN

CHAPTER 5
PERFORMANCE ANALYSIS

The single-service case is considered to do the experiments to analyze the performance of token bus LAN because this type of service is common to LAN.

The experiment results can be obtained by solving the GSPN models using stochastic Petri net package (SPNP), which is a C-language-based package [6]. The input to the SPNP consists of the immediate transitions, the timed transitions and their firing rates, the input and output arcs for each transition, and the initial marking. It generates the rechability graph, eliminates the vanishing markings, and constructs the continuous-time Markov chain. Finally, it obtains the steady-state probability of each marking using a combination of the successive overrelaxation and Gauss-Seidel methods. If requested, it also provides the mean number of tokens in each place.

The experiments are conducted for both symmetric and asymmetric systems. A symmetric system is the one in which each station has same GSPN model structure and has same arrival rate and service rate. An asymmetric system is the system in which each station may have different GSPN models (i.e., different service type) and different arrival rates and service rates.

5.1 Performance Variables

Performance variables can be classified into transient and steady state. The token bus model solution is based on steady state variables, which will be presented. Performance variables of interest for the IEEE 802.4 standard are average packet delay and average throughput.

Average packet delay is defined as the average elapsed time between the instant when a packet arrives at a station and the instant when the last bit of the packet is transmited. There are two major components of average packet delay: queueing

delay and transmision delay. Queueing delay includes the time when packets wait for their turn in order to be sent. Packets wait because the station does not have the *token* when they arrive. Transmission delay is the time that takes to send all bits in a packet at the normal data rate.

Throughput is defined as a number of bytes (or packets) of user data transferred per second. There are two kinds of throughput: the actual measured throughput and throughput that network is capable of providing. The actual measured throughput in packets per second is used in the experiments.

Once performance variables are identified, they need to be incorporated in the experiment file to obtain the corresponding solution model. The incorporation of average packet delay into the experiment file is relatively straightforward. By using the Little's law, average packet delay is calculated as the ratio of the average number of packets in the system (number of token from p_{1a} to p_{5a} and from p_{1b} to p_{5b}) over the average arrival rate of packets from the LLC sublayer (the firing rates from t_{1a} to t_{5a} and from t_{1c} to t_{5c}). Throughput:

$$S = \sum_{i=1}^{N} \bar{s}(t_{ib}) \tag{0.1}$$

Average packet delay:

$$D = \frac{\sum_{i=1}^{N} [\bar{M}_t(p_{ia}) + \bar{M}_t(p_{ib})]}{N \sum_{i=1}^{N} [\bar{s}(t_{ia}) + \bar{s}(t_{ic})]} \tag{0.2}$$

where, $\bar{M}_t(p_{ia})$ and $\bar{M}_t(p_{ib})$ are average number of tokens in places p_{ia} and p_{ib}; $\bar{s}(t_{ia})$, $\bar{s}(t_{ib})$ and $\bar{s}(t_{ic})$ are average firing rates of transitions t_{ia}, t_{ib} and t_{ic}.

5.2 Experiment Parameters

The following parameters are used in the experiments:

1. The arrival processes to each station are statistically equivalent Poisson processes with equal average arrival rates, λ packets/second.

2. The packet length is exponentially distributed with an average value of $X_p = 4096$ bytes.

3. The token length is fixed with the value of $X_t = 12$ bytes.

4. Channel capacity is $R = 10$ Mbps.

5. Token bus length is $L = 1000$ meters.

6. Number of station is $N = 5$.

7. The end-to-end propagation delay of 1000-meter bus is $\tau = 5\mu$sec.

8. The walk time, ω, is the time to transfer the token, plus the token propagation delay, between successive stations in the logical ring.

$$\omega = (X_t/R + \tau/3)$$

The walk time, ω, depends on the average propagation delay between stations. This delay is determined with the assumption that transfer between any two stations on the bus is equally likely. The solution to this problem in probability gives approximately one-third of the length of the bus as the average spacing between randomly chosen stations, or an average time delay of $\tau/3$ seconds.

9. Average service rate, μ, is supposed to be Poisson distributed with average value $\mu = R/X_p = 305$ packets/sec.

10. Average token rotation time, T_c, is approximately assumed to be exponentially distributed with average value $T_c = N(X_p/R + \omega)$ because the length of packets is exponentially distributed. So, the token rotation rate: $\gamma = 1/T_c$.

11. Offered load, ρ is defined as:

$$\rho = \frac{N \cdot \lambda}{\mu}$$

where, N is the number of stations, λ is an average arrival rate, and μ is an average service rate.

5.3 Overview of Experiments

The experiments have been done to prove the liveness of the GSPN models and to evaluate the performance of single service for both symmetric and asymmetric systems.

As far as the liveness property, all models of four-service types have been solved by means of SPNP based on C-language programs and reachability sets have been obtained. Therefore, there is no deadlock in these GSPN models and they are live. The reachability graph of single-service GSPN model with two stations is shown in Figure 5.1. The reachability graph of other models cannot be shown because of too many states.

For the symmetric system, the GSPN model shown in Figure 4.5 is used to investigate the network performance. All parameters of each station are assumed to be the same to meet the condition of symmetric systems. The offered load to the network is changeable so that the network performance can be measured. For a given offered-load, the arrival rate can be calculated accordingly and assigned to all timed-transitions, t_{ia}, $(i = 1, ..., 5)$ as the initial firing rates. After running SPNP, each group of the network data can be obtained numerically according to each value of the offered load. The values of performance variables, throughput (S) and average packet delay (D), are obtained by calculating each group of network data using formula (5.1) and (5.2). Data for transition firing rate are summarized in Table 5.1.

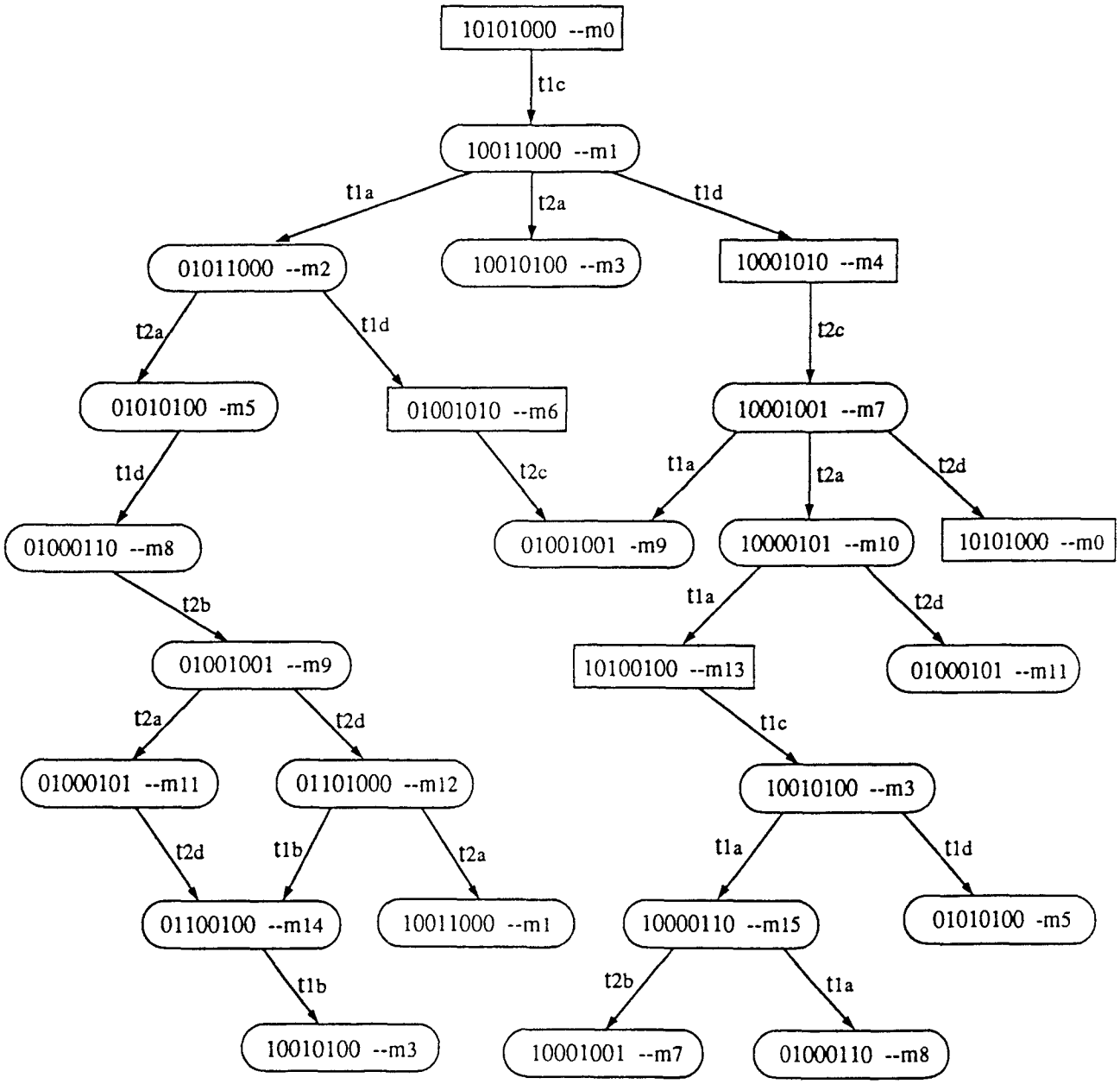

Figure 5.1 Reachability graph of single-service GSPN model with two stations

Transition	t_{ia}	t_{ib}	t_{ic}	t_{id}
Firing Rate (packets/s)	$\frac{\rho\mu}{N}$	305	immediate	12

Table 5.1 Data for firing rates

For the asymmetric system, the GSPN model, as shown in Figure 5.2, consists of one single-station model of exhaustive service and four single-station models of single service. In order to compare the performance with that of the symmetric system mentioned above, all parameters of each station, such as arrival rate λ, token rotation rate γ and so on, are kept the same. In this sense, the asymmtric system means that only the structure of GSPN model is not the same for each station in the network. In another word, each station in the network could have different service type. The buffer size of the station with exhaustive service is five ($s = 5$). With the change of offered load, the performance of the network is obtained.

The experiments of twenty-one station token bus LAN for single service have been conducted to show that the approximation method is effective. The detail will be discussed in the next chapter.

5.4 Results and Analysis

Two principal metrics of network performance are throughput and average packet delay, which are investigated in this study as the offered load varies.

5.4.1 Throughput vs Offered Load

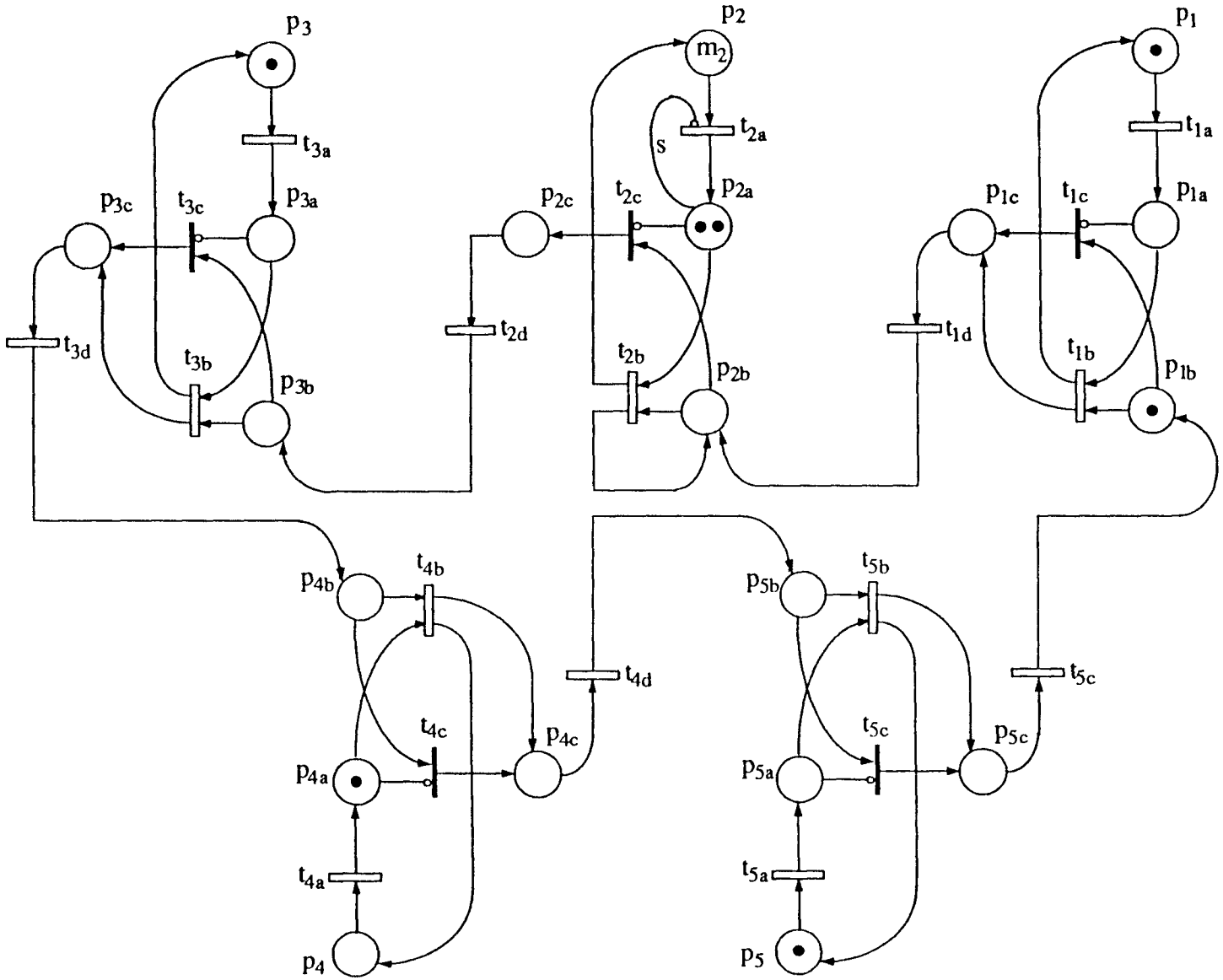

Figure 4.5 GSPN model for asymmetric token bus LAN

The throughput performances of both symmetric system and asymmetric system are shown in Figures 5.3 and 5.4. Both throughputs increase with offered load and reach their maximum values at higher load. For the same offered load, the throughput of the asymmetric system is higher than that of the symmetric system. This result can be analyzed qualitatively. The network throughput can be qualitatively represented as follows [22]:

$$S \propto \frac{T_h}{T_h + \omega}$$

where

T_h is the average token holding time of each station;

ω is token walking time which is a constant:

$$\omega = X_t/R + \tau/3$$

In the asymmetric system, there is one station with exhaustive service. when the token arrives at that station, the token cannot leave until that station has sent all of its packets. So, the token holding time of that station is much loger than that of the others, and average token holding time for each station in this asymmetric system becomes longer. In the symmetric system, each station has the same token holding time statistically (the length of packets is exponentially distributed), which is the service time of one packet. Because the token walking time is unchangeable, due to fixed token length, the network throughput mainly depends on the average token holding time T_h. That is why the throughput of asymmetric system is higher than that of symmetric system.

In the thesis, the offered load indicates the input of data packets only. The token rotating around all idle stations does not included in the offered load. In the design of the GSPN models, timed transition t_{ib} has been considered only for the transmission of

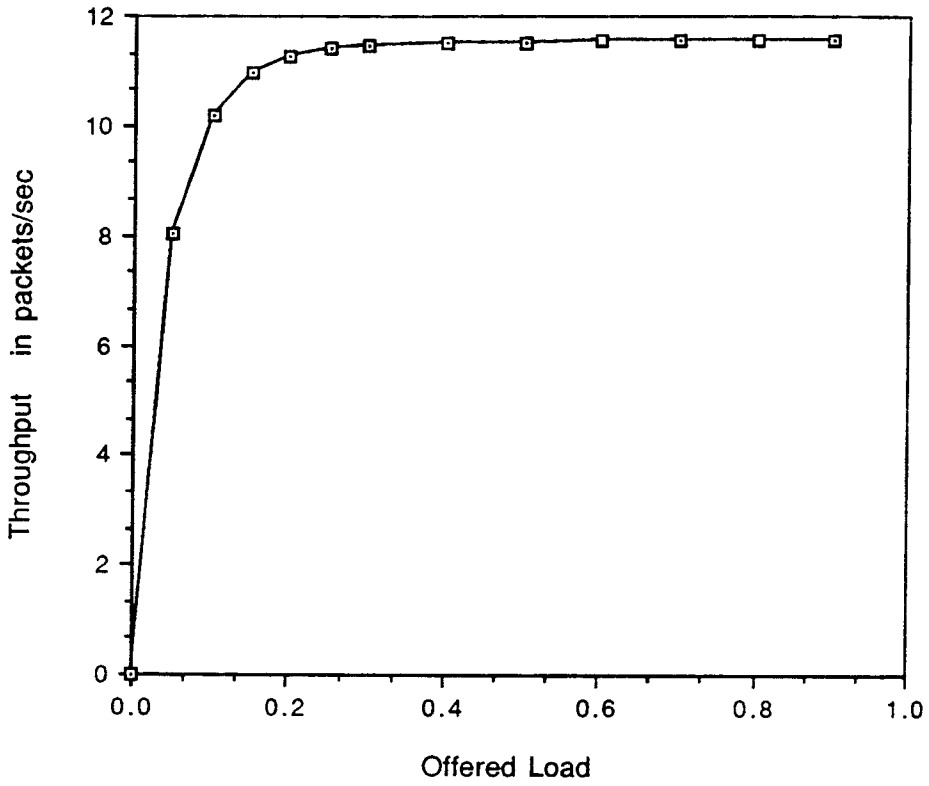

Figure 5.3 Throughput performance of symmetric system

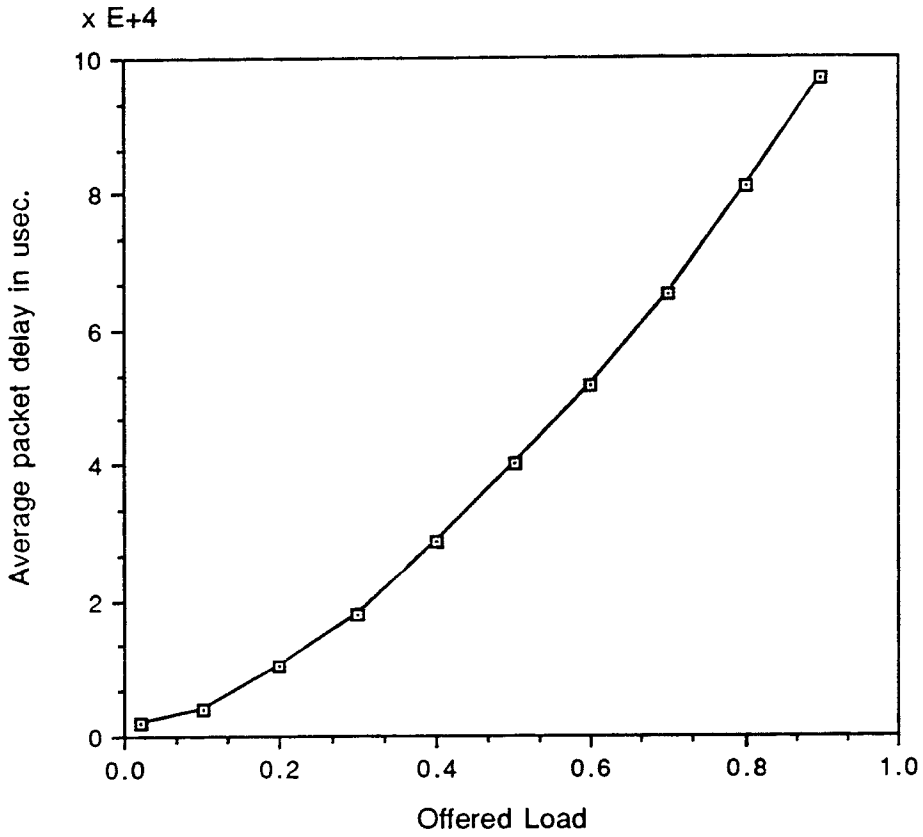

Figure 5.6 Delay performance of asymmetric system

data packets. The token transmission is modeled by transitions t_{ic} and t_{id}. Therefore, it can be seen from Figure 5.3 and Figure 5.4 that the thoughput curve passes through the origin when no packets come to the network.

5.4.2 Delay vs Offered Load

The performance of average packet delay is shown in Figures 5.5 and 5.6 for symmetric and asymmetric systems, respectively. Both delays increase with the offered load.

It can be seen that the delay of the latter is lower than that of the former at the same value of the offered load. Since one of stations has exhaustive service it makes average packet delay become lower for whole network. This result can be analyzed from queuing thoery qualitatively. The average packet delay for the three service type is as follows [24]:

Exhaustive Service

$$D_E \propto \frac{\omega(N-\rho)}{2(1-\rho)} + \frac{\rho\ddot{X}_p}{2\dot{X}_p R(1-\rho)} \tag{0.3}$$

Gated Service

$$D_G \propto \frac{\omega(N+\rho)}{2(1-\rho)} + \frac{\rho\ddot{X}_p}{2\dot{X}_p R(1-\rho)} \tag{0.4}$$

Limited Service

$$D_L \propto \frac{\omega(N+\rho)}{2(1-\rho-N\lambda\omega)} + \frac{\rho\ddot{X}_p}{2\dot{X}_p R(1-\rho-N\lambda\omega)} \tag{0.5}$$

where, $\ddot{X}_p$ is the second moment of packet length, $\dot{X}_p$ is the first moment.

From the corresponding average packet delay expresssions, it can be shown that

$$D_E \leq D_G \leq D_L \tag{0.6}$$

Single service can be considered as a special case of limited service. Therefore, the average packet delay of asymmetric system which has one exhaustive and four single service stations is less than that of symmetric system which has all single service stations.

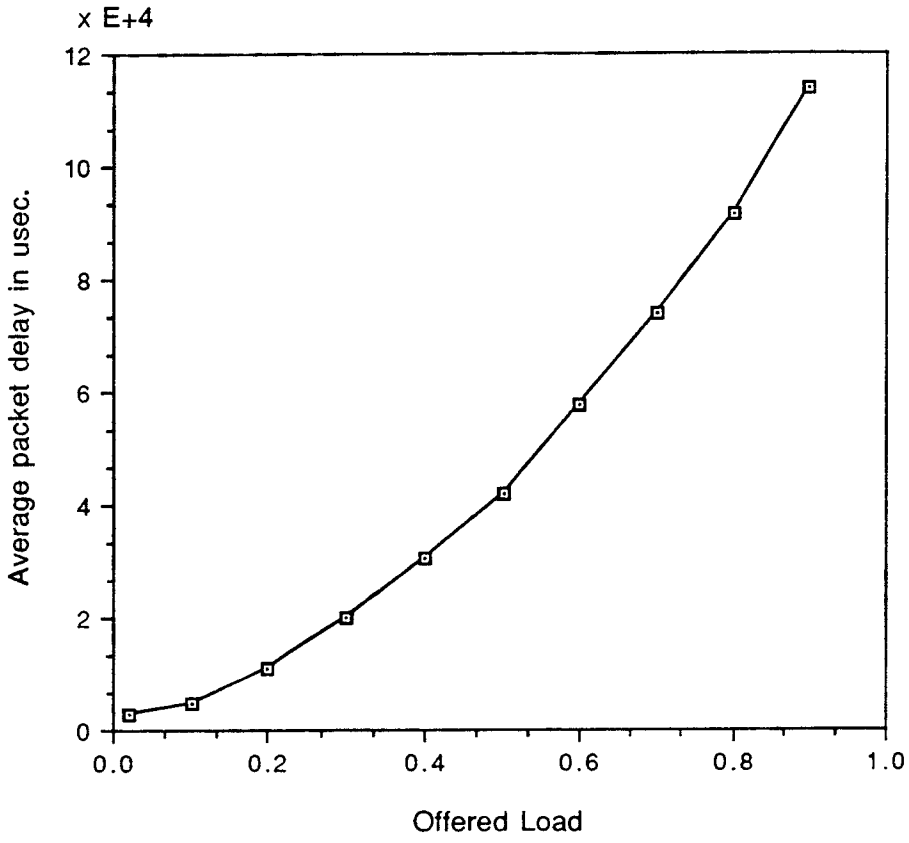

Figure 5.5 Delay performance of symmetric system

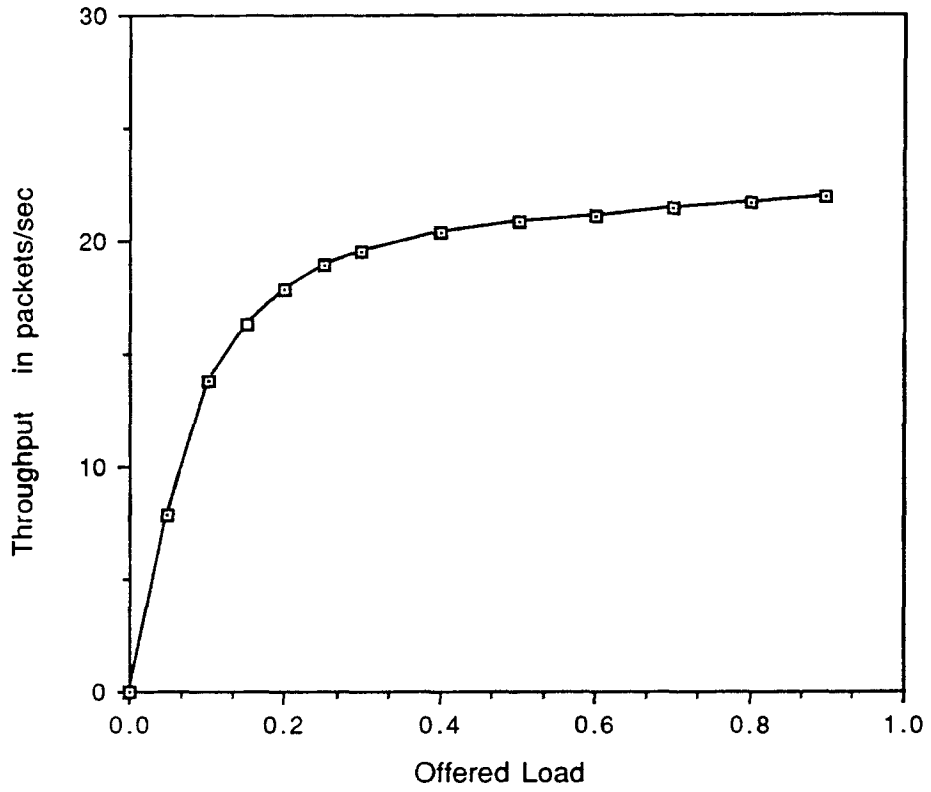

Figure 5.4 Throughput performance of asymmetric system

CHAPTER 6
APPROXIMATION METHODS

This chapter deals with the state space explosion problems in performance analysis of a token bus LAN with more than five stations. An effort is made to develop an effective approximation method for evaluating token bus LAN with any number of stations by reducing the state space.

6.1 Presentation of the Problem

As mentioned above, stochastic Petri nets and extensions are a truly useful model class for representing distributed systems, in general, and local area computer-communication networks, in particular. Among them, Generalized Stochastic Petri Nets (GSPN) allows not only exponentially distributed timing delays but also immediate transitions and probabilitic arcs. GSPN is well suited to modeling and performance of local area networks, as discussed in the last Chapter. However, with the changes of LAN environment, such as increase of buffer size for each or some stations, or station number growth, tranditional model construction and solution methods for these models limit their usefulness, due to the extremely rapid growth of the size of the state space used in model solution.

The experiment presents the state explosion problem as shown in Table 6.1.

6.2 Reduction Method of GSPN

There are many reduction methods which can be used to effectively reduce a large net to a small one only for non-timed Petri nets. However, some rules can be modified to hold true for generalized stochastic Petri nets when temporal/stochastic characteristics are considered. Reducible subnets are selected and analyzed in isolation by

Buffer Size \ Num of Station / Num of state		3	4	5
2	T	135	540	2025
	V	27	108	405
3	T	336	1792	8960
	V	48	256	1280
4	T	675	4500	28125
	V	75	500	3125
5	T	1188	9504	71280
	V	108	864	6480
6	T	1911	17836	156065
	V	147	1372	12005

Table 6.1 State space vs number of stations and buffer size

constructing their associated Petri nets.

6.2.1 Reduction Definitions

Flow equivalent server concept can be used to develop a reduced equivalent net based on the following definitions [24].

Definition 1: $Z' = (P', T', I', O', H', m', F', P'_r)$ is a subnet of $Z = (P, T, I, O, H, m, F, P'_r)$ if $P' \subset P$, $T' \subset T$, and if, $\forall p \in P'$, $t \in T'$, $I'(p,t) = I(p,t)$, $O'(p,t) = O(p,t)$, $H'(p,t) = H(p,t)$, $m'(p) = m(p)$, $F'(t) = F(t)$, and $P'_r(t) = P_r(t)$. We can write $Z' \subset Z$.

Definition 2: Given $Z' \subset Z$, Z' is a p-t subnet/block iff $P^{in} \neq \emptyset$, $P^{out} = \emptyset$, $T^{in} \neq \emptyset$, and $T^{out} \neq \emptyset$. Where, P^{in} is a set of input places; P^{out} is a set of output places; T^{in} is a set of input transitions; and T^{out} is a set of output places.

Definition 3: An associated Petri net of a p-t subnet Z', denoted by $S(Z')$ is a Petri net which links the input and output nodes of Z' based on the token flow directions in Z, where a node is referred to as either a place or a transition.

Definition 4: A GSPN is called an equivalint net, Z'', of a subnet, Z', iff under the same markings, 1) the average time from any input node in Z'' equals to that in Z'; 2) the number of states in $S(Z'')$ is less than that in $S(Z')$; and 3) $S(Z'')$ is so if $S(Z)$ is bounded, live, and reversible.

It is noted that "the same marking" mean that the number of tokens of the corresponding places in Z'' and Z' must be the same. Condition 3 in Definition 4 ensures that the reduced net has the desired qualitative properties.

6.2.2 Subnet Selection

In order to obtain the accurate approximation results, the following rule must be obeyed to select a subnet for a given GSPN.

Selection Rule: 1) The associated net of $Z' = (P', T', I', O', H', m', F', P'_r) \subset Z$ is bounded, live, and reversible; and 2) $\{p : \exists t \in T' - T^{in} \ni I(p,t) \neq 0,$

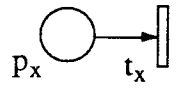

Figure 6.1 A p-t subset

or $t \in T' - T^{out} \ni O(p,t) \neq 0\} = P'$ and $\{t : \exists p \in P' - P^{out} \ni I(p,t) \neq 0$, or $p \in P' - P^{in} \ni O(p,t) \neq 0\} = T'$.

In this rule, the first part ensures that the subnet can be evaluated. In addition, the preservation of the qualitative properties of the reduction will be guaranteed, which implies the reduced original net can be further evaluated if an equivalent net replaces the subnet. The second part ensures that the subnet can be isolated, independent of the rest of Z. This rule ensures the flow equivalent principle.

6.2.3 Equivalent Net Construction

Based on the token flow equivalent concept, the flexibility to select various subnets leads to the variety of equivalent nets. The construction catalog can be classified as the following: *single-input-single-output* case, *single-input-multiple-output* case and *multiple-input-multiple-output* case. In *single-input-single-output* case, there are four types of subnet structures. one of them is called the place to transition subnet, p-t subnet [25] as shown in Figure 6.1. The p-t subnet is selected as a equivalent net in this research because it is suited to the token bus LAN environment. The details will be mentioned in the following section.

6.3 Procedure

As discussed in [25], for a discrete event system, such as token bus networks, a GSPN

Z is modeled and the initial marking is determined. A procedure to derive the results is formulated as follows:

1. If Z can be evaluated with the software packages available, it is done; otherwise,
2. According to the subnet selection rule, identify a subnet Z' while keeping those transitions or places or subnets in Z unchanged if they are of special interests.
3. Construct the equivalent net for this subnet and derive the parameters for equivalent net based on the throughputs on Z':
   (a) Find the maximum numbers of tokens possible in the related places
   (b) Find the throughput by starting from 1's in some places to the maximum numbers or the numbers whose increase will not change the throughputs of the subnet
   (c) Calculate the parameters in the equivalent net

   If $S(Z')$ cannot be evaluated with software packages, either re-select a subnet or select a sub-subnet in $S(Z')$ and continue this procedure.
4. Let Z'' be the net which is the reduced net of Z by replacing Z' with its equivalent net. Let $Z = Z''$, go to the first step.

It is necessary to keep the right size of the subnet since a big subnet itself will be difficult to evaluate though the final net may have a few number of states. The net which satisfies conditions may not exist. Then we must loosen the conditions at the expense of approximation accuracy.

6.4 Application Illustration

The traditional GSPN modeling method for token bus LAN results in the state space explosion problem with the increasing of station number or of buffer size, as mentioned

before. In order to show that GSPN analysis method is still effective even under the condition that token bus LAN has a large number of stations, it is necessary to combine the approximation method with the tranditional GSPN modeling method to solve the state space explosion problem.

The single-service case of token bus LAN with twenty-one stations, has been taken as an example to illustrate the application of the approximation method. The five-station GSPN model is selected as a subnet Z' of twenty-one station GSPN model Z, as shown in Figure 6.2. It is clear that the subnet Z' satisfies the subnet selection rule. Based on the reduction definitions mentioned above, the subnet Z' can be formulated as a single-input-single-output p-t subnet Z'', from p_{1b} through whole subnet to t_{5c} in Figure 6.2. Therefore, the associated reduced equivalent net can be obtained as the same as shown in Figure 6.1. In the GSPN model Z of a twenty-one station token bus LAN, every five-station is simplified as a p-t subnet Z'' so that the reduced GSPN model of Z has been obtained as shown in Figure 6.3.

At the point of network performance, the delay or throughput characteristics of token bus LAN with single service is mainly depend on the token rotation time which is the reverse of the firing rate of t_{id}, $i = 1, 2, ..., 5$ in Figure 6.2. That is the reason why the reduced subset is selected in that way.

The single-station performance evaluation of the reduced twenty-one station GSPN model has been done. The delay and throughput characteristics are shown in Figures 6.4 and 6.5. Compare to the five-station single-service case, the throughput has decreased and the dealy has increased. The state space of reduced twenty-one station model has been much reduced to 11 tangible markings and 1 vanishing markings. This has shown that the approximation method is very powerful in dealing with the state explosion for token bus LAN.

The two outstanding advantages can be obtained from above experiment which has been performed by this approximation method.

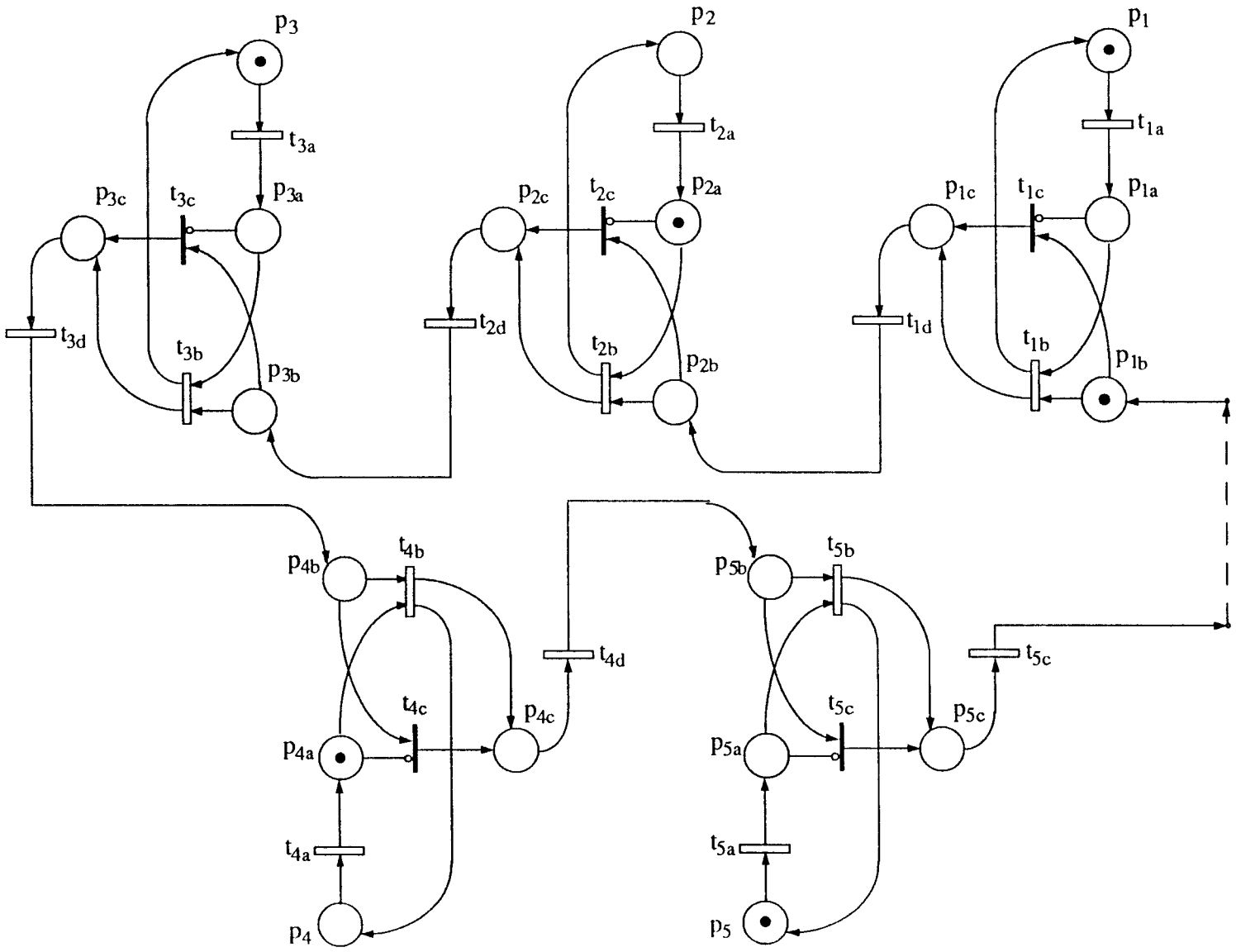

Figure 6.2 Subnet of token bus LAN with twenty-one stations

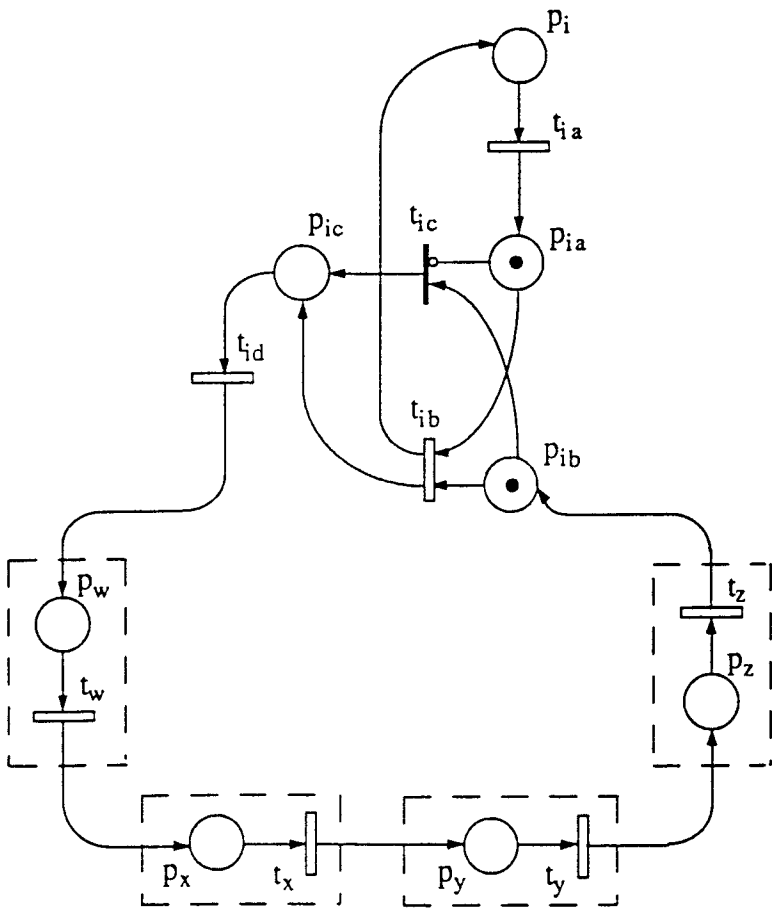

Figure 6.3 Reduced GSPN model of 21-station token bus LAN

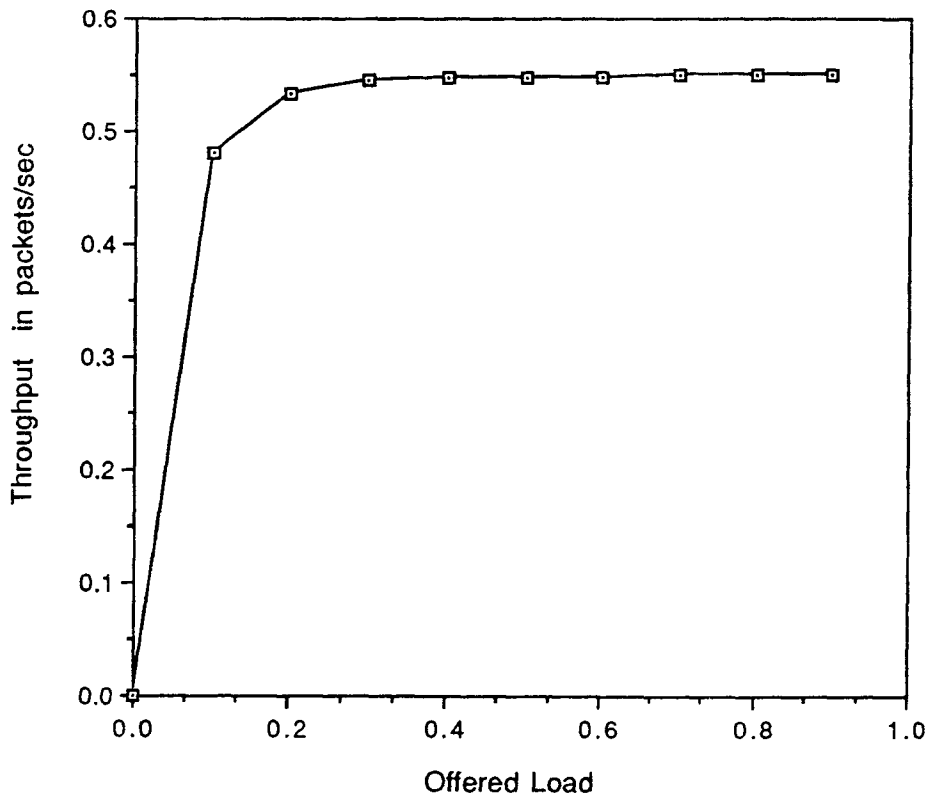

Figure 6.4 Throughput performance of reduced model

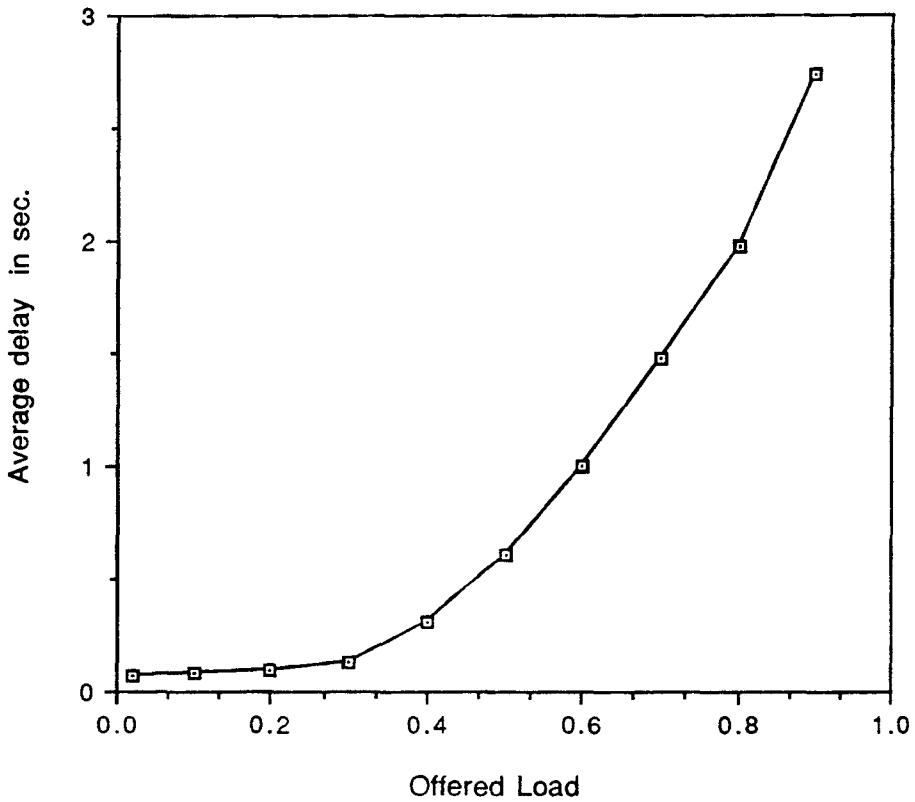

Figure 6.5 delay performance of reduced model

- It is clear that the performance of any single station in the toke bus LAN can be obtained by this method. The whole network performance could be evaluated based on every single station performance.

- This approximation method keeps GSPN effective in modeling and performance of token bus LAN, no matter how large number of stations the LAN has. For example, a token bus LAN with 101 stations can be reduced to 20 equivalent nets with 1 station and then evaluated.

Most LAN interconnections of personal computers have interfaces that can store at most one data packet at a time, thus token passing PC LANs can be modeled by single-service schemes. This research is only for single-service case of token bus LAN. It is possible to extend approximation methods for other service cases of token bus LAN.

CHAPTER 7
CONCLUSION AND FUTURE RESEARCH

In this thesis, GSPN models of token bus LAN with four types of services have been developed. Based on GSPN models, the performance of token bus LAN has been investigated. Athough computer-aided solution of stochastic Petri nets is used in the study, traditional model construction and solution method generate the state space explosion problem when GSPN models become more complicated with the change of network environment. In order to solve this problem, an approximation method of modeling token bus LAN has been applied. Network performance in this case is also investigated.

The GSPN models developed in this thesis are appropriate for performance studies on the basis of a small number of stations. The approximation method of GSPN models is effective in performance and modeling of token bus LAN with a large number of stations. The contribution of this thesis is that the modeling and performance evaluation of token bus LAN can be systematically conducted with GSPN not only for a small number of stations but also for any large number of stations in the single-service case. It can been seen that generalized stochastic Petri nets (GSPN) is a useful analytical tool of local area networks and their usefulness is featured as follows:

1. Exact numerical results of network performance can be obtained in an interactive way;

2. Petri net methods take much less time than simulation methods;

3. Both steady and transient state analysis can be performed for network performance;

4. Very detailed analysis can be concentrated on a part of a large-scale network;

5. Small modifications of the GSPN model can represent big changes of network operations;

6. Performance analysis of asymmetric systems does not create specific problems but they are difficult by using queueing models;

7. Reliability analysis can be performed by using GSPN.

The work in this thesis can be extended in four directions. First, performance of token bus LAN with other service cases, such as, limited and gated services, can be analyzed by using the proposed models in this thesis. Second, the approximation method for asymmetric network with a large number of stations could be developed based on the method in this thesis. The idea is that a) selecting a small number of stations as a subnet and reducing it to an approximate model, b) combining approximate models to form a reduced whole GSPN model of the token bus LAN. Some other approximation methods are presented in [25], [26], [27]. Third, the models of this thesis can be extended to incorporate all priority classes (i.e. class 4, 2. and 0). Performance evaluation of token bus LAN with priority classes is closer to practical applications of token bus LAN. Fourth, reliability analysis of token bus LAN could be performed, which is useful in industrial environment [28], [29]. For reliability analysis of token bus LAN based on GSPN models, two parameters can been chosen to assess the system: the MTTF (mean time to failure) linked to reliability and the mean number of packets transmitted in the network before failure as an index of performability. Failre times are assumed to be exponentially distributed random vriables. The LAN can be considered to be functioning as long as stations are functional.

APPENDIX

APPENDIX: Programs

program 1 {petri_5.c)

```
/* ------------------------------------------------------- */
/*   This program is uesd to evaluate performance of Token  */
/* Bus LAN with single service (symmetric system).          */
/* ------------------------------------------------------- */

#include "user.h"
float Arrivrate;
rate_type Servrate=305.0, Tc=12.0;
int  NumPsg;
int N=5;


parameters() {
        iopt (IOP_PR_FULL_MARK, VAL_YES);
        iopt (IOP_PR_RSET, VAL_YES);
        iopt (IOP_PR_MC, VAL_YES);
        iopt (IOP_PR_RGRAPH, VAL_YES);
        iopt (IOP_PR_PROB, VAL_YES);

        NumPsg = input ("initial tokens of each station:");
        Arrivrate = input ("initial rates of transitions:");

}

net() {
        place("P1");  init("P1",NumPsg);
        place("P1a");
        place("P1b"); init("P1b",1);
        place("P1c");

        trans("t1a");    rateval("t1a",Arrivrate);
        trans("t1b");    rateval("t1b",Servrate);
        trans("t1c");    probval("t1c",1.0); priority("t1c",1);
        trans("t1d");    rateval("t1d",Tc);

        iarc("t1a","P1");        oarc("t1a","P1a");
        iarc("t1b","P1a");       oarc("t1b","P1");
        iarc("t1b","P1b");       oarc("t1b","P1c");
        iarc("t1c","P1b");       oarc("t1c","P1c");
        iarc("t1d","P1c");       oarc("t1d","P2b");
```

```
        harc("t1c","P1a");

/* ********************************************** */

        place("P2");  init("P2", NumPsg);
        place("P2a");
        place("P2b");
        place("P2c");

        trans("t2a");     rateval("t2a",Arrivrate);
        trans("t2b");     rateval("t2b",Servrate);
        trans("t2c");     probval("t2c",1.0); priority("t2c",2);
        trans("t2d");     rateval("t2d",Tc);

        iarc("t2a","P2");          oarc("t2a","P2a");
        iarc("t2b","P2a");         oarc("t2b","P2");
        iarc("t2b","P2b");         oarc("t2b","P2c");
        iarc("t2c","P2b");         oarc("t2c","P2c");
        iarc("t2d","P2c");         oarc("t2d","P3b");

        harc("t2c","P2a");


/* ************************************************/

        place("P3");     init("P3", NumPsg);
        place("P3a");
        place("P3b");
        place("P3c");

        trans("t3a");     rateval("t3a",Arrivrate);
        trans("t3b");     rateval("t3b",Servrate);
        trans("t3c");     probval("t3c",1.0); priority("t3c",3);
        trans("t3d");     rateval("t3d",Tc);

        iarc("t3a","P3");          oarc("t3a","P3a");
        iarc("t3b","P3a");         oarc("t3b","P3");
        iarc("t3b","P3b");         oarc("t3b","P3c");
        iarc("t3c","P3b");         oarc("t3c","P3c");
        iarc("t3d","P3c");         oarc("t3d","P4b");

        harc("t3c","P3a");


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

```
      place("P4");       init("P4", NumPsg);
      place("P4a");
      place("P4b");
      place("P4c");

      trans("t4a");      rateval("t4a",Arrivrate);
      trans("t4b");      rateval("t4b",Servrate);
     trans("t4c");     probval("t4c",1.0); priority("t4c",4);
      trans("t4d");      rateval("t4d",Tc);

      iarc("t4a","P4");                oarc("t4a","P4a");
      iarc("t4b","P4a");               oarc("t4b","P4");
      iarc("t4b","P4b");               oarc("t4b","P4c");
      iarc("t4c","P4b");               oarc("t4c","P4c");
      iarc("t4d","P4c");               oarc("t4d","P5b");

      harc("t4c","P4a");


/*********************************************************\

      place("P5");       init("P5",NumPsg);
      place("P5a");
      place("P5b");
      place("P5c");

      trans("t5a");     rateval("t5a",Arrivrate);
      trans("t5b");     rateval("t5b",Servrate);
     trans("t5c");     probval("t5c",1.0); priority("t5c",5);
      trans("t5d");     rateval("t5d",Tc);

      iarc("t5a","P5");                oarc("t5a","P5a");
      iarc("t5b","P5a");               oarc("t5b","P5");
      iarc("t5b","P5b");               oarc("t5b","P5c");
      iarc("t5c","P5b");               oarc("t5c","P5c");
      iarc("t5d","P5c");               oarc("t5d","P1b");

      harc("t5c","P5a");

}


assert() {return (RES_NOERR);}
ac_init() {}
ac_reach() {fprintf(stderr, "/nThe reahibility graph has been
```

```
generated/n/");}

reward_type ep1() {return(rate("t1a")); }
reward_type ep2() {return(rate("t2a")); }
reward_type ep3() {return(rate("t3a")); }
reward_type ep4() {return(rate("t4a")); }
reward_type ep5() {return(rate("t5a")); }
reward_type pb1() {return(enabled("t1a")); }
reward_type pb2() {return(enabled("t2a")); }
reward_type pb3() {return(enabled("t3a")); }
reward_type pb4() {return(enabled("t4a")); }
reward_type pb5() {return(enabled("t5a")); }

reward_type ef1()
{return(rate("t1b")+rate("t2b")+rate("t3b")+rate("t4b")+
                          rate("t5b")); }

reward_type ef2()
{return(mark("P1a")+mark("P2a")+mark("P3a")+mark("P4a")+
                          mark("P5a")); }

ac_final() {
          FILE    *ff;
          double  x, y, z;

          ff = fopen("anal.res","w");
          fprintf(ff,"\t Offered load = %g\n",
(float)N*Arrivrate/Servrate);
          x = expected(ef1);
          fprintf(ff,"\t UnThroughput = %g\n",x/Arrivrate);
          fprintf(ff,"\t Throughput = %g\n", x);
          y =
expected(ep1)*expected(pb1)+expected(ep2)*expected(pb2)

+expected(ep3)*expected(pb3)+expected(ep4)*expected(pb4)
              +expected(ep5)*expected(pb5);
          z = expected(ef2)/(y*(float)N);
          fprintf(ff,"\t Ave_package_delay = %g\n", z);
          pr_std_average(); }
```

Program 2 (petri_5x.c)

```
/* ------------------------------------------------------- */
/*    This program is used for performance evaluation of   */
/* Asymmetric Token Bus LAN with five stations.            */
/* ------------------------------------------------------- */


float Arrivrate;
rate_type Servrate=305.0, Tc=12.0;
int  NumPsg, BufSize;
int N=5;

parameters() {
         iopt (IOP_PR_FULL_MARK, VAL_YES);
         iopt (IOP_PR_RSET, VAL_YES);
         iopt (IOP_PR_MC, VAL_YES);
         iopt (IOP_PR_RGRAPH, VAL_YES);
         iopt (IOP_PR_PROB, VAL_YES);

         NumPsg = input ("initial tokens of each station:");
         Arrivrate = input ("initial rates of transitions:");
         BufSize = input ("Buffer size:");

}

net() {
        place("P1");  init("P1",2*NumPsg+BufSize);
        place("P1a");
        place("P1b"); init("P1b",1);
        place("P1c");

        trans("t1a");   rateval("t1a",Arrivrate);
        trans("t1b");   rateval("t1b",Servrate);
        trans("t1c");   probval("t1c",1.0); priority("t1c",1);
        trans("t1d");   rateval("t1d",Tc);

        iarc("t1a","P1");         oarc("t1a","P1a");
        iarc("t1b","P1a");        oarc("t1b","P1");
        iarc("t1b","P1b");        oarc("t1b","P1b");
        iarc("t1c","P1b");        oarc("t1c","P1c");
        iarc("t1d","P1c");        oarc("t1d","P2b");

        harc("t1c","P1a");
        mharc("t1a","P1a",BufSize);
/* ********************************************** */
```

```
place("P2");  init("P2", NumPsg);
place("P2a");spell
place("P2b");
place("P2c");

trans("t2a");     rateval("t2a",Arrivrate);
trans("t2b");     rateval("t2b",Servrate);
trans("t2c");     probval("t2c",1.0); priority("t2c",2);
trans("t2d");     rateval("t2d",Tc);

iarc("t2a","P2");          oarc("t2a","P2a");
iarc("t2b","P2a");         oarc("t2b","P2");
iarc("t2b","P2b");         oarc("t2b","P2b");
iarc("t2c","P2b");         oarc("t2c","P2c");
iarc("t2d","P2c");         oarc("t2d","P3b");

harc("t2c","P2a");


/*****************************************************/

place("P3");    init("P3", NumPsg);
place("P3a");
place("P3b");
place("P3c");

trans("t3a");     rateval("t3a",Arrivrate);
trans("t3b");     rateval("t3b",Servrate);
trans("t3c");     probval("t3c",1.0); priority("t3c",3);
trans("t3d");     rateval("t3d",Tc);

iarc("t3a","P3");          oarc("t3a","P3a");
iarc("t3b","P3a");         oarc("t3b","P3");
iarc("t3b","P3b");         oarc("t3b","P3b");
iarc("t3c","P3b");         oarc("t3c","P3c");
iarc("t3d","P3c");         oarc("t3d","P4b");

harc("t3c","P3a");


/*********************************************************/

place("P4");      init("P4", NumPsg);
place("P4a");
place("P4b");
```

```
        place("P4c");

        trans("t4a");       rateval("t4a",Arrivrate);
        trans("t4b");       rateval("t4b",Servrate);
       trans("t4c");     probval("t4c",1.0); priority("t4c",4);
        trans("t4d")        rateval("t4d",Tc);

        iarc("t4a","P4");              oarc("t4a","P4a");
        iarc("t4b","P4a");             oarc("t4b","P4");
        iarc("t4b","P4b");             oarc("t4b","P4b");
        iarc("t4c","P4b");             oarc("t4c","P4c");
        iarc("t4d","P4c");             oarc("t4d","P5b");

        harc("t4c","P4a");


/*********************************************************/

        place("P5");       init("P5",NumPsg);
        place("P5a");
        place("P5b");
        place("P5c");

        trans("t5a");      rateval("t5a",Arrivrate);
        trans("t5b");      rateval("t5b",Servrate);
       trans("t5c");     probval("t5c",1.0); priority("t5c",5);
        trans("t5d");      rateval("t5d",Tc);

        iarc("t5a","P5");              oarc("t5a","P5a");
        iarc("t5b","P5a");             oarc("t5b","P5");
        iarc("t5b","P5b");             oarc("t5b","P5b");
        iarc("t5c","P5b");             oarc("t5c","P5c");
        iarc("t5d","P5c");             oarc("t5d","P1b");

        harc("t5c","P5a");

}


assert() {return (RES_NOERR);}
ac_init() {}
ac_reach() {fprintf(stderr, "/nThe reahibility graph has been
generated/n/");}

reward_type ep1() {return(rate("t1a")); }
reward_type ep2() {return(rate("t2a")); }
```

```
reward_type ep3() {return(rate("t3a")); }
reward_type ep4() {return(rate("t4a")); }
reward_type ep5() {return(rate("t5a")); }
reward_type pb1() {return(enabled("t1a")); }
reward_type pb2() {return(enabled("t2a")); }
reward_type pb3() {return(enabled("t3a")); }
reward_type pb4() {return(enabled("t4a")); }
reward_type pb5() {return(enabled("t5a")); }

reward_type ef1()
{return(rate("t1b")+rate("t2b")+rate("t3b")+rate("t4b")+
                          rate("t5b")); }

reward_type ef2()
{return(mark("P1a")+mark("P2a")+mark("P3a")+mark("P4a")+
                          mark("P5a")); }

ac_final() {
          FILE    *ff;
          double  x, y, z;

          ff = fopen("anal.res","w");
         fprintf(ff,"\t Offered load = %g\n", (float)N*Arrivrate/
Servrate);
          x = expected(ef1);
          fprintf(ff,"\t UnThroughput = %g\n",x/Arrivrate);
          fprintf(ff,"\t Throughput = %g\n", x);
          y =
expected(ep1)*expected(pb1)+expected(ep2)*expected(pb2)

+expected(ep3)*expected(pb3)+expected(ep4)*expected(pb4)
              +expected(ep5)*expected(pb5);
          z = expected(ef2)/(y*(float)N);
          fprintf(ff,"\t Ave_package_delay = %g\n", z);
          pr_std_average(); }
```

Program 3 (petri_5e.c)

```
/* ------------------------------------------------------ */
/*     This program is used for performance evaluation    */
/*  of five-station Token Bus LAN with exhaustive service */
/* ------------------------------------------------------ */


#include "user.h"
float Arrivrate;
rate_type Servrate=305.0, Tc=12.0;
int   NumPsg, BufSize;
int N=5;


parameters() {
         iopt (IOP_PR_FULL_MARK, VAL_YES);
         iopt (IOP_PR_RSET, VAL_YES);
         iopt (IOP_PR_MC, VAL_YES);
         iopt (IOP_PR_RGRAPH, VAL_YES);
         iopt (IOP_PR_PROB, VAL_YES);

         NumPsg = input ("initial tokens of each station:");
         Arrivrate = input ("initial rates of transitions:");
         BufSize = input ("buffer size:");
}

net() {
        place("P1");  init("P1",NumPsg);
        place("P1a");
        place("P1b"); init("P1b",1);
        place("P1c");

        trans("t1a");     rateval("t1a",Arrivrate);
        trans("t1b");     rateval("t1b",Servrate);
        trans("t1c");     probval("t1c",1.0); priority("t1c",1);
        trans("t1d");     rateval("t1d",Tc);

        iarc("t1a","P1");         oarc("t1a","P1a");
        iarc("t1b","P1a");        oarc("t1b","P1");
        iarc("t1b","P1b");        oarc("t1b","P1b");
        iarc("t1c","P1b");        oarc("t1c","P1c");
        iarc("t1d","P1c");        oarc("t1d","P2b");
        mharc("t1a","P1a",BufSize);
        harc("t1c","P1a");
 /* ********************************************** */
```

```
        place("P2");  init("P2", NumPsg);
        place("P2a");
        place("P2b");
        place("P2c");

        trans("t2a");      rateval("t2a",Arrivrate);
        trans("t2b");      rateval("t2b",Servrate);
        trans("t2c");      probval("t2c",1.0); priority("t2c",2);
        trans("t2d");      rateval("t2d",Tc);

        iarc("t2a","P2");          oarc("t2a","P2a");
        iarc("t2b","P2a");         oarc("t2b","P2");
        iarc("t2b","P2b");         oarc("t2b","P2b");
        iarc("t2c","P2b");         oarc("t2c","P2c");
        iarc("t2d","P2c");         oarc("t2d","P3b");
        mharc("t2a","P2a",BufSize);
        harc("t2c","P2a");
    /* ************************************************/

        place("P3");    init("P3", NumPsg);
        place("P3a");
        place("P3b");
        place("P3c");

        trans("t3a");      rateval("t3a",Arrivrate);
        trans("t3b");      rateval("t3b",Servrate);
        trans("t3c");      probval("t3c",1.0); priority("t3c",3);
        trans("t3d");      rateval("t3d",Tc);

        iarc("t3a","P3");          oarc("t3a","P3a");
        iarc("t3b","P3a");         oarc("t3b","P3");
        iarc("t3b","P3b");         oarc("t3b","P3b");
        iarc("t3c","P3b");         oarc("t3c","P3c");
        iarc("t3d","P3c");         oarc("t3d","P4b");
        mharc("t3a","P3a",BufSize);
        harc("t3c","P3a");
/*****************************************************/

        place("P4");      init("P4", NumPsg);
        place("P4a");
        place("P4b");
        place("P4c");

        trans("t4a");        rateval("t4a",Arrivrate);
        trans("t4b");        rateval("t4b",Servrate);
```

```
        trans("t4c");        probval("t4c",1.0); priority("t4c",4);
        trans("t4d");          rateval("t4d",Tc);

        iarc("t4a","P4");                  oarc("t4a","P4a");
        iarc("t4b","P4a");                 oarc("t4b","P4");
        iarc("t4b","P4b");                 oarc("t4b","P4b");
        iarc("t4c","P4b");                 oarc("t4c","P4c");
        iarc("t4d","P4c");                 oarc("t4d","P5b");
        mharc("t4a","P4a",BufSize);
        harc("t4c","P4a");
 /*************************************************************/

        place("P5");       init("P5",NumPsg);
        place("P5a");
        place("P5b");
        place("P5c");

        trans("t5a");         rateval("t5a",Arrivrate);
        trans("t5b");         rateval("t5b",Servrate);
        trans("t5c");        probval("t5c",1.0); priority("t5c",5);
        trans("t5d");         rateval("t5d",Tc);

        iarc("t5a","P5");                  oarc("t5a","P5a");
        iarc("t5b","P5a");                 oarc("t5b","P5");
        iarc("t5b","P5b");                 oarc("t5b","P5b");
        iarc("t5c","P5b");                 oarc("t5c","P5c");
        iarc("t5d","P5c");                 oarc("t5d","P1b");
        mharc("t5a","P5a",BufSize);
        harc("t5c","P5a");
      }

assert() {return (RES_NOERR);}
ac_init() {}
ac_reach() {fprintf(stderr, "/nThe reahibility graph has been
generated/n/");}

reward_type ep1() {return(rate("t1a")); }
reward_type ep2() {return(rate("t2a")); }
reward_type ep3() {return(rate("t3a")); }
reward_type ep4() {return(rate("t4a")); }
reward_type ep5() {return(rate("t5a")); }
reward_type pb1() {return(enabled("t1a")); }
reward_type pb2() {return(enabled("t2a")); }
reward_type pb3() {return(enabled("t3a")); }
reward_type pb4() {return(enabled("t4a")); }
reward_type pb5() {return(enabled("t5a")); }
```

```
reward_type ef1()
{return(rate("t1b")+rate("t2b")+rate("t3b")+rate("t4b")+
                          rate("t5b")); }

reward_type ef2()
{return(mark("P1a")+mark("P2a")+mark("P3a")+mark("P4a")+
                          mark("P5a")); }

ac_final() {
          FILE    *ff;
          double  x, y, z;

          ff = fopen("anal.res","w");
        fprintf(ff,"\t Offered load = %g\n", (float)N*Arrivrate/
Servrate);
          x = expected(ef1);
          fprintf(ff,"\t UnThroughput = %g\n",x/Arrivrate);
          fprintf(ff,"\t Throughput = %g\n", x);
          y =
expected(ep1)*expected(pb1)+expected(ep2)*expected(pb2)

+expected(ep3)*expected(pb3)+expected(ep4)*expected(pb4)
              +expected(ep5)*expected(pb5);
          z = expected(ef2)/(y*(float)N);
          fprintf(ff,"\t Ave_package_delay = %g\n", z);
          pr_std_average(); }
```

Program 4 (petri_5g.c)

```
/* ------------------------------------------------------ */
/*     This program is used for performance evaluation    */
/*  of five-station Token Bus LAN with gated service.     */
/* ------------------------------------------------------ */


#include "user.h"
float Arrivrate;
rate_type Servrate=305.0, Tc=12.0;
int   NumPsg, BufSize;
int N=5;


parameters() {
         iopt (IOP_PR_FULL_MARK, VAL_YES);
         iopt (IOP_PR_RSET, VAL_YES);
         iopt (IOP_PR_MC, VAL_YES);
         iopt (IOP_PR_RGRAPH, VAL_YES);
         iopt (IOP_PR_PROB, VAL_YES);

         NumPsg = input ("initial tokens of each station:");
         Arrivrate = input ("initial rates of transitions:");
         BufSize = input ("buffer size:");
}

net() {
        place("P1");  init("P1",NumPsg);
        place("P1a");
        place("P1b"); init("P1b",1);
        place("P1c");

        trans("t1a");     rateval("t1a",Arrivrate);
        trans("t1b");     rateval("t1b",Servrate);
       trans("t1c");    probval("t1c",1.0); priority("t1c",1);
        trans("t1d");     rateval("t1d",Tc);

        iarc("t1a","P1");         oarc("t1a","P1a");
        iarc("t1b","P1a");        oarc("t1b","P1");
        iarc("t1b","P1b");        oarc("t1b","P1b");
        iarc("t1c","P1b");        oarc("t1c","P1c");
        iarc("t1d","P1c");        oarc("t1d","P2b");
        mharc("t1a","P1a",BufSize);
        harc("t1c","P1a");
```

```
        harc("t1a","P1b");
/* ********************************************** */

        place("P2");  init("P2", NumPsg);
        place("P2a");
        place("P2b");
        place("P2c");

        trans("t2a");      rateval("t2a",Arrivrate);
        trans("t2b");      rateval("t2b",Servrate);
      trans("t2c");    probval("t2c",1.0); priority("t2c",2);
        trans("t2d");      rateval("t2d",Tc);

        iarc("t2a","P2");          oarc("t2a","P2a");
        iarc("t2b","P2a");         oarc("t2b","P2");
        iarc("t2b","P2b");         oarc("t2b","P2b");
        iarc("t2c","P2b");         oarc("t2c","P2c");
        iarc("t2d","P2c");         oarc("t2d","P3b");
        mharc("t2a","P2a",BufSize);
        harc("t2c","P2a");
        harc("t2a","P2b");

/* ************************************************/

        place("P3");    init("P3", NumPsg);
        place("P3a");
        place("P3b");
        place("P3c");

        trans("t3a");      rateval("t3a",Arrivrate);
        trans("t3b");      rateval("t3b",Servrate);
      trans("t3c");    probval("t3c",1.0); priority("t3c",3);
        trans("t3d");      rateval("t3d",Tc);

        iarc("t3a","P3");          oarc("t3a","P3a");
        iarc("t3b","P3a");         oarc("t3b","P3");
        iarc("t3b","P3b");         oarc("t3b","P3b");
        iarc("t3c","P3b");         oarc("t3c","P3c");
        iarc("t3d","P3c");         oarc("t3d","P4b");
        mharc("t3a","P3a",BufSize);
        harc("t3c","P3a");
        harc("t3a","P3b");

/*********************************************************/

        place("P4");       init("P4", NumPsg);
```

```
        place("P4a");
        place("P4b");
        place("P4c");

        trans("t4a");       rateval("t4a",Arrivrate);
        trans("t4b");       rateval("t4b",Servrate);
       trans("t4c");     probval("t4c",1.0); priority("t4c",4);
        trans("t4d");       rateval("t4d",Tc);

        iarc("t4a","P4");              oarc("t4a","P4a");
        iarc("t4b","P4a");             oarc("t4b","P4");
        iarc("t4b","P4b");             oarc("t4b","P4b");
        iarc("t4c","P4b");             oarc("t4c","P4c");
        iarc("t4d","P4c");             oarc("t4d","P5b");
        mharc("t4a","P4a",BufSize);
        harc("t4c","P4a");
        harc("t4a","P4b");

/*************************************************************/

        place("P5");       init("P5",NumPsg);
        place("P5a");
        place("P5b");
        place("P5c");

        trans("t5a");       rateval("t5a",Arrivrate);
        trans("t5b");       rateval("t5b",Servrate);
       trans("t5c");     probval("t5c",1.0); priority("t5c",5);
        trans("t5d");       rateval("t5d",Tc);

        iarc("t5a","P5");              oarc("t5a","P5a");
        iarc("t5b","P5a");             oarc("t5b","P5");
        iarc("t5b","P5b");             oarc("t5b","P5b");
        iarc("t5c","P5b");             oarc("t5c","P5c");
        iarc("t5d","P5c");             oarc("t5d","P1b");
        mharc("t5a","P5a",BufSize);
        harc("t5c","P5a");
        harc("t5a","P5b");
}


assert() {return (RES_NOERR);}
ac_init() {}
ac_reach() {fprintf(stderr, "/nThe reahibility graph has been
generated/n/");}
```

```
reward_type ep1() {return(rate("t1a")); }
reward_type ep2() {return(rate("t2a")); }
reward_type ep3() {return(rate("t3a")); }
reward_type ep4() {return(rate("t4a")); }
reward_type ep5() {return(rate("t5a")); }
reward_type pb1() {return(enabled("t1a")); }
reward_type pb2() {return(enabled("t2a")); }
reward_type pb3() {return(enabled("t3a")); }
reward_type pb4() {return(enabled("t4a")); }
reward_type pb5() {return(enabled("t5a")); }

reward_type ef1()
{return(rate("t1b")+rate("t2b")+rate("t3b")+rate("t4b")+
                          rate("t5b")); }

reward_type ef2()
{return(mark("P1a")+mark("P2a")+mark("P3a")+mark("P4a")+
                          mark("P5a")); }

ac_final() {
          FILE    *ff;
          double  x, y, z;

          ff = fopen("anal.res","w");
          fprintf(ff,"\t Offered load = %g\n",
(float)N*Arrivrate/Servrate);
          x = expected(ef1);
          fprintf(ff,"\t UnThroughput = %g\n",x/Arrivrate);
          fprintf(ff,"\t Throughput = %g\n", x);
          y =
expected(ep1)*expected(pb1)+expected(ep2)*expected(pb2)

+expected(ep3)*expected(pb3)+expected(ep4)*expected(pb4)
              +expected(ep5)*expected(pb5);
          z = expected(ef2)/(y*(float)N);
          fprintf(ff,"\t Ave_package_delay = %g\n", z);
          pr_std_average(); }
```

Program 5 (petri_21.c)

```
/* ------------------------------------------------------ */
/*     This programe is used for performance evaluation   */
/*  of twenty-one-station Token Bus LAN with single       */
/*  service.                                              */
/* ------------------------------------------------------ */


#include "user.h"
float Arrivrate, Scanrate;
rate_type Servrate=305.0, Tc=12.0;
int  NumPsg;
int N=21;


parameters() {
         iopt (IOP_PR_FULL_MARK, VAL_YES);
         iopt (IOP_PR_RSET, VAL_YES);
         iopt (IOP_PR_MC, VAL_YES);
         iopt (IOP_PR_RGRAPH, VAL_YES);
         iopt (IOP_PR_PROB, VAL_YES);

         NumPsg = input ("initial tokens of each station:");
         Arrivrate = input ("initial rates of transitions:");
         Scanrate = input ("Scan rate:");

}

net() {
        place("Pi");  init("Pi",NumPsg);
        place("Pia");
        place("Pib"); init("Pib",1);
        place("Pic");

        place("Pw");
        place("Px");
        place("Py");
        place("Pz");

        trans("tia");     rateval("tia",Arrivrate);
        trans("tib");     rateval("tib",Servrate);
        trans("tic");     probval("tic",1.0); priority("tic",1);
```

```
        trans("tid");      rateval("tid",Tc);


        trans("tw");       rateval("tw",Scanrate);
        trans("tx");       rateval("tx",Scanrate);
        trans("ty");       rateval("ty",Scanrate);
        trans("tz");       rateval("tz",Scanrate);

        iarc("tia","Pi");          oarc("tia","Pia");
        iarc("tib","Pia");         oarc("tib","Pi");
        iarc("tib","Pib");         oarc("tib","Pic");
        iarc("tic","Pib");         oarc("tic","Pic");
        iarc("tid","Pic");         oarc("tid","Pw");
        harc("tic","Pia");

        iarc("tw","Pw");           oarc("tw","Px");
        iarc("tx","Px");           oarc("tx","Py");
        iarc("ty","Py");           oarc("ty","Pz");
        iarc("tz","Pz");           oarc("tz","Pib");

}


assert() {return (RES_NOERR);}

ac_init() {}
ac_reach() {fprintf(stderr, "/nThe reahibility graph has been
generated/n/");}

reward_type ep1() {return(rate("tia")); }
reward_type pb1() {return(enabled("tia")); }
reward_type ef1() {return(rate("tib")); }
reward_type ef2() {return(mark("Pia")); }

ac_final() {
          FILE    *ff;
          double  x, y, z;

          ff = fopen("Anal.res","a");
          fprintf(ff,"\t Offered load = %g\n",
(float)N*Arrivrate/Servrate);

          x = expected(ef1);
          fprintf(ff,"\t UnThroughput = %g\n",x/Arrivate);
          fprintf(ff,"\t Throughput = %g\n", x);
```

```
y = expected(ep1)*expected(pb1);
z = expected(ef2)/y;
fprintf(ff,"\t Ave_package_delay = %g\n", z);
fprintf(ff,"\n");
pr_std_average(); }
```

BIBLIOGRAPHY

1. ANSI/IEEE Standard 802.4, *Token-Passing Bus Access Method*, New York: IEEE Press (1985).

2. Brooks, C. A., and O. C. Yue. " Effect of the token holding timer on MAP performance." Proc. Infocom'89. (1989): 342-344.

3. Ibe, O. C., and X. Cheng. "Approximate analysis of asymmetric single-servicee token-passing systems." *IEEE Trans. Commun.* 37 (1989): 572-577.

4. Marsan, M. A., G. conte, and G. Balbo. "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems." ACM Trans. Comput. Syst. 2 (1984): 93-122.

5. *Proceedings of the First International Workshop on Timed Petri Nets.* New York: IEEE Press (1985).

6. Ciardo, G., J. Muppala, and K. S. Trivedi. "SPNP: Stochastic Petri net Package." in *Proc. 3rd Int. Workshop Petri Net and Performance Models*, Kyoto, Japan (1989): 142-151.

7. Ciardo, G., and J. K. Muppala. *Manual for the SPNP Package.* Duke University, Durham (1990).

8. Peterson, J. L. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, NJ: Prentice Hall (1981).

9. Oliver, C., and K. S. Trivedi. "Stochastic Petri net models of polling systems." *IEEE J. on Select Area Commu.* 8 (1990): 1649-1657.

10. Guo, D., F. DiCiesare, and M. C. Zhou. "Moment generating function approach for performance evaluation of extended stochastic Petri nets." in *Proc. IEEE Int. Conf. Rob. & Aut.*, Sacramento, CA (1991): 1309-1314.

11. Zhou, M. C., and F. DiCesare. "Modeling buffers in automated manufacturing systems using Petri nets." *Rensselaer's Second International Conference on Computer-Integrated Manufacturing*, Troy, NY (1990): 265-272.

12. Zhou, M. C., K. McDermott, P. A. Patel, and T. Tang. "Construction of Petri net based mathematical models of an FMS cell." *Proc. of IEEE International Conf. on Syst., Man, and Cybernetics.* Chariottesville, Virginia (1991): 367-372.

13. Murata, T. "Petri nets: properties, analysis and application." *Proc. of the IEEE.* 77(4) (1989): 541-579.

14. Diaz, M. "Modeling and analysis of communication and cooperation protocols using Petri net based models." *Comput. Networks.* 6 (1982): 419-441.

15. Sajkowski, M. "Protocol verification using discrete-event models." *Discrete Event Systems: Models & Applications.* New York, Springer-Verlag, (1989): 100-113.

16. Diaz, M., and P. Azema. "Petri net based models for the specification and validation of protocols." in *Lecture Notes in Computer Science*, New York: Springer-verlag. 188 (1985): 101-121.

17. Juanole, G., and C. Faure. "On gateway for internetworking through ISDN: architecture and formal modelling with Petri nets." *Proc. Infocom'89.* (1989): 458-467.

18. Mukherjee, A., L. H. Landweber, and J. C. Strikwerda. "Simultaneous analysis of flow and error control strategies with congestion-dependent errors." *ACM Performance Evaluation Review.* 18 (1990): 86-95.

19. Gressier, E. "A stochastic Petri net model for Ethernet." in *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, Luly 1-3 (1985): 296-303.

20. Marsan, M. A., and V. Signore. "Timed Petri net performance models of fiber optics LAN architectures." in *Proc. Int. Workshop Petri Nets Perform. Models*, Madison, WI (1987): 66-74.

21. Marsan, M. A., G. Chiola, and A. Fumagali. "An accurate performance model of CSMA/CD bus LAN." in *Lecture Notes in Compter Science*, Advances in Petri Nets. New York: Springer-Verlag. 266 (1987): 146-161.

22. Tanenbaum, A. S. *Computer Networks*, Second edition. Prentice Hall, Englewood Cliffs, N.J. (1988).

23. Robbi, A. D., and R. Kurnool. "Token bus LAN performance: modeling and simulation." *Proceedings 21st Annual Pittsburgh Conference on Modeling and Simulation* (1990).

24. Pimentel, J. R. *Communication Networks for Manufacturing*, Prentice Hall, Englewood Cliffs, N.J. (1990).

25. Ma, J. M., and M. C. Zhou. "Performance evaluation of discrete event systems via stepwise reduction and approximation of stochastic Petri nets." Working paper, New Jersey Institute of Technology (1992).

26. Sanders, W. H., and J. F. Meyer. "Reduced basemodel construction methods for stochastic activity networks." *IEEE J. on Select Area in Commu.* 9 (1991): 25-35.

27. Ciardo, G., and K. S. Trivedi. "A decomposition approach for stochastic Petri net models." in *Proc. of the 4th Int. Workshop on Petri Nets and Performance Models*, Melbourne, Australia (1991): 74-83.

28. Catania, V., L. Milazzo, and A. Puliafto. "Enhancing reliability in an aindustrial LAN: design and performability evaluation." *IEEE Trans. on Industrial Electronics*, 37 (1990): 433-441.

29. Casale, S., V.Catania, and L. Vita. " Fault tolerance increasing in token ring LAN's." in *Proc. IEEE 13th Conf. Loacl Comput. Networks*, Minneapolis, MN (1988).