Petri Net Simulation of Computer Communications Systems

by

David B. Shoat, BSc

Thesis submitted for the Degree of
Master of Science
of the University of Glasgow
June 1991

Acknowledgements

I would like to thank the two supervisors of this work, Professor Peter Macfarlane and Dr. Lewis McKenzie, for their many useful comments and criticisms during the writing of this thesis. I am also indebted to Professor Macfarlane for his support and encouragement throughout the duration of the work. In addition, I would like to thank Dr. Mark Watts, who supervised the design and construction of the CART electrocardiographs and also the installation of the Broadband network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Summary

This thesis presents the design of a simulation method for computer networking systems using timed Petri Nets along with the development of a simulation program based on this method. The use of this simulation program to evaluate the performance of a number of communications systems is described and its suitability as a general-purpose performance evaluation tool for networking systems is discussed.

The development of the simulation method arose from the installation within Glasgow Royal Infirmary of a communications network for the transmission of digitised electrocardiograms. This was part of an ongoing project to develop a program for the automatic analysis of electrocardiograms by computer. The networking methodology currently in use on this network was developed using the simulation program.

The aim in designing the simulator was to develop a tool which would be of use in simulating as wide a range of systems as possible and to this end a class of Petri net was developed which had a wide range of simulation capabilities. It was further intended that any extensions to the "Classical" Place/Transition net model should be made in such a way that the simplicity of the original model was preserved in as large a measure as possible.

During the course of this work, various extensions to the basic Place/Transition net model were indeed made in order to increase the power of the simulation program. Some of these extensions are believed to be unique to the present program, in particular the use of a timed-place scheme. Most current work on timed Petri nets concentrates on timed-transition models as these are easier to implement and analyse; this thesis seeks to show that a timed-place model is viable as a simulation tool and is in many ways preferable to the timed-transition model.

To test the range of the simulator, two extra simulation experiments were undertaken in addition to the simulation of the Royal Infirmary network, the first being the evaluation of a simple queueing system and the second the simulation of an Ethernet network.

The simulation of the Ethernet network also tested the capability of the timed-place model to handle stochastic Petri net simulations, a type of simulation which is being used increasingly to model computer and networking systems and which is currently dominated by timed-transition models.

Descriptions of all three simulation projects are presented along with an analysis of the results of each.

Chapter 1. Introduction

Petri Nets

The Petri Net is a graphical modeling tool, first developed by C.A. Petri in 1962[1] and whose use was to model the flow of control and information in automated systems. Petri nets have been found to be a useful modeling technique primarily because of their ability to model concurrency and synchronisation, thus making them ideal for the analysis of such systems as communications protocols and computer operating systems. Although the bulk of the work in Petri net research to date has been in the development of analytical techniques to predict system behaviour from a net model, the method is also well suited to simulation studies because a Petri net contains a simple set of *execution rules* which allow any given net to be simulated by a computer program.

Detailed explanations of the theory of Petri nets are given in the introductory texts on the subject by Peterson[2] and Reisig[3]. However a short description of the operation of Place/Transition nets, upon which the present simulation method is based, is given here along with an outline of other developments in Petri nets which are of relevance to the present model.

Place/transition nets

An example of a simple Place/Transition (P/T) net is shown in Figure 1. A P/T net is a graph consisting of two types of node, PLACES (represented graphically by circles) and TRANSITIONS (represented by squares) with directed arcs joining places to transitions and transitions to places. Each arc may be assigned a numeric *WEIGHT* and each place a numeric *CAPACITY*. A weight is represented graphically by writing the value of the weight next to the arc, while the place capacity is normally written next to the place in question. If no weight or capacity is indicated then a default value of one is assumed. Each place may contain a number of *TOKENS* (represented graphically by small dots drawn within the place) which is less than or equal to the capacity of the place. The disposition of tokens in the net is known as the *MARKING* of the net.

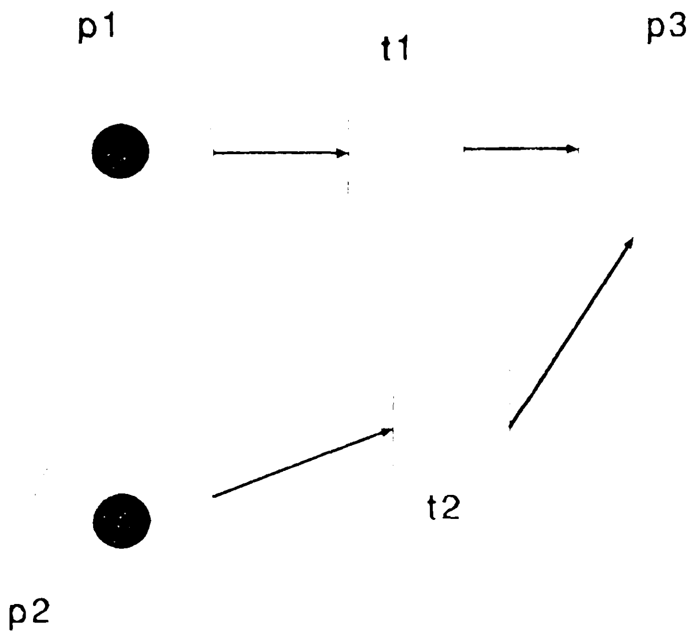

FIGURE 1. A simple Place/Transition Net

A transition may fire at any time if it is *ENABLED* and this occurs if the number of tokens in each of the input places is greater than or equal to the weight of the arc joining the input place to the transition *AND* the number of tokens to be deposited in each output place does not exceed the capacity of the output place. When a transition fires, a number of tokens are removed from each of its input places depending on the weight of the arc joining each input place to the transition and a number of new tokens are added to each output place, again depending on the weights of the appropriate arcs. Execution of the net continues in this way, with the tokens being moved around the net according to the rules given below. This execution pattern is somewhat akin to the moving of pieces in a board game and the execution of a net is sometimes referred to as "playing the token game."

Figure 2. shows the effect of firing transition T2 in the net of Figure 1. The execution rules for P/T nets are summarised formally in the following section.

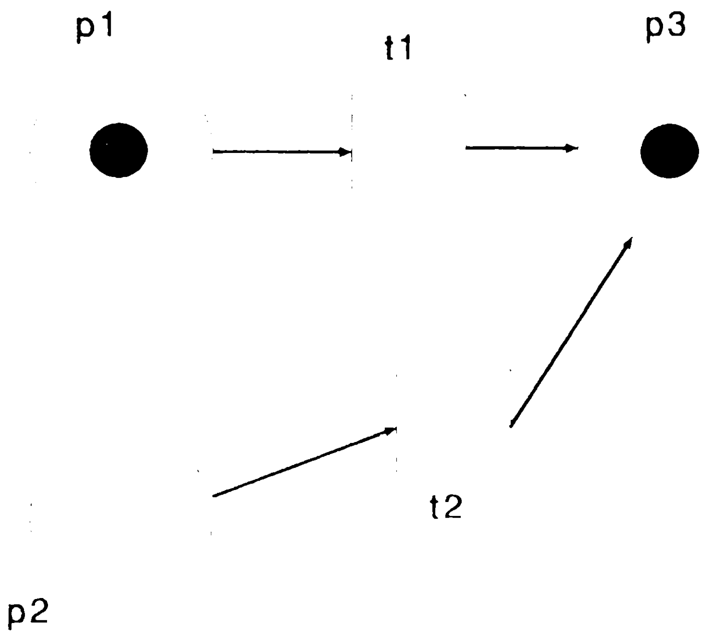

FIGURE 2. The net of Figure 1 after firing T2

Definitions

A Place/Transition net is a 6-tuple N = (P,T,F,K,M,W) where

$\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$ is the set of places of the net,

$\mathbf{T} = \{t_1, t_2, \ldots, t_m\}$ is the set of transitions of the net,

$\mathbf{F} \subseteq (\mathbf{P} \times \mathbf{T}) \cup (\mathbf{T} \times \mathbf{P})$ is the *flow relation*, or the set of input and output arcs of the net,

$\mathbf{K} = \{K_1, K_2, \ldots, K_i\}$ is the set of place capacities,

$\mathbf{M} = \{\mu_1, \mu_2, \ldots, \mu_j\}$ is the set of place markings,

$\mathbf{W}: \mathbf{F} \rightarrow \mathbf{N} \backslash \{0\}$ is the weight function which associates a numeric weight with each arc of the net.

The *preset* $^\bullet t$ of a transition is the set of all places which act as inputs to the transition.

The *postset* $t^\bullet$ of a transition is the set of all places which act as outputs from the transition.

Markings

Let **N** be a P/T net and let $p \in \mathbf{P}$ and $t \in \mathbf{T}$ be a place and transition belonging to the net. Then

A mapping $\mathbf{M}_N: \mathbf{P} \rightarrow N \cup \{\Omega\}$ is a marking of **N**

iff $\mu_p \leq \mathbf{K}_p \ \forall p \in \mathbf{P}$.

In this context, the symbol Ω is used to denote an arbitrarily large positive integer. It is used when the number of tokens in a place is large, but where the actual value is of no consequence. Ω is treated as an integer for analytical purposes.

If $\mathbf{M}_N$ is a marking of **N** then

A transition $t \in \mathbf{T}$ is M-enabled

iff $\mu_p \geq \mathbf{W}(p,t)$ $\qquad \forall p \in {}^\bullet t$

and $\mu_p \leq \mathbf{K}_p - \mathbf{W}(t,p)$ $\qquad \forall p \in {}^\bullet t$

An M-enabled transition may *fire*, yielding a *follower marking* **M′** of **M** such that $\forall p \in \mathbf{P}$:

$\mathbf{M}'_p = \mathbf{M}_p - \mathbf{W}(p,t)$ $\qquad$ iff $p \in {}^\bullet t \setminus t^\bullet$

$\mathbf{M}'_p = \mathbf{M}_p + \mathbf{W}(t,p)$ $\qquad$ iff $p \in t^\bullet \setminus {}^\bullet t$

$\mathbf{M}'_p = \mathbf{M}_p - \mathbf{W}(p,t) + \mathbf{W}(t,p)$ $\qquad$ iff $p \in {}^\bullet t \cap t^\bullet$

$\mathbf{M}'_p = \mathbf{M}_p$ $\qquad$ otherwise.

If t_1 and t_2 are transitions such that $t_1, t_2 \in \mathbf{T}$

and $\mathbf{C} = {}^\bullet t_1 \cap {}^\bullet t_2 \neq \{\emptyset\}$

and $m_p = 1 \quad \forall p \in \mathbf{C}$

then t_1 and t_2 are said to be in *conflict*.

Modelling with P/T nets

When using P/T nets to model a system, the places of the net are normally used to denote conditions, i.e. elements of the state of the system while the transitions represent events, i.e. transitions from one state to another. The condition associated with a given place is held to be satisfied if there are one or more tokens in the place, so that the disposition of tokens in the places of the net gives the overall state of the system. For example, Figure 3 shows a Petri net representation of a simple job-processing system. The token in P1 indicates that the processor is idle. To represent a job arriving for service, a token would be placed in P2, as shown in Figure 4. In this case transition T1 would be enabled and would fire, leaving places P1 and P2 empty and one token in P3, denoting that the job was active. This situation is shown in Figure 5 and leads in turn to transition T2 becoming enabled. Figure 6. shows the state of the net after firing T2, indicating that the job has been completed and the processor is idle once again.

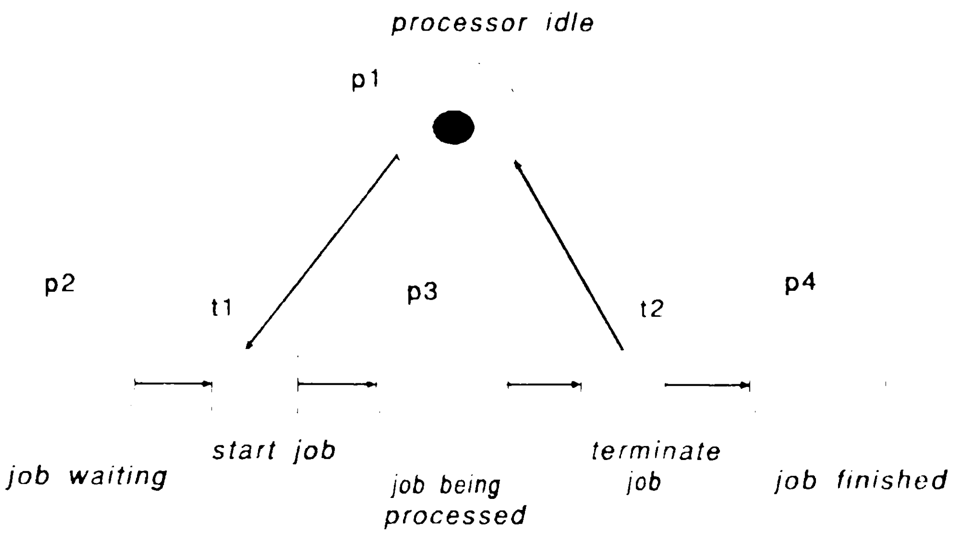

FIGURE 3. P/T net representation of a job-processing system.

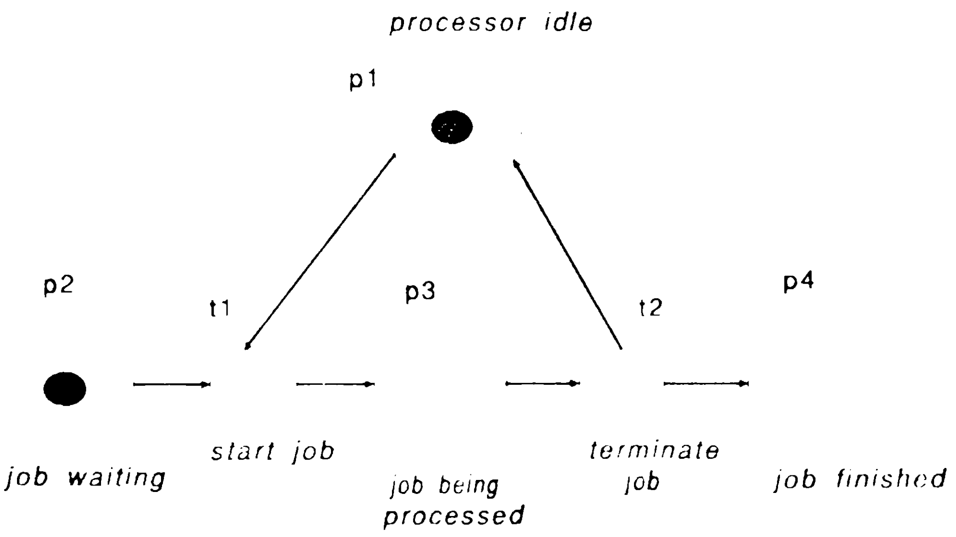

FIGURE 4. The net of Figure 3 with a job waiting to be processed.

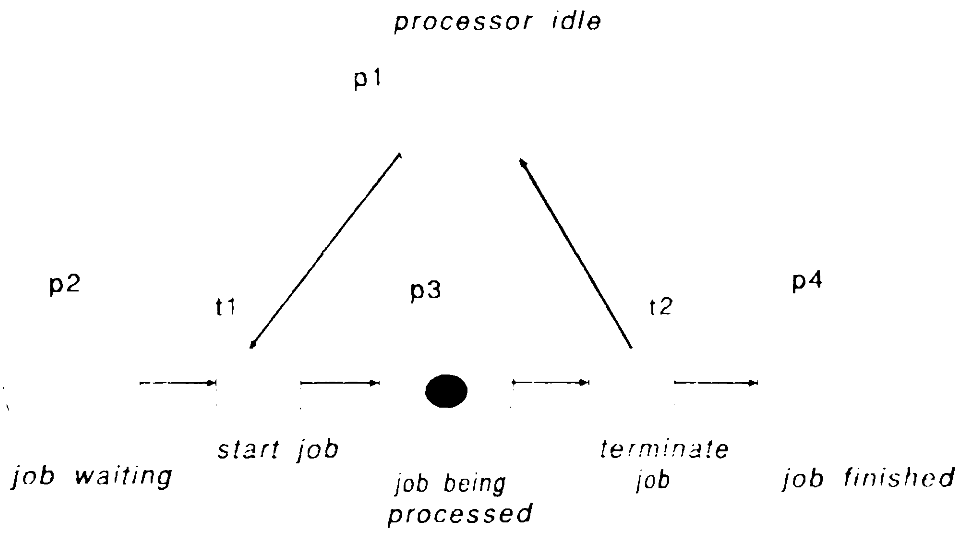

FIGURE 5. The job-processor net after firing T1 showing that a job is in progress.

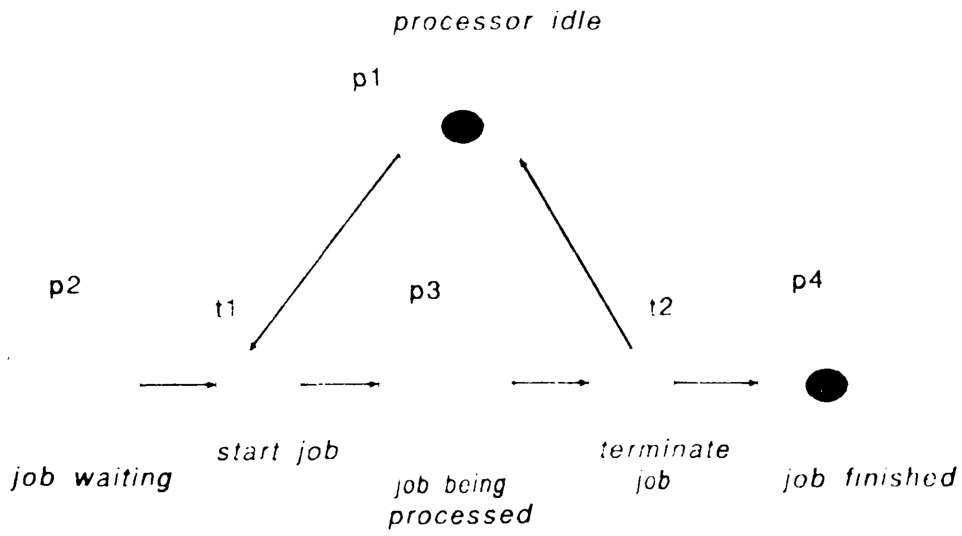

FIGURE 6. The job processor net after firing T2 and representing completion of the job.

Time in P/T nets

Transitions in a P/T net may fire whenever they become enabled; the firing of a transition is defined to be instantaneous. This means that in practice the concept of time has no meaning within the context of the standard P/T net. (Although the *sequence* of events and their synchronisation can be determined, the actual time taken by a process cannot be measured.) This makes it impossible to use P/T nets to evaluate system performance, where the measurement of process times is of the utmost importance. In view of this, and other limitations, it is necessary to make various extensions to the standard P/T net in order that realistic performance evaluation is made possible.

The addition of time to the P/T net model

The expansion of the Petri net model to include an element of time was first proposed by Ramchandani[4] and expanded by Sifakis[5] and Zuberek[6]. Both these authors developed techniques for the analysis of timed Petri nets in order to gain a realistic measure of the performance of modelled systems. Sifakis proposed a model whereby time was associated with the places of the net, while in Zuberek's model the firing of a transition was assigned a finite time. Most work on timed nets to date has concentrated on the timed-transition model (Razouk & Phelps[7], Holliday & Vernon[8].)

Stochastic Petri Nets

Stochastic Petri nets are Place/Transition nets with some additional properties. Time is added to the net, usually being associated with the transitions, but instead of being fixed, the delay associated with a place or the firing time of a transition may be a random variable. Similarly where a conflict situation exists between several transitions the output can be decided on a probabilistic basis. This type of net is discussed in Pagnoni[9] and Massan & Chiola[10]. Stochastic methods, although used primarily for analysis, are of importance in the simulation environment, notably for the modelling of queuing systems and noisy communications channels.

Chapter 2. The Development of the Simulation Method

Using Petri nets as a simulation tool

In 1981 Torn[11] described a simulation tool, based on a timed Petri net model, called the Simulation Graph. This was later expanded into the Simulation Net[12]. This is very similar to the net model presented in this thesis and, in fact, the method of reporting results and gathering statistics throughout the simulation is based on Torn's system. The Simulation Net is a timed-transition model with a variety of extensions, which Torn has applied to some simple simulation problems. The Simulation Net has no capability for modelling random or probabilistic events.

A similar type of system was developed by Alanche, et al[13] to simulate a flexible manufacturing system. The structure of this simulator is similar to the present one in that it consists of a model of the net acted upon by a token player and a statistics-gathering agent. However in this case the basic net model has been "enhanced" by the addition of extra decision-making functions to the transitions of the net, thus adding an extra level of program to the model. Nelson et al[14] similarly complicate the net model to produce a system for net execution which maps a Petri Net onto a programming language which can be compiled and run. The method appears suited more to analysis and proof-of-correctness type problems than generalised simulation work and the application of the system to real-world problems was not discussed.

A higher-level model, the Function Net was designed by Schiffner and Godberson.[15] Once again the simplicity of the net model has been sacrificed for the sake of simulation power, but many of the extensions have parallels in the present simulator, notably methods of moving variable numbers of tokens from place to place during transition firing. Time is not modelled in the Function Net system.

Bauman and Turano[16] discuss a mapping of Petri nets on to a production language model. The technique is applied successfully to the simulation of a simple communications protocol. The method addresses several of the problems of using Petri nets as a simulation tool, such as the resolution of conflict sets, but the net model used (basically a simple P/T net) is of rather limited power.

The design of a net class for the simulator

The aim in designing the present simulator was to choose a net model which preserved the essential simplicity of the P/T net while adding the necessary features which would give the method the necessary simulation power. A Petri net is a graphical structure - this is true in both the graph-theoretical sense and in the sense that a graph is a *visual* device. The program, i.e. the operational structure of the system being modelled, should be discernible from a visual inspection of the associated Petri net. (It is of course realised that for large and very complex nets, this is not a trivial exercise!)

As a starting point, it was decided to take the simple P/T net model as described in Chapter 1 and add an element of time to the model thus enabling the performance evaluation of systems. This timed-net model was then extended in several other ways to fulfil additional simulation requirements.

Adding time to the net model

It was necessary, in designing a simulation scheme using timed nets, to decide whether to use a timed-place or a timed-transition model. In fact, it has been stated by Sifakis that the two models are equivalent - this is certainly true for the purpose of analysis, but there is a considerable difference between the two methods at the conceptual level and at the practical level of implementation as a computer program. It was eventually decided to implement a timed-place model and the reasons for this decision are given in the following paragraphs. It is assumed in the following discussion that time (in the context of the execution of a timed Petri net) is a discrete quantity and that the token game is played in fixed arbitrary time intervals.

In the timed-transition model, the firing of a transition is said to take a certain time. Thus time elapses when the system makes a transition from one state to another. In the timed-place model, each place in the net has an associated delay. When a token enters the place it is said to be *unavailable*. This means that it may not take any part in the firing of transitions. After the delay associated with the place has elapsed, the token becomes *available* and may participate in transition firings as it would normally.

Attaching time to a place or a transition represents an extension of the state space of the system. Figure 7. shows a place with a delay value of 4 (the time scale is arbitrary.) Figure 8. shows how the action of the place can be modelled in an informal fashion by a system of queues, each of which is a simple Petri net system, with the number of queues equal to the capacity of the place. As a token enters the place it is assigned an empty queue and starts its progress at the first place in the queue. At each time interval the token moves up the queue until it reaches the end, whereupon it becomes an enabling token for any transition having this place as an input. A similar description may be given of a timed transition - when the transition fires, tokens which have been removed from the input places enter a queue just as in the timed-place example. When they emerge from the queue they are assigned to the output places of the transition. It should be emphasised that this does not constitute a formal description of timed nets nor is it claimed that the net can usefully be decomposed in this fashion; the illustration merely serves to show that the operation of timed places or transitions is deterministic given that time proceeds in discrete intervals.

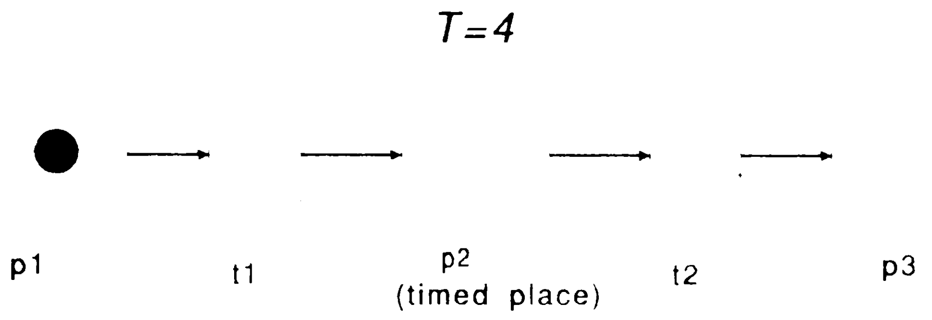

FIGURE 7. A timed place with delay 4.

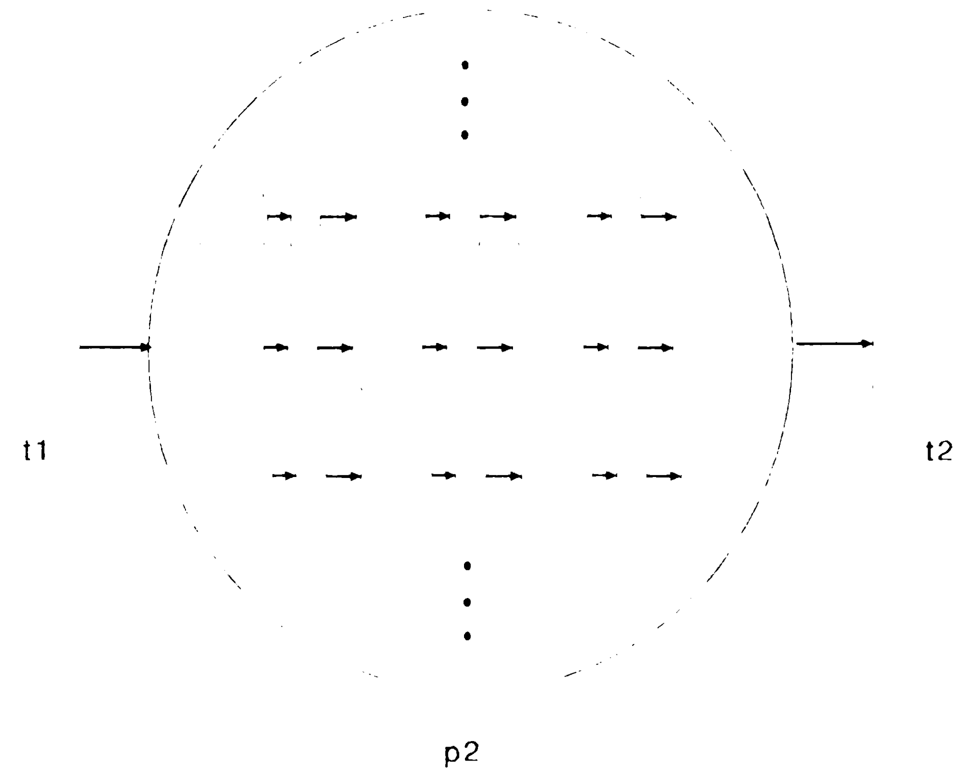

FIGURE 8. Place P2 of Figure 7 modelled as a system of queues.

It is intuitive that the state space of the process represented by a Petri net is modelled entirely by the set of reachable markings of the net and that the state of the system at any given time is modelled entirely by the disposition of tokens in the places of the net. That transitions fire instantaneously serves to highlight this quality. It is also intuitive that changes of state occur instantaneously. In view of this it seems desirable to associate time with the places of the net.

If time is assigned to the transitions of the net (and if one preserves the intuitive notions of states and transitions outlined above) then an anomalous situation arises whereby the changes of state take time while the states of the system occupy no time at all. This clearly raises problems since during the firing of a timed transition the state of the system is undefined unless one assumes that transitions have internal states which themselves represent states of the system as a whole thus effectively reversing the roles of the places and transitions. In discussing a class of timed-transition nets, Zuberek[17] writes "In M-timed nets the tokens are removed from corresponding places at the beginning of transition firings and remain 'in' transitions for the whole period of firing. The 'state' describes the distribution of tokens in the places as well as transitions..." This serves to confuse the natural demarcation of states and transitions within the net model.

Razouk & Phelps state that it is a moot point whether tokens within the input places of a transition remain active within the place during the firing of a timed transition and are thus liable to be removed by the firing of another transition, or whether they disappear into the transition for the duration of the firing interval. This possible discrepancy does not exist within the timed-place model.

Finally, Petri[18] concludes that time should be primarily associated with the state-elements of the net, i.e. with the places.

There is, inevitably, a price to be paid for choosing timed places over timed transitions, mainly in the area of implementation of the simulation method as a computer program. It is much easier to program a timed-transition simulator. Also, a useful feature such as the ability to cut short a delay (as described in Chin & Willsky[19] and also used in the Ethernet simulation of Chapter 6) is more elegantly modelled under the timed-transition model. It was, however, considered important to adhere as closely as possible to a simple net model with places and transitions performing their intuitive functions so the timed-place model was implemented.

The resulting net model is known as a **Timed Place/Transition Net (TPTN)**.

Timed place/transition nets

Time is associated with the places of the net in the following manner:

Execution of the net occurs in fixed *instants* of time. Each place is assigned a *delay* of an integral number of instants (by means of which execution times may be associated with places.) Each token in a place may be in one of two states: *available* or *unavailable*; only available tokens may participate in the firing of transitions. When a token is deposited in a place it is initially set to the unavailable state. After a number of instants (determined by the delay of the place and during which the token may not participate in any transition firings), the token becomes available. At each instant, any transition which is enabled (according to the usual firing rules for P/T nets, but using only the available tokens) may fire. The appropriate number of available tokens is removed from the input places and an appropriate number of unavailable tokens is deposited in the output places. A typical firing sequence is shown in Figure 9.

Other extensions to the basic net model

In addition to the inclusion of time within the net model, several other extensions were necessary to increase the simulation power of the nets. Peterson[8] and others have discussed the conflict between decision power and modelling power in Petri nets and related models. A finite-state system, for example, has a very high decision power in that a complete analysis of the system is possible. A finite-state machine, however, is severely limited in the types of system which can be modelled without inducing a state explosion. At the other extreme lies the Turing machine with its ability to model any system, but for which the power of decideabiity is lost. The standard P/T net model lies somewhere between these two extremes. However in the present context where the aim was simulation rather than analysis it was felt that extensions to the basic net model which increased the simulation power of the net would be advantageous. The extensions made were as follows:

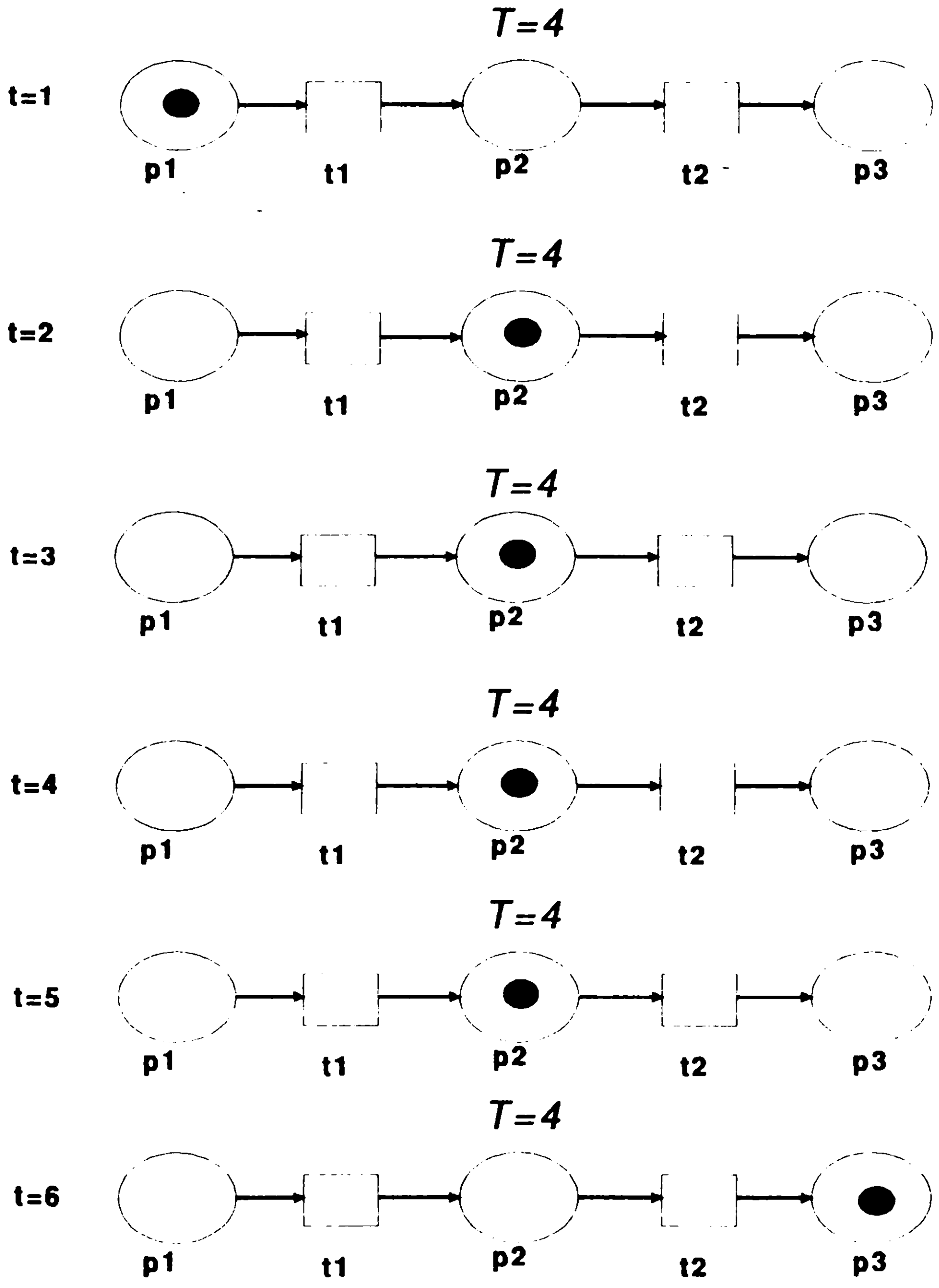

FIGURE 9. The firing sequence for the net of Figure 8.

Inhibitor Arcs

This type of arc has been in use almost since the inception of the Petri Net. An inhibitor arc is an arc from an input place to a transition having the property that the transition may not fire unless the input place is empty. This effectively supplies a means for zero-testing, which greatly increases the modelling power of the net. It has been shown (Peterson[8] etc.) that a P/T net extended in this way is equivalent to a Turing machine in modelling power (but with a corresponding loss of decision power). An inhibitor arc is distinguished graphically by having a small circle drawn at the transition end of the arc.

Multi-weighted arcs

This extension, developed as part of the present simulation model, allows an arc to have a weight which is defined to lie between upper and lower limits w_u and w_l. The firing rules are modified so that a transition having this type of arc as an input may fire if the number of tokens μ_p in the corresponding input place lies between w_u and w_l.

When the transition fires, the number of tokens removed from the input place is

$$w_u \text{ if } \mu_p \geq w_u$$

$$\mu_p \text{ if } w_l \leq \mu_p < w_u$$

This type of arc is useful in modelling queues and buffers. An example of the operation of such an arc is shown in Figure 10. The multiple-weighted arc connects place p2 and transition t2. In net a.) the token count in p2 is below the minimum weight of the arc, so transition t2 cannot fire. In b.) the firing of t1 causes the token count in p2 to equal the lower limit so that t2 can fire.

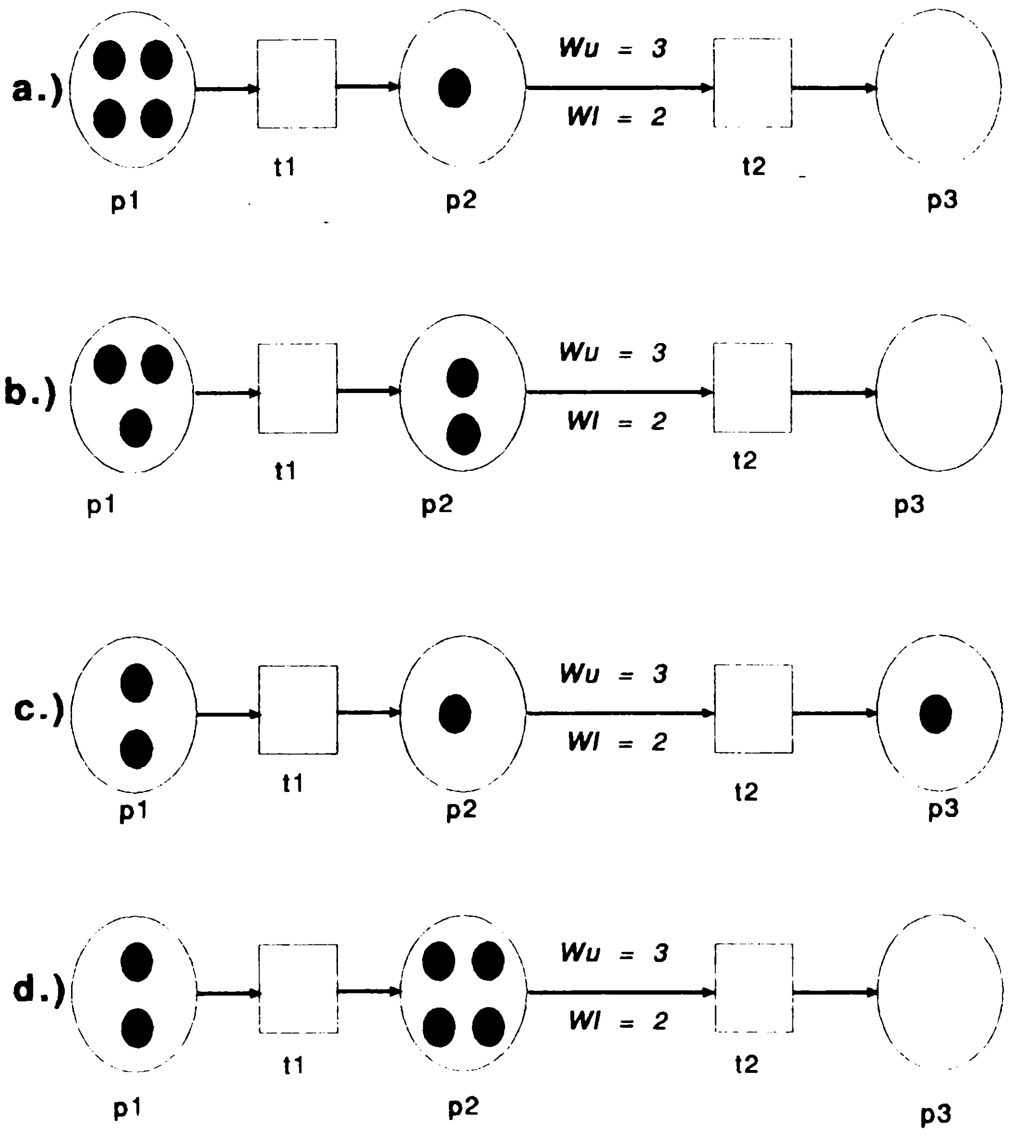

FIGURE 10. A multi-weighted arc.

This is shown in c.) In net d.) the token count in p2 is above the upper limit of the arc so t2 cannot fire.

Omega-weighted arcs

Arcs with weight omega are restricted to the configuration shown in Figure 11. The top row of places and transitions represents the queue, the tokens of which are moved along under the control of the places and transitions in the second row. Parts a, b and c show the execution of this net. An Omega-weighted arc is an enabling arc if the contents of the input place are greater than zero. When the transition fires, ALL of the tokens in the input places are removed and deposited in the output place. Omega-weighted arcs are especially useful in modelling queues where the entire contents of each place need to be moved along, as is shown in the Figure. This element was also developed specially for the present system.

Random-Input places

In order to provide for the occurrence of random events, the Random-input place was developed. This is a place where, at each instant of execution of the net, a random number of tokens may appear. This type of place has many uses in simulation, including the random arrival of customers at a queue.

Random-delay places

This type of place was developed in order to aid the simulation of stochastic nets. The place has the characteristic that a token entering the place is given a random delay.

Probabilistic places

A token entering a probabilistic place is initially set to the unavailable state. At each subsequent instant it may or may not become available according to the probability associated with the place.

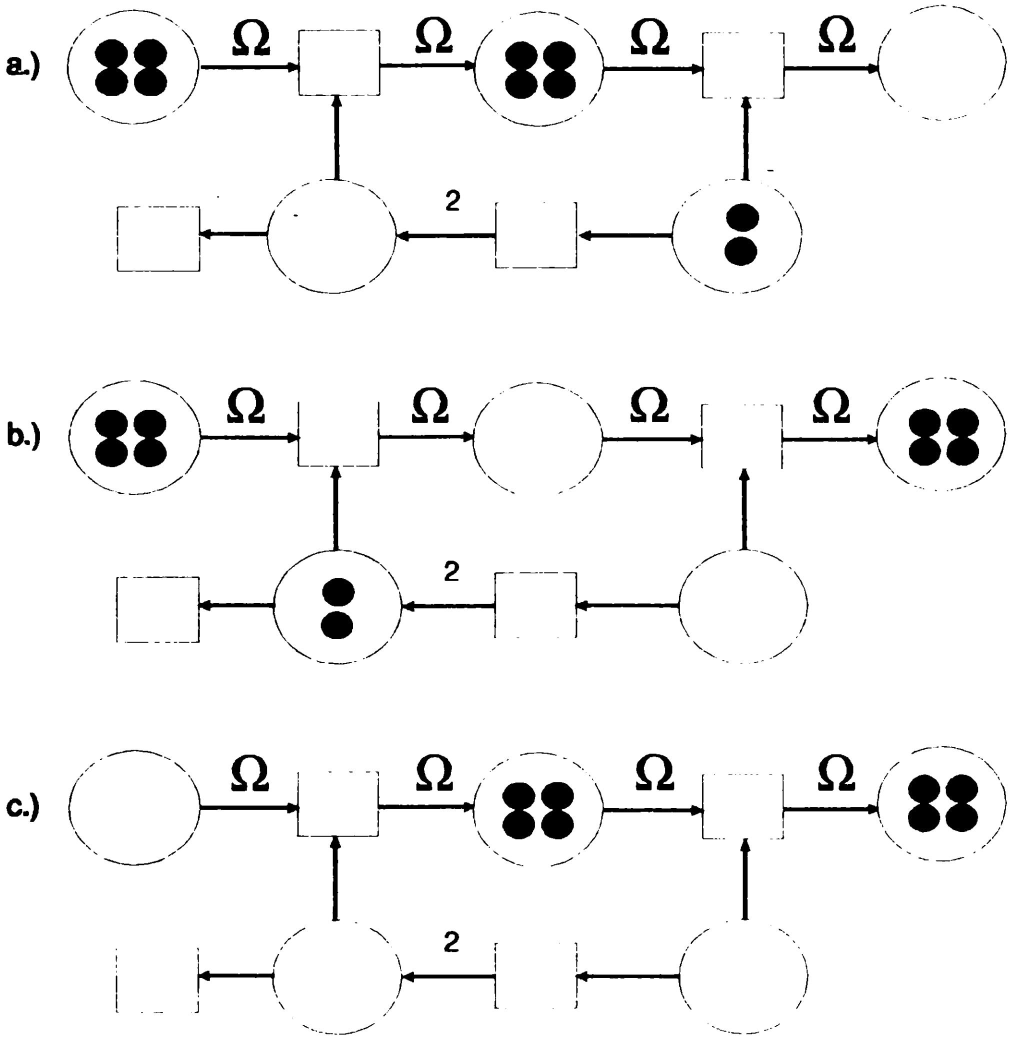

FIGURE 11. The operation of an Omega-weighted arc.

Apart from the fact that they were felt to be rather undesirable in themselves, the other extensions to P/T nets which have been proposed by various researchers, such as attaching conditions to transitions and arithmetic/logical operations to places were not necessary to model the systems under consideration here.

Chapter 3. The Design of the Simulator.

The simulation method described in the previous chapter was developed into a program which could simulate various types of timed Petri net. The aim was to produce a piece of software which would take as its input a description of a net, execute the net and produce as its output a set of statistics gathered during the course of the execution.

The first version of the simulator was implemented as three separate programs written in C. The first module, a net editor, allowed the parameters of the net model to be entered and edited in a rather primitive manner. The editor produced an intermediate net file which could be read by the simulator proper. The second module, the simulator, converted the net specified in this file into a series of static data structures which were then manipulated by a token player in order to execute the net. During this execution, various statistics relating to the places and transitions of the net were accumulated. When the simulation had run its course a table containing a summary of the various statistics was printed out. A third module comprising a net compiler was then added and this allowed the net to be specified as a text file containing a clause for each place and transition in the net, this specification then being converted into the intermediate net file.

It was felt, after some experience with the original simulator, that it would be advantageous to write the simulator in an object-oriented programming language because it was clear that Petri Nets fitted the object-oriented paradigm rather well. To this end the entire system was eventually re-written as a single program using the C++ programming language and it is this final version which will be discussed in this chapter. For details of the C++ language, the reader is referred to the reference text by Stroustrup[20].

The object-oriented method allows various data structures to be encapsulated as *objects.* An object contains various items of data along with a series of *methods* which specify the various operations which may be carried out on these items. By means of *data-hiding* the internal structure of an object may be made invisible to the program in which it is used, the external appearance of the object being defined by its methods. In the C++ language the methods are implemented as a series of functions associated with each object. The program may only access the object via these functions which in turn manipulate the data items belonging to the object. The mechanism of *inheritance* allows a new object to be created from an existing similar object. The new object *inherits* both the data types and the methods of the ancestor object and may then add new data items and methods of its own.

Each of the various components of a Petri net may be treated as an object. For example, in the present timed Petri net model a token is characterised by its availability, i.e. the token may be either available or unavailable. The state of the token is decided by various flags and counters, but these are internal to the token itself, only its availability is of importance to the other parts of the net and consequently this is the only property of the token which is visible to the rest of the program. Similarly a place has only one external characteristic, namely its marking, i.e. the number of available tokens within the place. The mechanism by which this number is decided is hidden within the place object. Lastly, a transition has one external property, namely whether it is enabled or not. It also has the ability to fire. Both the calculation of its state and its behaviour in firing are decided internally by the transition object.

Obviously, the model is not watertight, for example in order for the transition to fire it must have access to the tokens within each of its associated places, but such less-than-ideal features are handled quite readily by the C++ language.

The various features of the simulator, including some of the programming and implementation details, are discussed in the following short sections.

Input of net specifications

The net specification is entered as a text file which is then compiled by the simulator into a set of internal objects. The specification file takes the form of a series of clauses, one for each place or transition in the net, along with a single clause which specifies some general parameters of the net as a whole. The format of each of these clauses is given below in Figure 12. The keywords for each clause are shown in bold face and the parameters are shown in angled-brackets. The parameters in square brackets are optional.

```
 net <name>
      {
      run_time <no of instants>
      }

place <name>

     {
     type <type>
     capacity <capacity>
     marking  <unavailable>
     delay    <delay>
     }

      .
      .

random_place <name>
     {
     type <type>
     capacity <capacity>
     marking  <unavailable>
     delay    <delay>
     distribution <mean> <tolerance>
     }

      .
      .

probabilistic_place <name>
     {
     type <type>
     capacity <capacity>
     marking <unavailable>
     probability <probability>
     }

      .
      .

  transition <name>
      {
      input <place_name> <type> <lower weight> [<upper weight>]
      input <place_name> <type> <lower weight> [<upper weight>]
      .
      .

      output <place_name> <type> <weight>
      output <place_name> <type> <weight>
      .
      .

      }
      .
      .
```

FIGURE 12. Format of the net specification file.

The **net** clause specifies the name of the net, i.e. the title of the simulation and the number of instants for which the simulation is to be run.

The **place** clause specifies the various parameters associated with the place - one clause is included for each place in the net. The *type* field specifies whether the place is a normal place or not. The *capacity* field specifies the maximum number of tokens which may be present in the place. The *marking* field is used to specify the initial marking of each place so that the execution of the net can begin from any given state. The delay field specifies the time delay, In instants, associated with the place. A value of zero in this field means that tokens entering the place will become available immediately.

The **random_place** and **probabilistic_place** clauses specify the extra values associated with these place types. In the case of the random place, the type value determines whether the place is a random-place in which a random number of tokens may appear each instant, or a random-delay place in which each token is given a random delay value on entry. The mean and tolerance fields specify the mean value of the distribution used to calculate these random values and the allowed limit of accuracy of the mean value. The *probability* field of the probabilistic place specifies the probability with which each token within the place may become active at each instant.

The **transition** clause specifies the input and output places for each transition so that the appropriate arcs may be constructed between them. Places are referenced by name so that transitions and places may be entered in any order in the specification file. However the order of entry of the transitions may have an effect of firing priority; this will be discussed later. The *type* parameter may be *normal* or *omega* depending on the type of the arc. If the second weight parameter is present then the arc is considered to be multi-weighted, while if it is absent then the arc is a normal single-weighted one.

Place and **transition** clauses may be mixed freely so that the places and transitions belonging to various sub-sections of the overall net structure may be grouped together for clarity.

An example of a complete net specification file, namely that for the terminal-computer system described in Chapter 5, is given in appendix 1.

Compilation and internal representation of the net

The net specification file is compiled by the simulator into a series of objects. Objects are known as classes in C++ and a separate class has been used for each of the principal components of the net. There is a place class, a transition class, a token class, etc. As an example, the **place** class and its derived classes **random_place** and **probabilistic_place** are shown in Figure 13.

```
class place
    {
protected:
    static int time;                  // global time for all places
    place_type type;                  // type of place
    int  delay;
    int  capacity;
    int  unavailable_tokens;
    token *tokens;                    // pointer to first token in list
    stat_counter *time_spent;         // statistical variables
    stat_counter *zero;
    stat_counter *marking;
    stat_counter *change_of_marking;

public:
    char name[NAME_LENGTH];
    int  available_tokens;            // number of tokens available
    place *next;                      // pointer to next place in list
    place(char *n, place_type tp, int c, int d, int t);
    void add_token();                 // add a new token
    boolean remove_token();           // get rid of a token
    boolean isactive();               // any active delays?
    int update();                     // update markings
    void print_stats();               // write out statistics
    friend enabled_type transition::isenabled();
    friend void petri_net::output_special_data();
    friend void petri_net::update_time(boolean flag);
    friend ostream& operator<<(ostream&, place&);
    };

class random_place : public place
    {
private:
    float mean;                       // mean number of tokens in place
    float limit;                      // allowed error limit for mean
    random_vector *random_list;       // file of random numbers
    random_vector *setup_random_list(int count);
    stat_counter  *random_data;       // Stats on random tokens

public:
    random_place *next;               // pointer to next random place
    random_place(char *n, place_type tp ,int c, int d, int t, float mu,
float l, int count);
    void update(boolean flag);
    };

class probabilistic_place : public place
    {
private:
    float probability;
    stat_counter *random_data;
public:
    probabilistic_place *next;
    probabilistic_place(char *n, place_type tp ,int c, int t, float
probability);
    void update(boolean flag);
    void add_token();                              // need special routines
    void print_stats();                            // to handle the delays
    };
```

FIGURE 13. Class definitions for the place and associated classes.

The classes **random_place** and **probabilistic_place** are derived from the class **place** using the C++ inheritance mechanism. This is appropriate since these places are just normal places with some extra properties. The elements of each class specification prefaced by the *private:* keyword are internal to the place class; they cannot be accessed by any part of the program other than the member functions of the place class. The elements in the *public:* section may be accessed by any part of the program. Note that member functions are also declared as part of the class - these are the functions by which the elements of the class are manipulated. These functions are accessible to the entire program and are the only way in which the class or object may be accessed or manipulated.

Functions belonging to other classes may access the place class if they are declared here using the *friend* keyword. Thus, for instance, the function **isenabled()**, which is a member of the **transition** class, may access the tokens of a place. This is necessary so that a transition may decide whether it is enabled or not.

Similar classes exist for transitions, tokens, input and output arcs, statistical counters, etc.

Internal storage

Memory is allocated to the various objects at compile-time from the heap. This means that small nets may be simulated on a PC with limited memory resources while larger nets may be run on a larger system with more memory and processing power.

Each net contains a single instance of the class petri_net. The places and transitions of the net are stored as linked lists of instances of the place and transition classes, with a pointer to each list being stored in the petri_net structure.

Places with random elements

Random input places, as mentioned earlier, are places into which, at each instant of simulation, a random number of tokens may appear. In order that the mean of the distribution controlling the operation of the place may be tightly specified, the set of random numbers for each random place is calculated and stored before simulation is commenced. Because the number of tokens which may appear in a place in a given instant is an integer there is considerable variation in the mean between sets of random numbers, due mainly to the nature of the modular random number generators involved. Pre-calculating the random numbers ensures that the means are always accurate, although with some cost in storage space. The basic random number generator used was a modified version of the congruential pseudo-random type. This generator was modified to produce random numbers having a Poisson distribution rather than the uniform distribution of the original generator.

Internally, the class **random_place** contains an instance of another class, the **random_vector**. This is a structure used to hold the list of random numbers. The constructor for the **random_vector** class has been heavily modified so that each time a new **random_vector** is created, a list of random numbers with the correct mean is generated and stored in a file. The file is left open for reading and the file descriptor is stored in the **random_vector** structure. A public function exists within the **random_vector** class to return the next random number from the list.

The same method is used for random-delay places. A random-delay place is an instance of the **random_place** class with the *type* field set to indicate a random_delay type. This type of place is treated as a normal place until such time as a new token is added. At this point, the next random number is read from the associated **random_vector** and the token is assigned this value as its delay.

The probabilistic place operates slightly differently in that at each instant the availability of each token is decided by a "throw of the dice," the token having a

1-in-*n* chance of becoming available where *n* is the reciprocal of the probability associated with the place.

Transition-firing priorities

The transition firing order is only of interest in a conflict situation. This was defined in Chapter 1 and occurs when two transitions share an input place and are simultaneously enabled by a single token in that place. The regular action is that the enabled transitions are fired in the order in which they occur in the list of transition which is stored in the petri_net structure. First, a pass is made through this list and another list is created of those transitions which are enabled. A pass is then made through this second list, firing each transition in turn.

There is an option to scramble the firing list before firing the transitions and this is useful in situations where conflicts are allowed to occur but need not be resolved in any predetermined order. In any case, a check is made for conflicts and if any is found to occur, a diagnostic message is printed out by the program.

Time handling

Two methods of advancing the time-frame are possible in discrete event simulation systems: next-event time advance and fixed-increment time advance, the former method being the most commonly used. In the next-event methodology, time is advanced by one clock increment each time an event occurs. Because each state-change of the system takes place upon the occurrence of an event, periods of inactivity are skipped over.

In the fixed increment method, time is advanced in fixed intervals by means of an external clock which corresponds to "real-time." At the end of each interval the system is examined to determine whether any events have occurred.

The present model allows both types of time to be used. A global timer advances with each "instant" of simulation of the net while, in addition, a counter representing the "simulation time" is advanced whenever a delay is active in any place of the net. Any delay associated with a place becomes active immediately after the firing of the appropriate transition and continues until the delay has elapsed. During this period the net may be said to be "active." Note that this is not true "next-event" operation because the state of the net is still evaluated for every tick of the global timer.

It would be possible to improve the performance of the simulator by detecting those periods when no transitions are enabled, finding the shortest active delay and advancing the timers by this amount. This would entail some modification of the statistical modules as the statistical counters are presently updated with each tick of the global timer.

In models where every place in the net can be considered "active" i.e. each place is associated with some time-consuming process, the global time can be used effectively. This is also the case with untimed nets, as in the case of the first single-terminal model discussed below. Where only a subset of the net places are timed, however, the simulation time gives a measure of the total activity time of the model. This allows the effect of places which are not part of timed processes to be eliminated from the final timings.

Termination of simulations

A simulation run may be terminated in one of three ways:

- **Time limit exceeded**

When the simulation is started, a maximum allowed value of the global time is specified. If this is exceeded then the run will terminate and a report will be printed.

- **Terminal marking reached**

A terminating marking can be specified at the start of each run. If this marking is encountered during the course of the simulation then the run will terminate. This allows the simulation to be stopped when the system is in a given state so that the statistics may be examined at that point.

- **Deadlock reached**

Deadlock is defined to occur when no transitions are enabled and no place delays are active - in this case no further execution of the net is possible and the run is terminated. This feature may be turned on or off depending on the type of system under examination. For example if it desired to examine the "steady-state" behaviour of a system then deadlock checking may be turned off and the system simulated for a given number of instants to allow the system to stabilise. In other systems, such as communications protocols, it is useful to see if and when deadlock occurs and in this case deadlock checking can be activated.

Gathering of statistical information

A class named "stat_counter" is defined which contains various counters and functions for calculating statistical quantities. Several instances of this class occur in the transition and place classes to allow measurement of their behaviour with respect to several parameters. The parameters which are measured for places are:

- **Time spent**

The time spent parameter is a measure of how long a given place is occupied by tokens. Whenever a token enters a place, an *episode* is said to begin and a counter is started. This runs until the place becomes empty once more. The maximum, minimum and mean times for these episodes are calculated for each place. This is useful in determining the length of time the system (or part of the system) remains in a given state, e.g. measuring the idle time of a processor.

- **Zero**

This parameter is a measure of the amount of time a place remains empty, i.e. contains no tokens. The first and last instants at which the place is empty are logged, along with the total empty time expressed as a percentage of the total simulation time.

- **Marking**

The maximum, minimum and mean marking for each place is calculated. This calculation is made on the basis of *available* tokens.

- **Change of marking**

This parameter provides an indication of the activity of each place and is useful for determining the level of activity of various parts of the system under simulation. The first and last instants at which the marking changed are logged along with the number of changes. This parameter is calculated on the basis of the *total* marking, i.e. the number of unavailable tokens plus the number of available tokens. In this way a change of marking occurs only when tokens enter or leave the place and not when an unavailable token becomes available.

The parameters for transitions are:

- **Firing times**

The first and last instants at which each transition fires is logged along with the total number of firings. The maximum, minimum and mean *time between firings* is also calculated. These parameters yield useful information about the occurrence of *events* within the system.

Output of results

When each simulation has run its course, a table of results is written out to a file. This table contains a listing of the various statistical parameters described above for each place and transition in the net, along with various parameters relevant to the simulation run as a whole, such as the marking of the net at the start and end of the run, the computer runtime for the run, etc. An example of such a table (in this case for the communications system described in the next chapter) is given in appendix 2.

A function also exists as part of the class petri_net which will write out any special output data required for a particular simulation. This function may be activated by means of a command-line switch and is useful for writing out data in a form which is readable by a spreadsheet or graphics program. The function has access to the statistical counters for all of the places and transitions in the net by means of the friend mechanism of C++. Although the idea of having to write a piece of C++ code for each simulation in order to obtain the output data in this way, it was felt that this was preferable to writing an all-purpose output routine since the type of output required varies so widely between simulations.

Chapter 4. Simulation of the Glasgow Royal Infirmary Cart-computer communications system

Electrocardiography

Since its inception at the start of the century, clinical electrocardiography has progressed to the point where it is the primary non-invasive cardiological investigative technique in use today. Electrocardiography is the measurement of the electrical activity of the heart as it is manifested at the body surface. This electrical activity is measured by placing electrodes at various points on the body surface and recording the potential difference between various combinations of the electrodes, with each particular combination being known as a *lead*. The equipment used to record the signal is known as an *electrocardiograph*; the output from each lead is usually written out on a strip of paper in real time by a chart recorder and is known as an *electrocardiogram* or an *ECG*. An electrocardiogram thus takes the form of a plot of voltage against time. In a normal clinical situation, a *12-lead electrocardiogram* is recorded from the patient under investigation and Figure 14 shows the positions of the various electrodes for this type of electrocardiogram.

The 12 leads in question fall into three groups:

The Bipolar Limb Leads

Lead I - records the voltage between the left arm and the right arm.

Lead II - records the voltage between the left leg and the right arm.

Lead III - records the voltage between the left leg and the left arm.

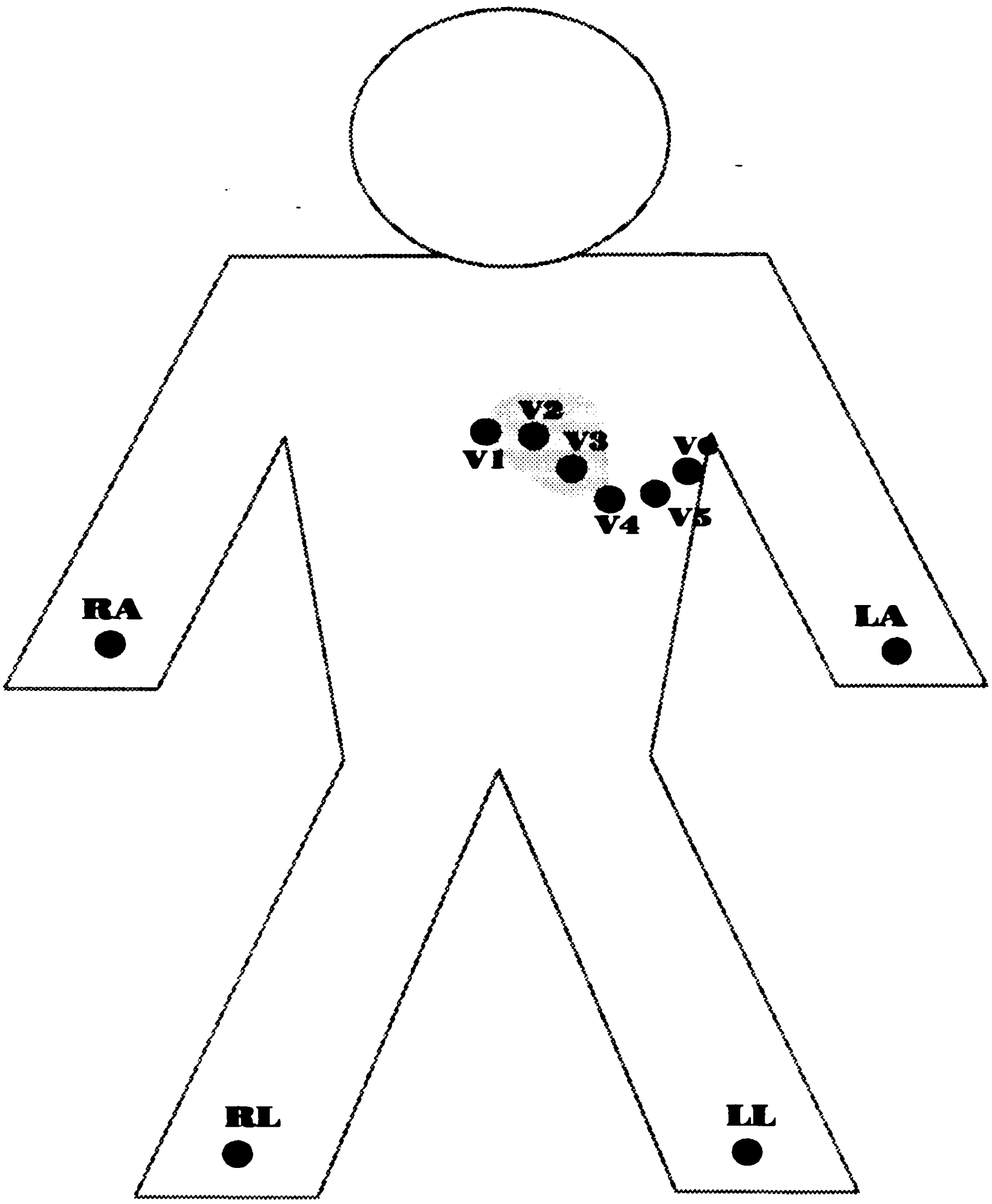

FIGURE 14. Electrode positions for the 12-lead Electrocardiogram

The Augmented Limb Leads

Each of these leads records the voltage between one of the limb electrodes LA, RA or LL, and a reference potential formed by averaging the potentials on the other two. Note that the right leg electrode is not actively part of the lead system. The leads are:

aVR - the voltage between the right arm and the average of LA and LL.

aVL - the voltage between the left arm and the average of RA and LL.

aVF - the voltage between the left leg and the average of RA and LA.

The Unipolar Precordial leads

Each of these leads is formed by recording the voltage between one of the chest electrodes (V1 to V6 as shown in Figure 14) and a relatively constant reference potential formed by averaging RA, LA and LL. This reference is known as the Wilson Central Terminal. The precordial leads are named after the chest electrode positions, **V1** through to **V6**.

The reason for recording so many different leads is that each one presents, in effect, a separate *view* of the electrical activity of the heart. The left precordial leads **V3** through **V6** predominantly represent the activity of the left ventricle while the right precordial leads **V1** and **V2** "look" at the right ventricle. An alternative interpretation is that each lead measures the component of the total cardiac electrical activity in the direction associated with the lead. This differentiation is of great importance in the localisation of heart defects when interpreting an electrocardiogram.

Figure 15 shows an example of a 12-lead ECG recording.

The ECG Waveform

As can be seen from Figure 15, the waveform in each lead consists of a series of repeated patterns each of which in fact corresponds to a single beat of the heart (or *cardiac cycle*.) Figure 16 shows a representative waveform for a single cardiac cycle with each of the component parts labelled. These parts are as follows:

The P wave

This small wave results from the electrical activation of the atria which produces their contraction at the start of the cardiac cycle.

The QRS complex

The QRS complex corresponds to the excitation of the ventricles. The initial negative deflection is known as the **Q wave** while the first positive deflection is known as the **R wave**. The small final negative deflection of the complex is the **S wave**. Not every QRS complex has all three components while some may have four or five.

The T wave

The large wide wave following the QRS complex is the **T wave** which is caused by the repolarisation of the ventricular mass after contraction.

Other lead systems

In addition to the 12 lead ECG just described, it is possible to record an ECG with a different set of leads using alternative electrode positions. The most common alternative approach is that of *Vectorcardiography*. This technique records the electrical activity of the heart in three orthogonal planes by using a set of three orthogonal leads named X, Y and Z. The X lead measures the voltage between electrodes placed on the right and left sides of the chest, the Y lead between electrodes on the left leg and the neck or head and the Z lead between electrodes on the sternum and the back of the patient. The X, Y and Z leads can be combined in pairs to produce x,y plots or vectorcardiographic loops which often show information of clinical value not readily seen in a normal scalar ECG display such as that of Figure 16.

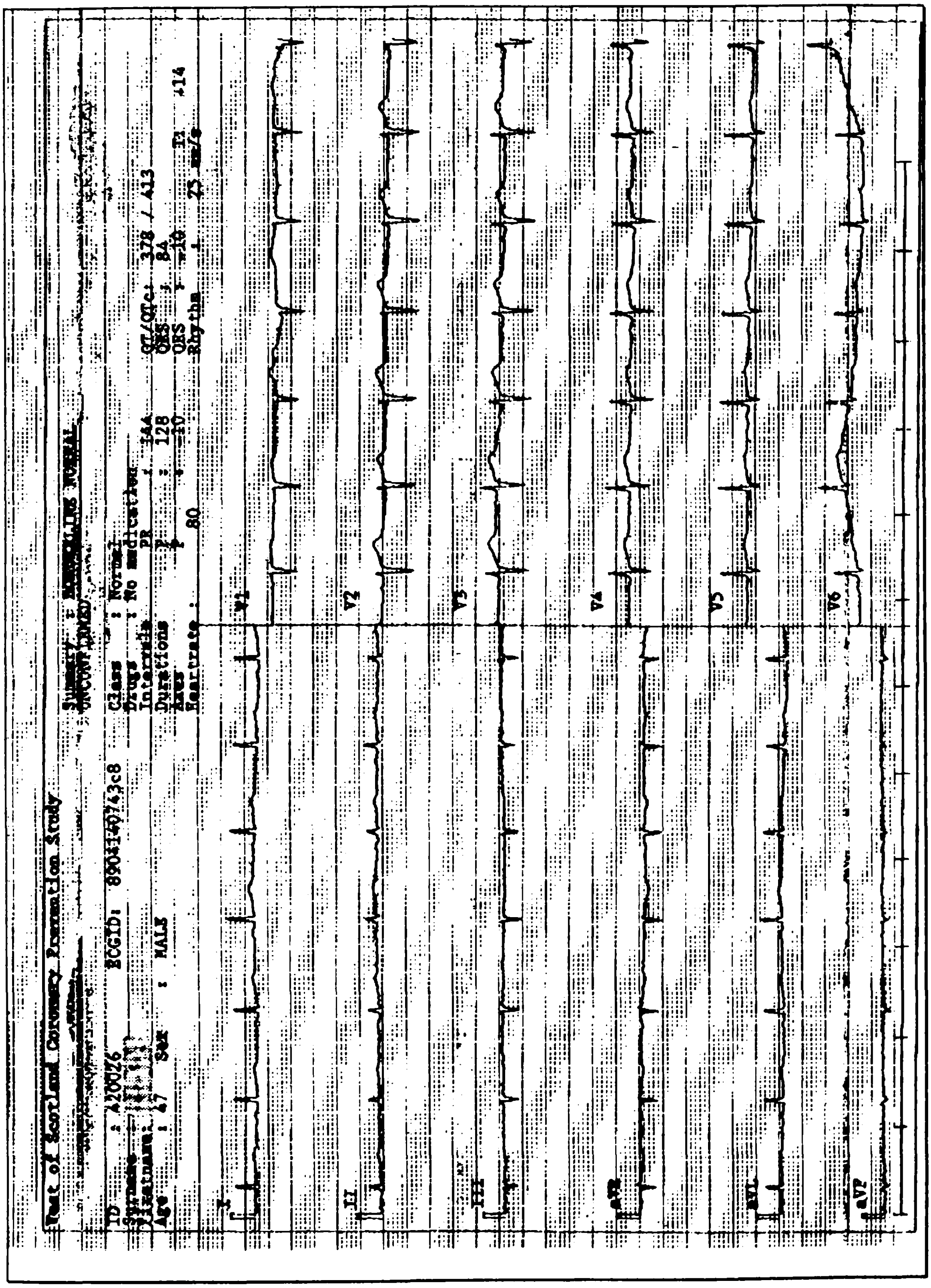

FIGURE 15. A 12-lead ECG recording.

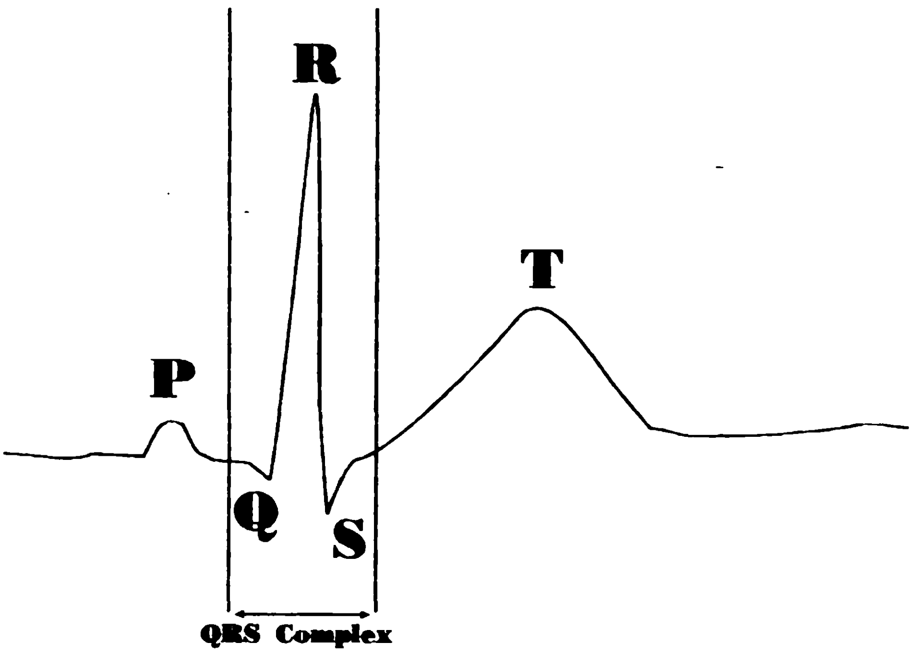

FIGURE 16 The ECG waveform for a single cardiac cycle.

Interpretation of the ECG waveform

Interpretation of an electrocardiogram is normally carried out by a physician who in large measure adopts a process of pattern recognition. Diagnostic criteria have evolved over the years which have established the normal limits for the P,QRS,T appearances and the clinical significance of deviations from these norms. Variations in the time interval between successive beats and changes in the morphology of the waveform from beat to beat are used to asses the cardiac rhythm. Once again, a body of diagnostic criteria exists relating these variations to different types of cardiac arrhythmia.

The use of computers in ECG analysis

As has been stated previously, the ECG is the primary non-invasive diagnostic technique used in modern cardiology. As a result of this, in any cardiology department in a large hospital, over 25,000 ECGs are recorded in a typical year, placing a heavy workload not only on the clinicians who interpret the ECGs but also on the clerical staff who must archive the recordings and reports. This has led to the increasing use of automated interpretation of ECG recordings by computer both as a way of reducing this workload and of allowing the use of ECG interpretation in clinical areas where no adequate cardiological expertise exists for such interpretation to be undertaken.

Digital recording of ECG waveforms

A normal electrocardiograph is a purely analogue device; the voltages from the body-surface electrodes are amplified and written on to paper using analogue electronics. In order for computer analysis of ECG waveforms to be undertaken, the signals must first be converted to digital form. This has resulted in the emergence of a new type of electrocardiograph designed specifically for this purpose. A simplified block diagram of such a device is shown in Figure 17. After amplification by normal analogue means, the ECG signals are sampled at regular intervals by a multi-channel analogue-to-digital converter. The resulting series of digital values for each lead is then stored in memory for later use. The electrocardiograph may contain hardware and software which enable it to produce an immediate interpretation from the ECG signals, or the digitised data may be compressed and stored for later transmission to a central computer for analysis.

The sampling interval used in the digitising process is normally 2 milliseconds or 4 milliseconds. Studies of the frequency spectrum of the ECG have shown that most of the components which are of diagnostic significance lie in the frequency range 0Hz - 100Hz[21] so these sampling rates are adequate according to the sampling theorem. A more comprehensive view of electrocardiograph technology is given in Watts and Shoat[22] and the particular approach used in Glasgow Royal Infirmary will be presented in a following section.

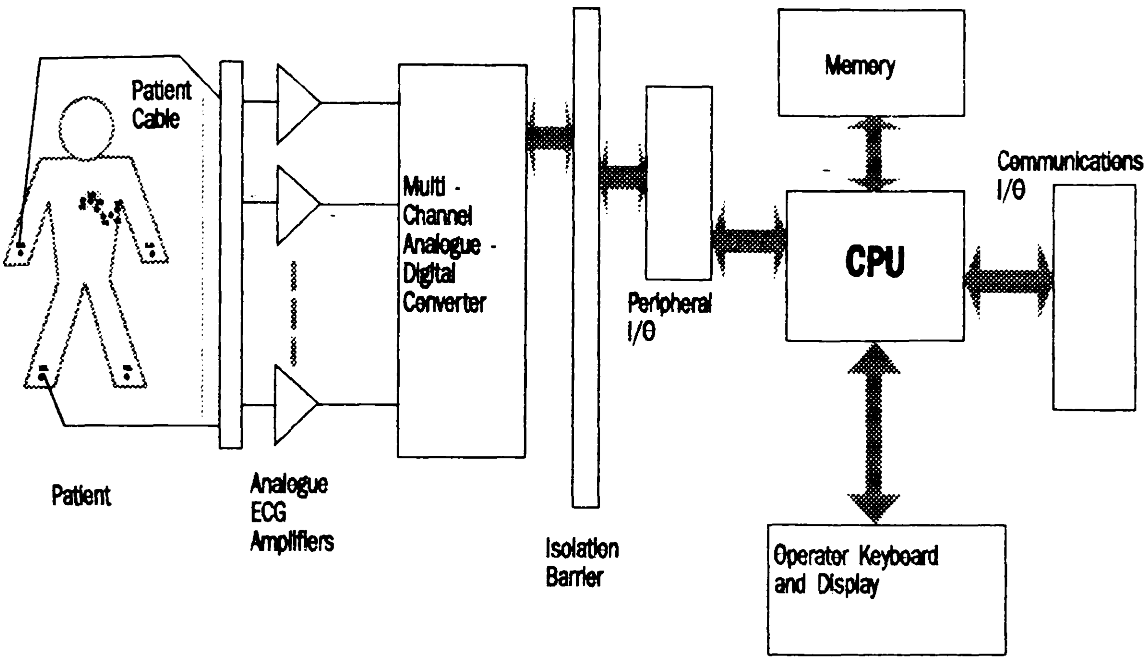

FIGURE 17. Simplified block diagram of Digitizing Electrocardiograph.

Automated ECG Analysis at the Glasgow Royal Infirmary

Research began in Glasgow Royal Infirmary into the subject of automated reporting of electrocardiograms in 1964. The original system was based on the 3-orthogonal lead ECG and was implemented on a PDP-8 minicomputer using PAL-8 as the programming language; this system is described in detail in Macfarlane et al[23] and Macfarlane et al[24]. The system was later expanded by Taylor[25] to allow analysis of cardiac arrythmias.

This early system used analogue recording methods, the ECGs being recorded on magnetic tape using a purpose-built mobile recording trolley and replayed into the PDP-8 computer through an analogue-digital converter. A mechanism was also developed to transmit ECGs from a remote hospital using analogue transmission over telephone lines[26].

In 1978 a project was begun to upgrade the system to provide improved recording and diagnostic facilities. The system was based around the standard

12-lead ECG rather than the less commonly-used 3-lead ECG of the older system. The analogue recording method was abandoned and a new digital method adopted, to which end a special-purpose digitising electrocardiograph (known as the CART and herein referred to as such) was designed. In addition, the ECG analysis program was re-written to run on a PDP-11/60 minicomputer, using Fortran instead of assembly-language and a pilot transmission system was installed which allowed ECGs to be transmitted from the Coronary Care unit and cardiology clinics to the central PDP-11/60 computer via twisted-pair wiring.

In 1982 the decision was taken to design an improved version of the CART electrocardiograph and to install a transmission network which would allow ECGs to be transmitted from most of the high-usage areas of the hospital, thus allowing a substantial proportion of ECGs to be reported automatically. The ECG analysis program currently runs on a MicroVAX computer linked via an Ethernet to the PDP 11/44 which acts as a front-end to the analysis system, handling communications, archiving and database query functions.

The Analysis Program

The ECG analysis software consists of a suite of programs which are executed in sequence on a set of ECG waveforms which have previously been digitised by a CART and transmitted to the central computer. Each program, or phase as it is known, is responsible for a separate aspect of the analysis process. The initial phases carry out preliminary signal processing on the digitised signal, including removal of low- and high-frequency artifacts such as baseline wander and spikes caused by implanted pacemakers. This is followed by an identification of the PQRST complexes and the formation of an averaged complex for each of the 12 leads based upon an analysis of the morphologies of the various complexes. Next, a set of measurements is made on each averaged complex comprising the amplitudes and durations of the various components of the complex. These measurements are fed into the diagnostic phase which produces an interpretation based on the measurements by means of a decision tree. The diagnostic module contains criteria for 12-lead ECG recordings from both adults and children as well as a facility to analyse 3-orthogonal ECG recordings from an XYZ lead set.

The CART.

The CART is an automated electrocardiograph which has been specially designed for the acquisition and transmission of ECG data[2]. It is based on the Intel 8086 microprocessor and has 32K-bytes of RAM for ECG storage. An alphanumeric keyboard and display allow communication with the operator. During normal operation, the CART carries out the following functions:

- Allows entry of patient details:
- Accepts the patient's name, date of birth, sex, clinical classification and other relevant details which can be edited if necessary during entry.
- Performs various self-calibration and signal-quality checks, informing the operator if there are any problems.
- Digitises, compresses and stores eight seconds of ECG data:
- Compresses the data using a differencing technique with Huffman coding of the residuals. This reduces both the storage requirement and the time taken to transmit the data.
- Checks the data for any possible coding errors.
- Transmits the compressed data to the central computer and waits for confirmation that the data has been stored successfully by the computer.

- Optionally provides a hard copy of the ECG waveform.

The Communications System

The network which has been installed is based around a Broadband co-axial cable system. Stations are interfaced to the network via radio-frequency modems operating at a data rate of 96000 bps. The system is arranged in a multipoint configuration whereby all of the modems on the network operate on the same channel (i.e. with the same transmit and receive frequencies). Control of the network is undertaken by the PDP11/44 minicomputer, situated at the head-end of the network. The communications functions of the PDP11 are undertaken by a DMR11 controller, a microprocessor controlled unit which implements Digital's DDCMP link level protocol independently of the main processor.

The network was initially brought into use by implementing a simple set of communications protocols and a simple polling algorithm. This system operated at the data-link level with only a simple polling protocol and a rudimentary file transfer protocol layered on top of the DDCMP link-level protocol. In order to see if the system could be enhanced by adopting a more complex strategy it was decided to simulate any such strategy in advance of implementation.

A simplified diagram of the system is shown in Figure 18.

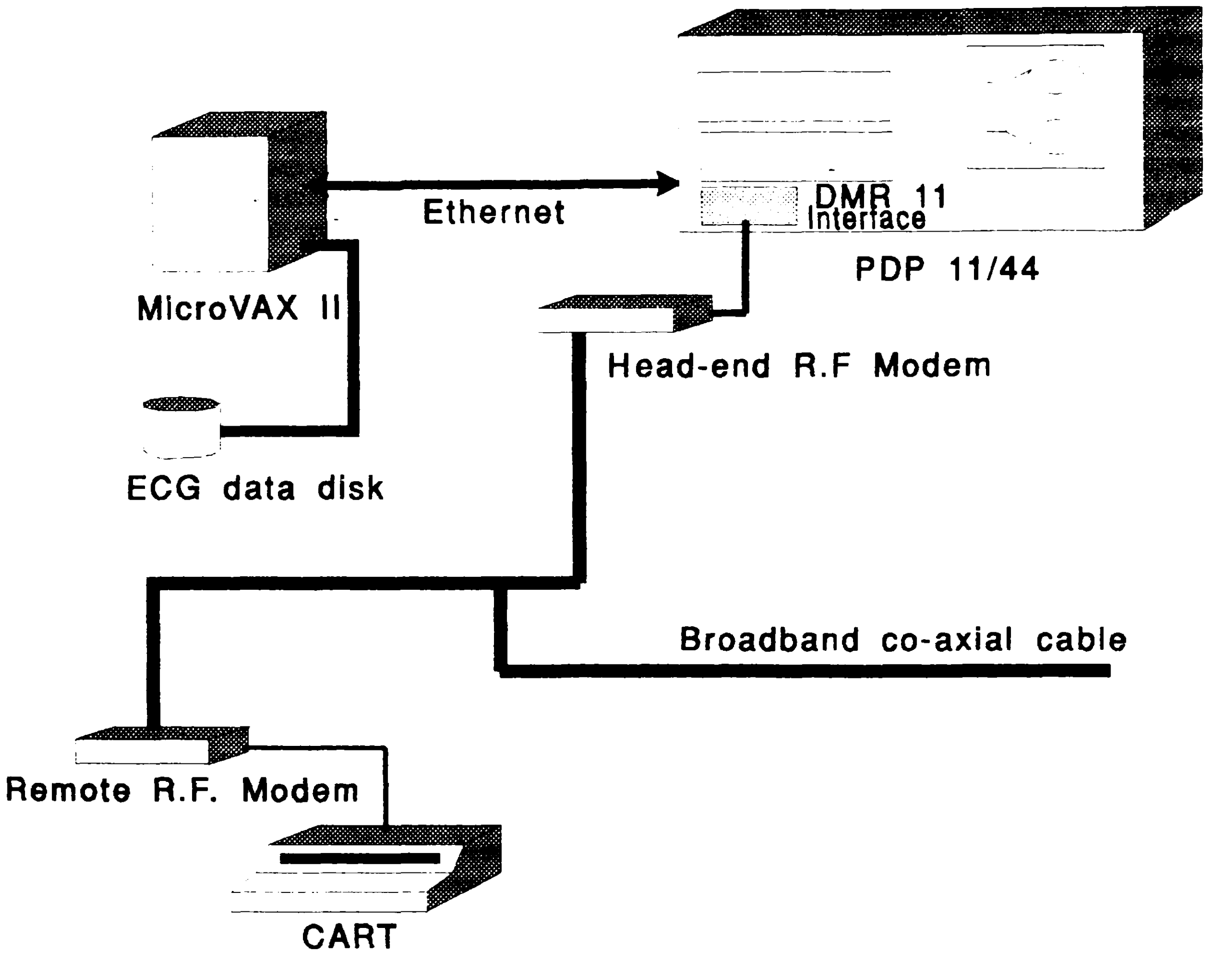

FIGURE 18. Simplified diagram of the Glasgow Royal Infirmary ECG transmission network.

Two different approaches to the design of the communications system are examined here. The first handles transactions with the Carts on a non-multiplexed basis while the second attempts to service multiple Carts simultaneously. Before examining the two systems in more detail however, some of the characteristics of the network and of the system as a whole are examined.

Communications functions required by the Carts

Communication with the central computer is required at two points during the on-line data acquisition process (described previously). These are:

- **At login time**

Each Cart will log in to the network at the start of each recording sequence. The purpose of the login procedure is twofold:

To check that the network is operating properly before going ahead with the recording.

To transfer the current date and time to the Cart so that the time of recording can be printed on any hard copy which is obtained during the recording.

- **After data acquisition**

The Cart will transmit the ECG data to the central computer for storage and analysis. A confirmation message will be sent back informing the operator that the ECG has been successfully stored at the central computer. There is a delay of around 5 seconds between reception of the ECG and transmission of the confirmation message while the ECG is decoded as a check on the integrity of the coded data.

A Conceptual Model for the Network Architecture

Before choosing a communications strategy for the system it was necessary to decide what type of network the present configuration represented and what type of network architecture was best suited to it. The choice of strategy was constrained by features of the access method (in this case polling) and by limitations in the data link layer protocol (DDCMP) both of which were dictated by the choice of network hardware.

As has been described above, the operation of the Cart requires that communication be undertaken with the central computer at two points during the recording sequence. The average time for any one of these transactions is of the order of 1-2 seconds, while the time between any two transactions by the same

Cart will be around 1-2 minutes. The time taken to perform a complete recording cycle is of the order of 5 minutes.

It can be seen that, from the operational point of view, the network resembles a transaction system more than it does any normal local area network configuration. Transaction processing systems require that each user receives a prompt response from the system and also that each transaction is *atomic*, meaning that the total interaction with the central database during the period of the transaction is indivisible: either the entire transaction is completed or the system is *rolled back* to the state prior to the transaction. This is necessary for the correct transmission of ECG data which must be transmitted, verified and successfully written to the central database before the transaction can be successfully terminated. Otherwise the transaction must be aborted and the entire transmission process begun again. A transaction-processing model was therefore adopted as a description of the network to facilitate the detailed design of the communications procedures and protocols.

In the following sections a transaction will denote the period during which a Cart is logged on to the central computer. Using this terminology, the login procedure and the transmission of the ECG data represent two separate transactions.

Subnet operation

The operation of the *subnet* is now considered. This term is taken to refer to those layers which handle the network-dependent aspects of the communications system, which in normal cases means the layers up to and including the network layer which provide services to the transport layer (using the OSI[27] terminology.) There are two broad categories of subnet: those which use virtual circuits and those which transmit datagrams. A full discussion of the relative merits of the two types of system is given in Tanenbaum[28]. It would be convenient, in the context of the present system, to use a datagram-based approach, since the transactions represent short, independently addressed blocks of data. However, as will become clear from the ensuing discussion of the polling protocol, the network exhibits many of the characteristics of a virtual-circuit system because, once a station has been polled and has responded, a point-point link effectively exists between the station and the central computer. In fact, the description chosen depends in large part on the outcome of the design exercise presented below. There is, therefore, a choice of services which the subnet can provide to the transport layer. Within the subnet however, the virtual-circuit description prevails as each data unit delivered by the subnet is preceded by a single polling sequence and delivered in point-point fashion.

The Polling Mechanism

This section describes the method developed to allow the central computer to poll the remote stations. This discussion is limited to communication between the PDP11/DMR11 controller and the Carts.

Due to the fact that the DMR11 communications controller is designed to operate only on point-to-point links and that consequently it was impossible to address data frames individually at the data link level it was therefore necessary to devise an alternative method of addressing the data. The method chosen was to address each message individually using a separate polling protocol, this protocol being implemented with the maintenance mode of DDCMP. For a full description of the DDCMP protocol, the reader is referred to the relevant technical manual.[29]

The DDCMP protocol operates in two distinct modes. The normal **data mode** provides sequenced error-free data frame transmission between the two ends of the link, while the **maintenance mode** provides a simple non-guaranteed broadcast frame transmission service. The maintenance mode provides the foundation for the polling protocol, and the sequence of events during the polling process is as follows (assuming that several stations are awaiting the attention of the central controller):

1.) The controller broadcasts a poll request frame (which contains a remote station address as part of the data field).

2.) The frame is received by each remote station.

3.) The remote stations decode the frame and examine the address field.

4.) The remote station with the corresponding address responds with a poll response frame (also using maintenance-mode DDCMP).

5.) The central controller begins a data-mode START-STACK sequence with the responding station while the other stations return to the listening state.

This mechanism forms the basis of each transaction.

Estimation of Network Delays

One of the most important considerations in attempting to estimate the performance of any given implementation is the determination of the various delays present in the network. Some of these delays are caused by the network hardware (transmission speeds, etc.) and some by the software (operating system overhead, protocol performance etc).

As an example, the delays present during a typical transaction are examined. This transaction consists of the transmission of a message containing five 1024-byte data frames.

Consider the normal message sequence (assuming error free transmission for the moment i.e. no retransmissions are necessary):

```
HOST                              REMOTE

POLL        ---------->

            <----------           POLL RESPONSE

START       ---------->

            <----------           STACK

ACK(0)      ---------->

            <----------           DATA FRAME(1)

ACK(1)      ---------->

                 .

                 .

                 .

            <----------           DATA FRAME

ACK(5)      ---------->
```

The transmission times for the various message types are given in table 1 below for a transmission rate of 96000 bps and a data frame size of 1024 bytes. These timings are calculated from the bit-transmission time and do not take into account any overhead caused by synchronisation and turnaround.

Message Type	Size (in bytes)	Transmission time
POLL	29	T_p = 2.5ms
POLL RESPONSE	29	T_r = 2.5ms
START	10	T_{st} = 0.83ms
STACK	10	T_{sk} = 0.83ms
DATA FRAME	1036	T_d = 86.0ms
ACK	10	T_{ack} = 0.83ms

TABLE 1. Frame times for the message sequence.

As can be seen from the message sequence, each poll/response or frame/acknowledge sequence has a total time comprising the transmission time for the frames involved and the frame processing time in the communicating stations. The Data Frame comprises 1024 data bytes and 12 header bytes.

In addition, a frame processing time (T_{pr}) has been included, as well as a decode-check time. These values were calculated, using the pilot transmission system, as follows.

For each ECG transmitted on the system, the total run time (transmission of the ECG data plus the ECG decode test time) was logged, along with the time taken for transmission of the data alone. Subtracting the transmission time from the total time gives the decode-test time. The total processing overhead for the data transmission was calculated by subtracting the time spent transmitting bits on the line (calculated, in turn, from the data rate and frame lengths) from the transmission time measured during program execution. Dividing by the number of frames transmitted gives the processing overhead per frame. The values used for the run times were the mean values from a sample of 100 ECGs. This processing delay also includes a contribution from the CART. Due to the design of the serial communications controller used in the CART, it was necessary to calculate the Cyclic Redundancy Check (CRC) value for the DDCMP frame header in software rather than using the on-chip CRC generator. This calculation is performed for each data frame transmitted by the CART. The resulting values are:

Processing delay per frame (T_{pr})	130 ms
ECG decode test time	5.73 seconds

In assigning the processing delay the per-frame delay is divided by two to signify that the delay is contributed to by both the transmitting and receiving station. This is not strictly accurate, as the PDP11 contributes the larger share of the processing delay, but serves the purposes of the analysis and simulation adequately. This gives

$T_{pr} = 65$ ms.

If N_p is the number of frames in a message then the total transaction time for one message is:

$$T_{dat} = T_p + T_r + T_{st} + T_{sk} + (N_p \times T_d) + \quad (4.1)$$

$$((N_p+1) \times T_{ack}) + (2N_p \times T_{pr})$$

which, in the case of the 5-frame message, is 1.09 seconds.

The two implementation methods

In any multidrop system, the performance of the network depends fundamentally on the central controller as all communications on the network must pass through it and be controlled by it. There are two ways in which the central controller can service the various remote stations in the present transaction-based system:

a) Non-multiplexed

With this method, the controller will allow a remote station exclusive access to the network for the duration of the entire transaction (i.e. the transmission of an ECG from a Cart or the output of an ECG report to an output station.) All the other stations must wait until the transaction is complete before being serviced by the controller.

b) A Multiplexed system

This method attempts to service all active stations simultaneously by limiting the number of frames which may be transmitted at any one time. The controller behaves somewhat like an operating system task-scheduler, dividing transmission time between competing stations.

For the assessment of the two types of service it is assumed that 6 Carts are simultaneously waiting to transmit ECGs and that the ECG data comprises 15 frames. It is also assumed that the central computer has one ECG report of 20 frames ready to send to an output station.

Analysis of the non-multiplexed system

The performance of this type of system is easily evaluated. The central computer polls each Cart and the output station in turn and, accordingly, grants permission to transmit in turn. Each ECG or ECG report is sent as a single message.

From formula (4.1) above, the time for the transmission of each ECG (N_p=15) is 3.26 seconds. The time to transmit the ECG report from the central computer to the output station is 4.34 seconds. To each of these figures the ECG decode time of 5.73 seconds must be added giving a total transaction time of 8.99 seconds for the ECG and 10.07 seconds for the report. This gives a total time to complete all the transactions of 64.01 seconds. (A decode delay is added to the ECG report transaction in order to simplify the comparison with the multiplexed system analysed later.)

If no processing overheads or delays are assumed other than those already discussed then the time which each Cart or output station would have to wait before starting its transaction would be 9 - 10 seconds multiplied by its position in the polling queue. If, for example, the output station were last in the queue then it would have to wait for approximately 66 seconds before receiving the ECG report, assuming that 6 Carts were waiting to transmit.

Analysis of the multiplexed system

The analysis of this system is considerably more complex than that of the previous system. In the first instance, a model which describes one possible implementation of a multiplexed system is outlined.

The main components of the system are:

- A transaction queue
- A transaction scheduler
- A list of active stations

The Transaction queue

The transaction queue holds details of all the transactions currently in progress. Each entry in the queue holds several pieces of information about a particular transaction. This information includes the number of frames to be transferred during the transaction, the number of frames transferred so far, the address of the remote station and information relating to message buffers etc.

The transaction scheduler

The transaction scheduler controls the multiplexing of transactions between the various remote stations. The scheduler operates in the following manner:

An entry is pulled off the transaction queue. The maximum number of frames allowed in the message is calculated according to the number of active stations and the maximum time considered acceptable for a station to wait for service. A message of size less than or equal to this maximum is transferred between the remote station and the host. If the transaction has not been completed (i.e. there are some frames still to be transferred) then the data in the queue entry is updated and the entry is placed at the end of the transaction queue. Then the next entry is pulled off the queue, and so on.

The active station list

This is a list containing the addresses of all the active stations. The information in this list is used in the calculation of the maximum permissible message size and also to determine which stations are inactive and therefore need to be polled periodically to see if they have any information to transmit.

A net model of the Non-multiplexed system

The net representation of the non-multiplexed system is shown in Figure 19. The net may be divided broadly into three parts. The section at the top of the diagram models the queue of stations waiting to be serviced. The region at the bottom right of the net represents the transmission and reception of the individual data frames of a message. Finally, the small region to the left of this section models the decoding process at the central computer and the return of a confirmation message to the sending station.

The firing of transition T14 de-queues the frames for the next station in the queue while transitions T13 - T8 firing in sequence cause the queue to move up. The transmission mechanism is modelled by the lower section of the net. The transmission and acknowledgement of each frame is modelled individually and when all the frames for a particular station have been transmitted, T33 fires and initiates the sending of the confirmation message from the central computer. The section of the net comprising place P1 and transitions T1-T8 detects if the queue is active and is, strictly, superfluous but is included to maintain compatibility with the multiplexed system described below.

The timings on the relevant places were derived from the pilot system and are similar to those for the multiplexed system discussed below. Finally, place P30 representing the decode check was given a delay of zero for this run. This is because the simulation was being run on a relatively slow computer and the rest of the net would have been idle during the period when the decode delay was active. The place and its surrounding transitions behave predictably and implementation of the delay of 5.73 seconds would merely have slowed down the simulation run without imparting any useful information.

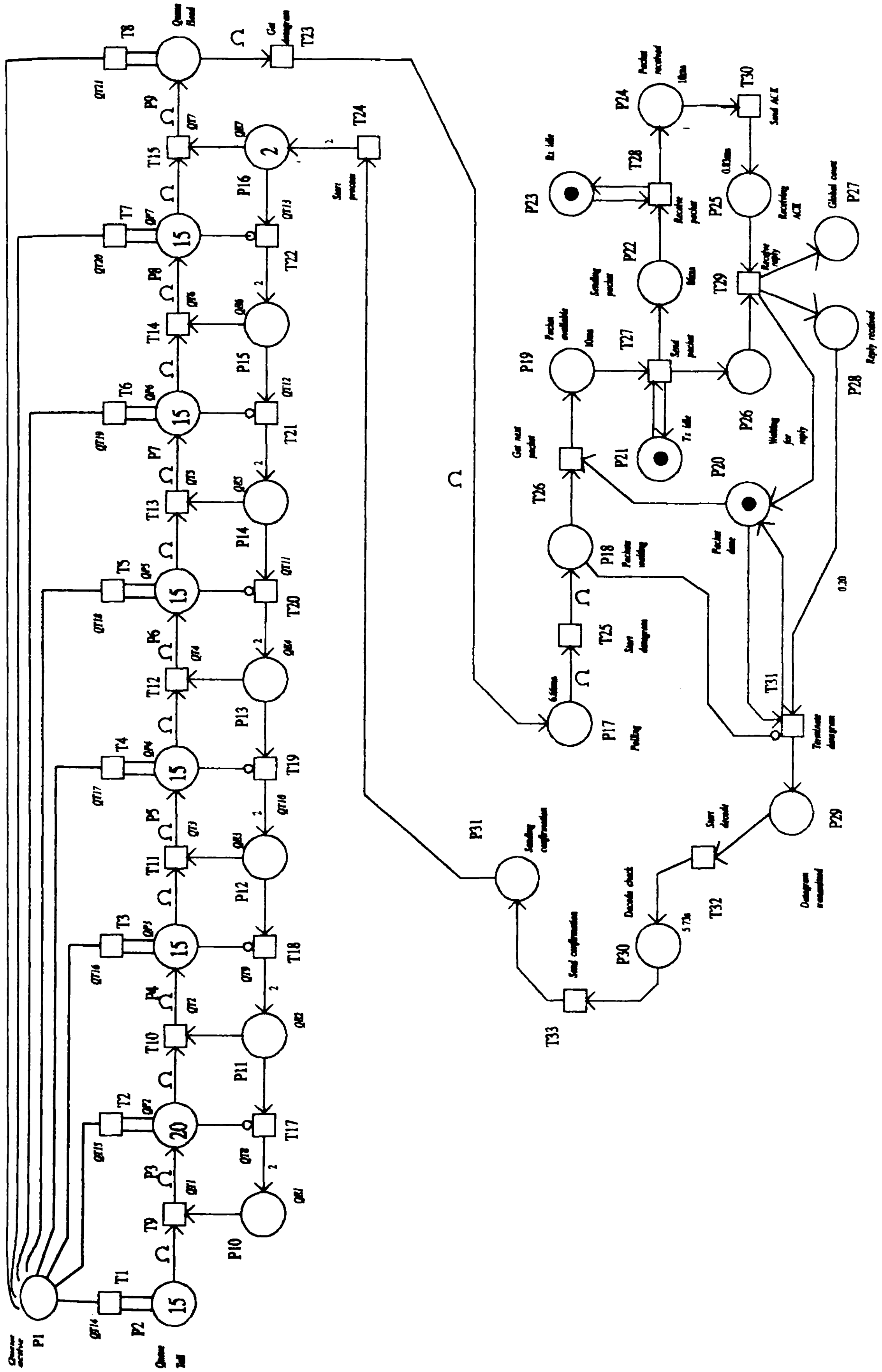

FIGURE 19. Timed Place/Transition Net representation of the Non-multiplexed system

Simulation results

The output report from the simulator is shown in Appendix 3. The total time taken to complete all the transmissions is given by the total elapsed simulation time of 24.04 seconds to which must be added the execution time of the decode check for each of the messages; this gives a total time of 64.15 seconds. The time for transmission of all the frames for one station is given by the mean time between firings of transition T17. This is 3.46 seconds to which must be added the decode check time of 5.73 seconds giving a total time of 9.187 seconds. The transmission time for each frame is given by the time between firings of transition T23. This is 0.218 seconds.

These results are in accordance with those derived from the simple analysis of the system presented above. It should be noted that the time for transmission of all the frames for one station is a mean value in the simulation results. Taking a mean value from the simple analysis (6 ECGs @ 8.99 seconds and 1 report @ 10.07 seconds) yields a result of 9.14 seconds which is in accordance with the simulation result.

This verification is useful because the basic blocks of the non-multiplexed model, the queue and the transmission mechanism, are used in the model of the multiplexed system which follows.

The TPTN representation of the multiplexed system

The TPTN representation of the multiplexed system is shown in Figure 20. Some of the features of this net are carried over from the non-multiplexed model. Of the three elements mentioned in the description of the system given previously (transaction queue, transaction scheduler and active station list), only the first two are explicitly modelled by the net, the active station list being irrelevant to the performance evaluation of the system. (Note, however, that the queue, as modelled by the net, performs some of the functions of the active station list by keeping track of the non-empty queue nodes which represent active transitions). The net consists of two main sections, one of which models the transaction queue while the other models the transaction scheduling/transmission mechanism, and a number of places and transitions which allow communication between the queue and the transmission system.

Once again the net may be broadly divided into several regions. As in the case of the non-multiplexed system, the section at the top of the diagram models the queue of stations waiting to transmit data and the region at the bottom right represents the transmission and reception of individual data frames. The sections at the middle-right and middle-left control the de-queing of transactions and also ensure that the data frames from incomplete transactions go back into the queue. Finally, the section at the bottom left-hand side of the diagram models the decode process and the return of the confirmation to the sending station when a transaction is completed.

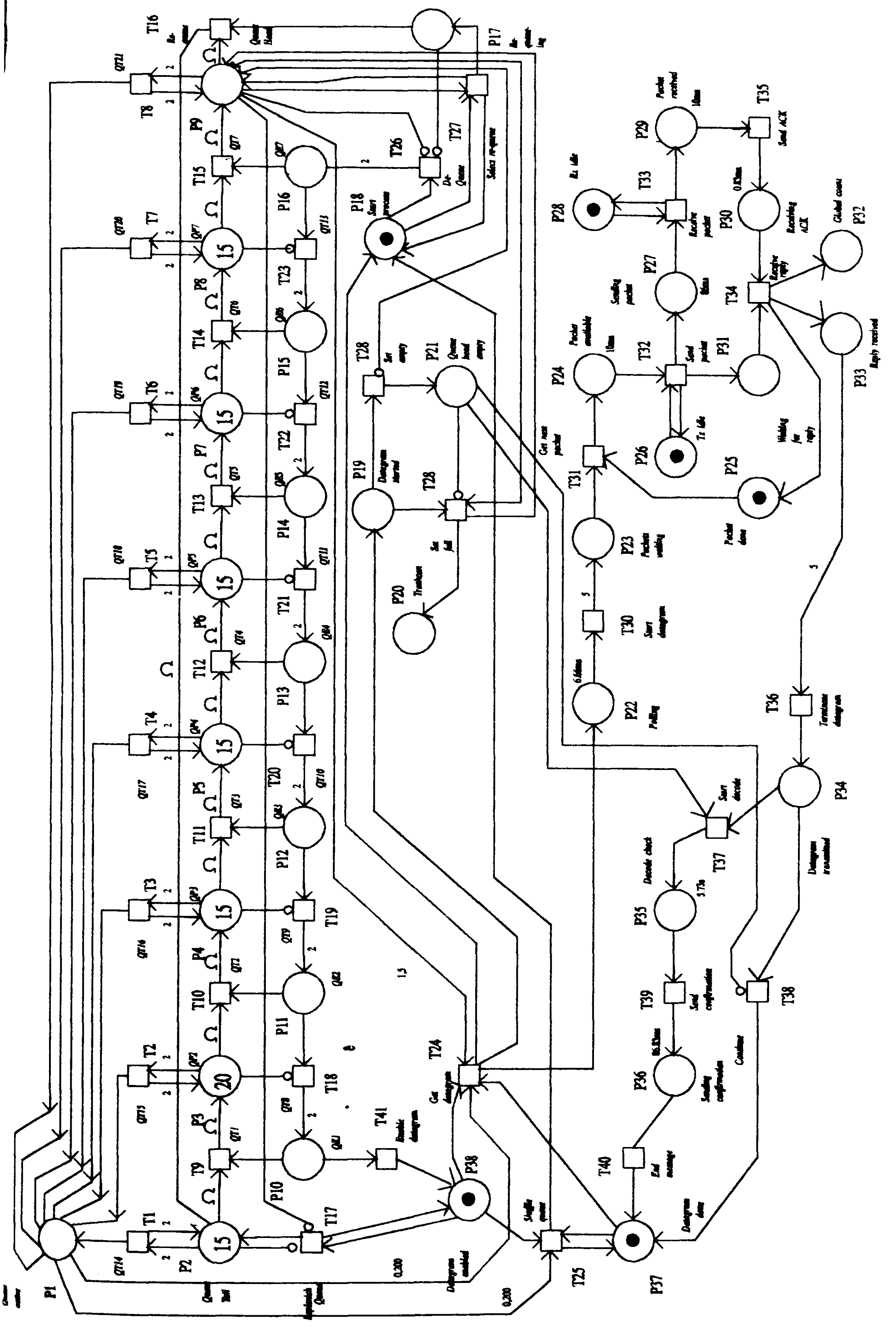

FIGURE 20. Timed Place/Transition representation of the multiplexed system.

The transaction queue

The transaction queue is modelled by the upper section of the net consisting of places 1-17 and transitions 1-23. Places 2-9 represent the queue elements which, in turn, represent the transactions, and the tokens therein represent the individual frames for each transaction. The queue could have been represented as a simple pool containing all of the frames to be transmitted and modelled by a single place, but it was felt that the more detailed representation was more appropriate in that it allowed events to be simulated on a per-transaction basis whereas the simple pool model would only have allowed simulation on a per-frame basis. This is especially relevant in the case where some action must be taken upon the completion of a transaction; the separate treatment of each transaction allows transactions of varying length to be accommodated. Specifically, when each ECG/report transaction has been completed, a confirmation message is returned to the sender. In addition, the model assumes that report transactions will be of a different length from ECG messages.

Omega-weighted arcs are used to link the places representing the queue elements; this ensures that the entire contents of each of the places is passed on the next when the queue is advanced. Also a multi-weighted arc connects the queue head (P9) to T24 so that if five frames are available for the current transaction then they will be de-queued, but if there are less than five available (as is the case with the final message of any transaction whose frame count is not an integer multiple of five) then all of the remaining frames will be de-queued.

It should also be noted that queue operations and message transmission take place concurrently. When T24 fires, delivering a de-queued transaction to the transmission system, a token is also placed in P18, thus enabling the queue to be advanced and partially transmitted transactions to be re-queued.

The transmission mechanism

The transmission system is modelled by the lower section of the net. The model consists of a transmitter, which sends out data frames one at a time and a receiver which responds to each data frame with an acknowledgement. In this model, an error-free link is assumed, i.e. every frame will be acknowledged and no NAKs will be sent. Messages are assumed to be five frames long.

Queue-transmitter communication and control functions

Various places and transitions are included for the control of the queue and transmission system. These include detecting if the last message of a transaction has been de-queued, detecting if there are transactions left in the queue and deciding whether to advance the queue in the absence of frames at the queue-head. Each message is started by the firing of T24, and tokens representing the five frames of the message are deposited in P23 by T30.

Timings

Processing and transmission delays are confined to the transmission section of the net. The actual delays are the same as those used for the first-come- first-served system. The frame processing delays are assigned to places 24 (transmit processing overhead) and 29 (receive processing overhead.) These places are each assigned half the per-frame processing delay. The delay associated with the ECG decode check (5.73 seconds at place 35) was omitted from the simulation model for the same reason as that given above for the non-multiplexed system. The effect of this delay is easily evaluated by noting the number of firings of P35. The delay for the place representing the polling sequence includes the combined delays of the poll, poll response, START and STACK messages. All delays shown are in milliseconds.

Initially, no delays were assigned to section of the net which models the transaction queue as this section of the net executes concurrently with the transmission mechanism as would be the case in a normal multi-tasking system.

The simulation results

The simulation was run with a time limit of 40000 instants and an initial marking as shown in the diagram. The complete output from the simulation is shown in Appendix 2.

Performance parameters derived from the simulation

- **Total time taken to complete all transactions**

The total time taken by the process is given by the simulation time at termination, i.e. 24.63 seconds. To this must be added the time taken by the ECG decode procedure, represented by 7 firings of T37 and 7 delays of 5.73 seconds assigned to P35. This gives a total time of 64.74 seconds.

- **Time taken to transmit one message**

This can be found by subtracting the time of the first firing of T24 from that of T36. This gives a time of 1.09 seconds.

- **Time between messages for any Cart or output station**

This is given by the mean time between firings of T24, i.e. 1.116 seconds. By adding the time of the first firing of T24, the time after which the second station in the queue would receive service is obtained, i.e. 1.117 seconds. Thereafter, stations would be serviced at intervals of 1.116 seconds.

- o Time taken for one Cart to complete transmission of an ECG (Transaction time)

This is given by the mean time between firings of transition T39 (send confirmation) and is equal to 1.361 seconds.

Queuing delays

As an exercise, several simulation runs were carried out in order to find out what magnitude of transaction queuing delay would impinge upon the total execution time of the simulation. It was found that a delay (distributed between places P10 to P16 which control the movement of the queue) of 10 seconds was necessary. This is much larger than any conceivable processing time for servicing the queue, so it would appear that the assumption of no queuing delays does no harm to the overall performance of the model.

Comparison of the multiplexed and non-multiplexed systems

As can be seen, the total execution times of the two systems are almost identical: 64.15 seconds for the non-multiplexed system versus 64.74 seconds for the multiplexed system. This system performs the same processing tasks as the non-multiplexed system, only in a different order and with some additional overheads. The frame transmission times are the same and while the delay in waiting for service is shorter in the case of the multiplexed system, this has to be balanced against the longer transaction time.

Because the overall performance times of the two systems were nearly identical, it was concluded that the choice of system could be made by considering the relative merits of the most important operational characteristics of the systems. These were:

- **Relatively short transmission time with increased waiting time**

  This represents the non-multiplexed system. Any given station will have to wait until all stations which precede it in the transaction queue have completed their entire transactions. However once the transaction has begun it is completed without interference from any other stations.

- **Reduced waiting time with increased overall transmission time**

  This is characteristic of the multiplexed system. Any station with a pending transaction will receive the attention of the central computer relatively quickly, but the time taken to complete the transaction will necessarily be longer than that of the non-multiplexed system.

Choice of system

The choice of system is made easier by considering the behaviour of the pilot network over a period of time. This shows that the number of Carts requiring service at any one time is usually quite small, typically one or two. Thus, in the majority of cases there is little or no delay before receiving service. Taken together with the fact that, from the Cart operator's point of view, it is important to complete the transmission of an ECG as quickly as possible, this makes the implementation of the non-multiplexed system seem more sensible. In view of this, the non-multiplexed method was implemented for the system.

Chapter 5. The simulation of the single-terminal system

In order to investigate the capabilities of the simulator in the evaluation of queue-based models, a system consisting of a single terminal sending characters to a CPU was chosen. A block diagram of the system is shown in Figure 21.

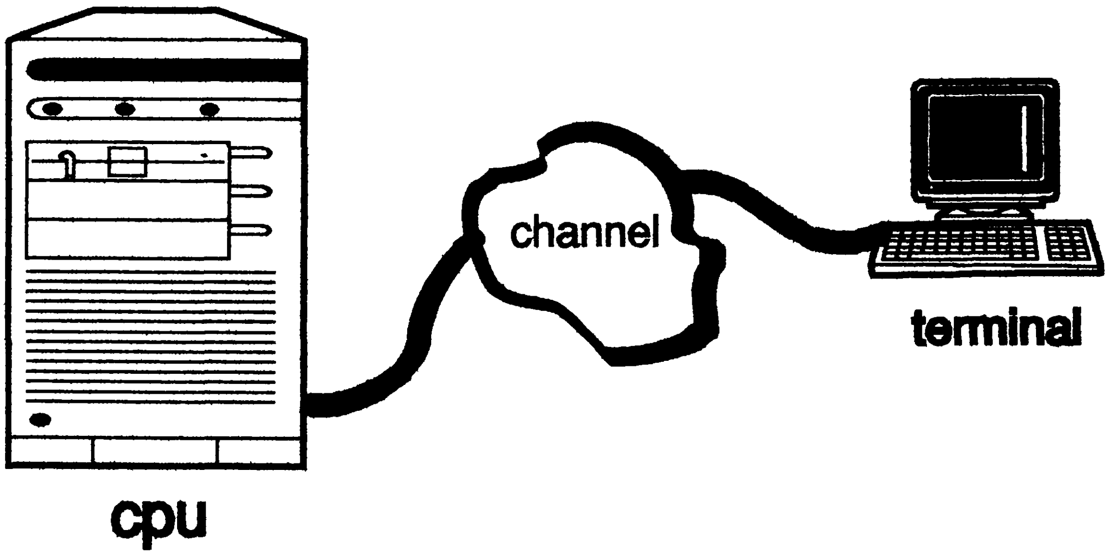

FIGURE 21. The single-terminal system.

The system consists of three elements: a terminal, a communications channel and a CPU. The terminal sends characters at intervals into the channel where they are processed by the CPU. The operation of the system may be considered to proceed in a series of discrete time-slots. During each time-slot, the terminal may transmit a random number of characters into the channel, and during the same time-slot the CPU may remove a single character from the channel for processing.

An analytic model for the behaviour of this system has been derived by Konheim and Chu[30] and the aim of this particular simulation was to compare the results of this analytical model with the results produced by the simulator.

The Petri Net model of the single-terminal system

The Petri net model for the single-terminal system is shown in Figure 22. The model corresponds closely to the actual system in terms of the elements modelled and the behaviour of those elements. The following paragraphs describe the operation of the model.

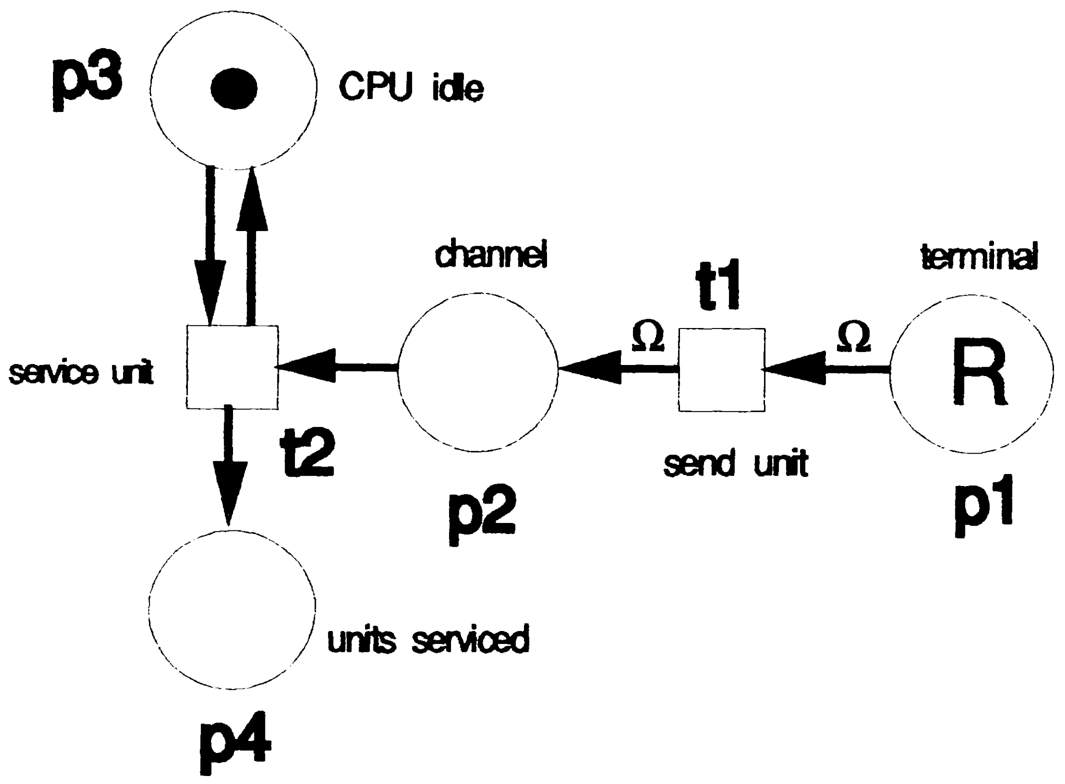

FIGURE 22. Petri net representation of the single-terminal system.

The terminal is modelled by a random place (denoted by the large "R" in the diagram) whose mean token count is fixed for a given simulation run. At each simulation instant a random number of tokens is deposited in the place and become available immediately, there being no time delay attached to the place. Each token corresponds to a single character or data unit. The set of random numbers used to feed the place during the course of the simulation has a Poisson distribution in order to maintain a correspondence with the analytic model.

The channel is modelled by a normal place and is fed by the terminal place via a transition (t1) with two omega-weighted arcs. Thus at each firing of transition t1 all of the tokens in the terminal place are deposited in the channel place.

The CPU is modelled by a place and a transition (p3 and t2.) The place represents the CPU-idle state and enables the transition to remove a single token from the channel place during each instant. A fourth place, entitled "Units serviced," serves as a sink for the processed characters.

It should be apparent that this model, although very simple, corresponds very closely to the operation of the system as specified above and as such should be able to provide a useful comparison with the analytic model.

The analytic model of the single-terminal system

The model developed by Konheim and Chu allows the mean and variance of the length of the queue represented by the channel in the single-terminal model to be determined given the distribution of data units entering the channel from the terminal. It is convenient to assume that each data unit is simply a single character from the terminal, although this makes no difference to the operation of the model. The equations for the mean (E) and variance (var) of the queue length are:

$$E(L^*) = \frac{1}{2}\frac{\sigma^2}{1-\mu} + \frac{1}{2}\mu \qquad (1)$$

$$\mathrm{var}(L^*) = \frac{\mu_3}{3(1-\mu)} + \left[\frac{1}{2}\frac{\sigma^2}{1-\mu} + \frac{1}{2}(1-\mu)\right]^2 - \frac{1}{12}(2\mu-1)(2\mu-3) \qquad (2)$$

where

L^* is the length of the buffer queue in data units.

μ is the mean of the input distribution, i.e. the mean number of data units emitted by the terminal at each instant.

σ^2 is the variance of the mean of the input distribution.

μ_3 is the 3rd central moment of the input distribution.

The simulation

The simulation was designed to test the equations from the previous section and to find out how accurately they modelled the single-terminal system. The method used was to conduct a series of simulation runs, varying the mean of the input distribution between the limits of 0.01 and 0.99. This was easily achieved by varying the mean number of tokens which could appear in the **terminal** place (p1) at each instant. The various probabilities were measured in terms of time units which corresponded to the transmission time for a single data unit, normalised over the total number of time units which elapsed during the simulation.

The net specification file is shown in Figure 23.

```
;
; Net specification file for the single terminal simulation.
;
net Single_terminal_simulation
    {
    run_time 300
    }

random_place terminal
    {
    type random
    capacity 100
    marking 0
    delay 0
    distribution 0.5 10.0
    }

transition send_unit
    {
    input terminal omega

    output channel omega
    }

place channel
    {
    type normal
    capacity 500
    marking 0
    delay 0
    }

place cpu_idle
    {
    type normal
    capacity 2
    marking 1
    delay 0
    }

transition service_unit
    {
    input channel normal 1
    input cpu_idle normal 1

    output cpu_idle normal 1
    output units_serviced normal 1
    }

place units_serviced
    {
    type normal
    capacity 1000
    marking 0
    delay 0
    }
```

FIGURE 23. The net specification file for the single-terminal simulation.

The **output_special_data()** function of the simulator (see Chapter 3) was modified to produce an output file which contained the following information for each simulation run:

- **The input mean**

  This was the mean number of tokens appearing at the terminal place p1 at each instant.

- **The input variance**

  This was the variance of the mean described above.

- **The 3rd central moment of the input distribution**

  Taken from the μ_3 field of the stat_counter variable associated with place p1.

- **The measured mean of the channel queue length**

  This was the mean number of tokens in the channel place (p2) at each instant, representing the mean number of characters queued for processing.

- **The measured variance of the mean of the channel length.**

  The variance of the mean number of tokens in the channel.

The resulting data were read into a spreadsheet package where the two quantities $E\,(L^*)$ and var (L^*) were calculated using the input mean, variance and 3rd central moment from each simulation run. These values were then compared against the mean and variance of the channel length measured during the simulation runs. The exercise was repeated three times as a consistency check and the resulting data were averaged to produce the final results.

Simulation results

The graphs of Figures 24-27 show the results for the mean and variance of the channel length. Figure 24 shows the measured and calculated mean channel lengths plotted against the mean of the input distribution. The "measured" mean is the actual mean value measured during the course of the simulation while the "calculated" value is that calculated using Konheim and Chu's equations. Figure 25 shows the corresponding graphs for the measured and calculated variances.

It can be seen that the calculated mean channel length follows the measured mean very closely until the mean of the input distribution reaches about 0.9. This is demonstrated more clearly in Figure 26 which shows the same graph with the y-axis having a logarithmic scale. This is to be expected as the mean channel length as calculated from equation 1 above tends to infinity as the input mean approaches 1. However, the correspondence to the measured values is good over most of the range.

In the case of the variance, this too gives a close fit, up to an input mean value of about 0.75 where it diverges for the same reason as the mean. Once again the divergence is made clearer by showing the y-axis on a logarithmic scale in Figure 27.

The reason that the measured mean and variance values do not tend to infinity along with the calculated values id probably due to the relatively short duration of each simulation run. This restriction was imposed by the slow hardware (a VAX) upon which the simulation program was being run at the time.

Conclusion

The simulation provides a useful vindication of Konheim & Chu's model of the single-terminal queueing system. The divergence of the results of the results of the equations from those produced by the simulation for values of the input mean close to 1 is not too serious a discrepancy. In real life, the output from a terminal is necessarily sporadic and it is unlikely that a typist could achieve a constant flow of characters for very long as would be the case as the input mean approached a value of 1.

This example also shows that the Petri net simulator is useful in this type of situation and provides a simple and concise model of the queueing system.

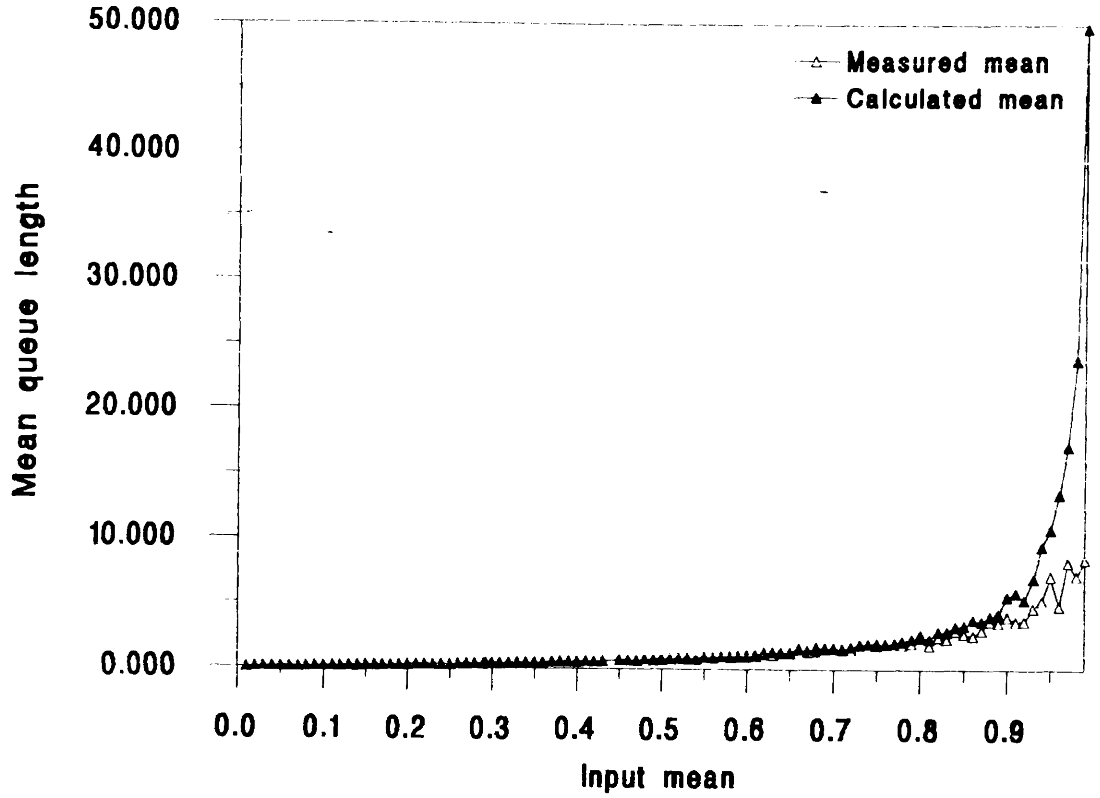

FIGURE 24. Graph of the measured and calculated mean channel lengths for the single-terminal system simulation.

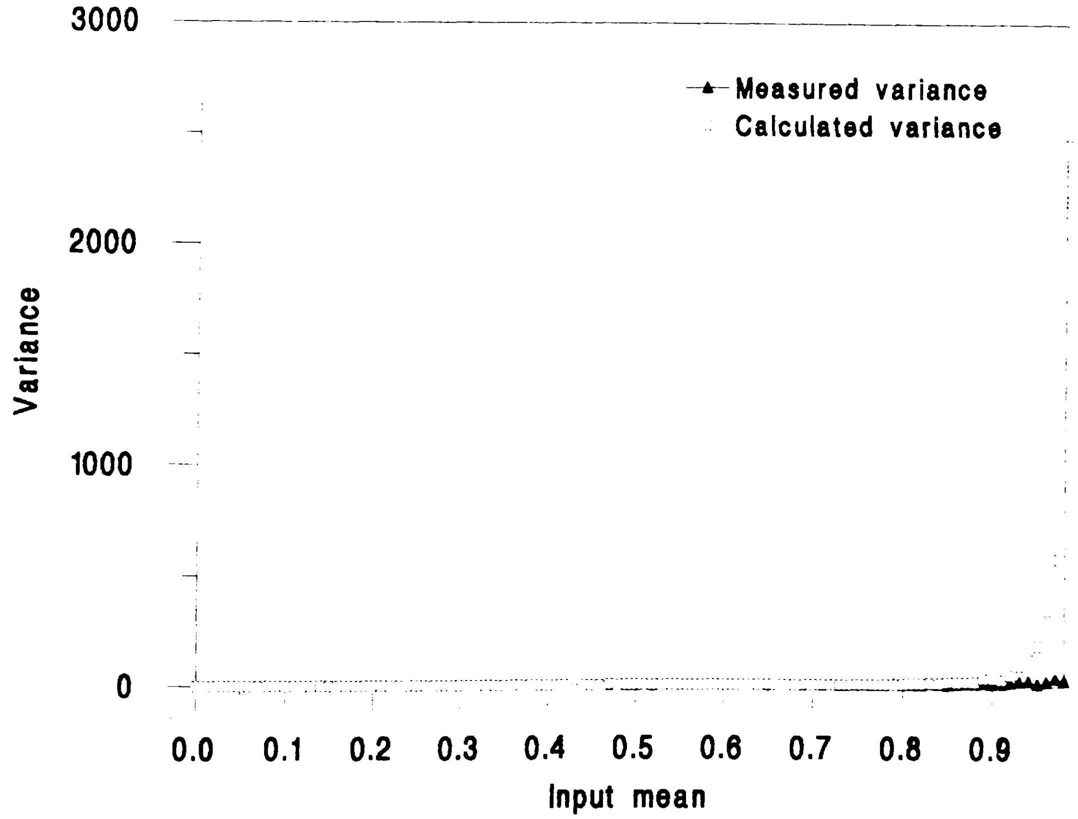

FIGURE 25. Graph of measured and calculated variance of the channel length for the single-terminal system simulation.

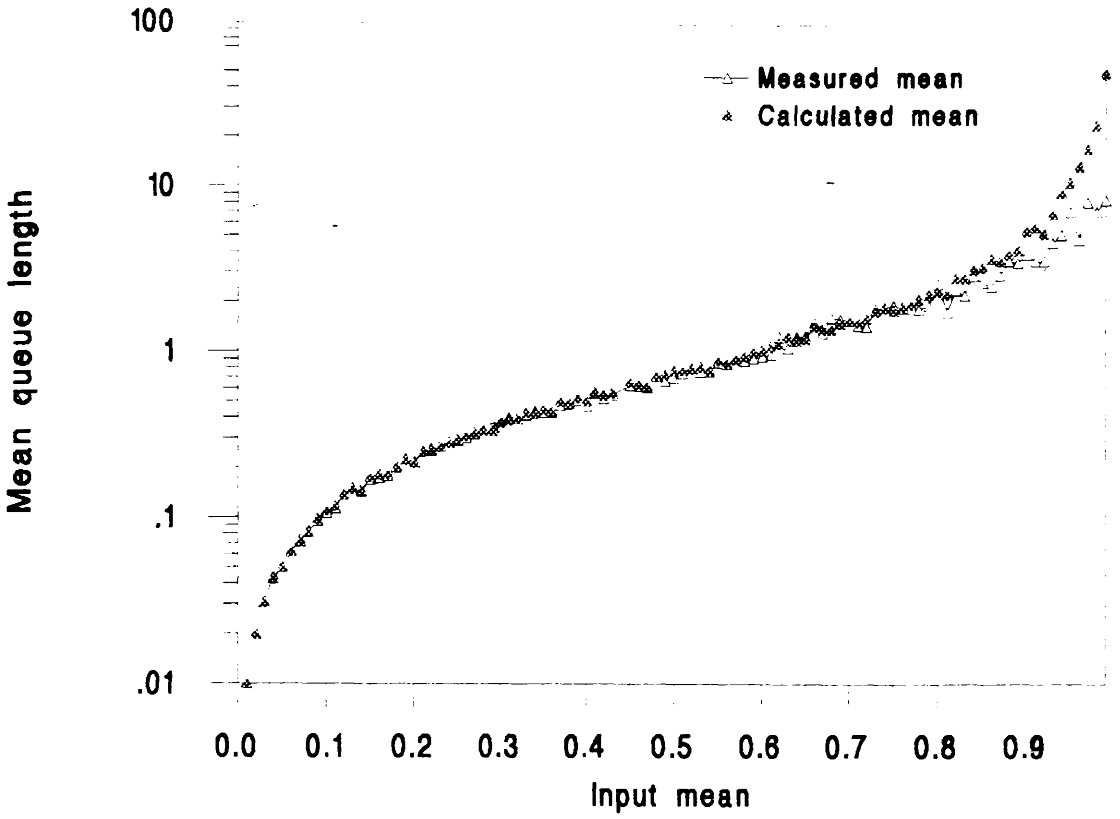

FIGURE 26. The graph of Figure 24 with logarithmic y-axis.

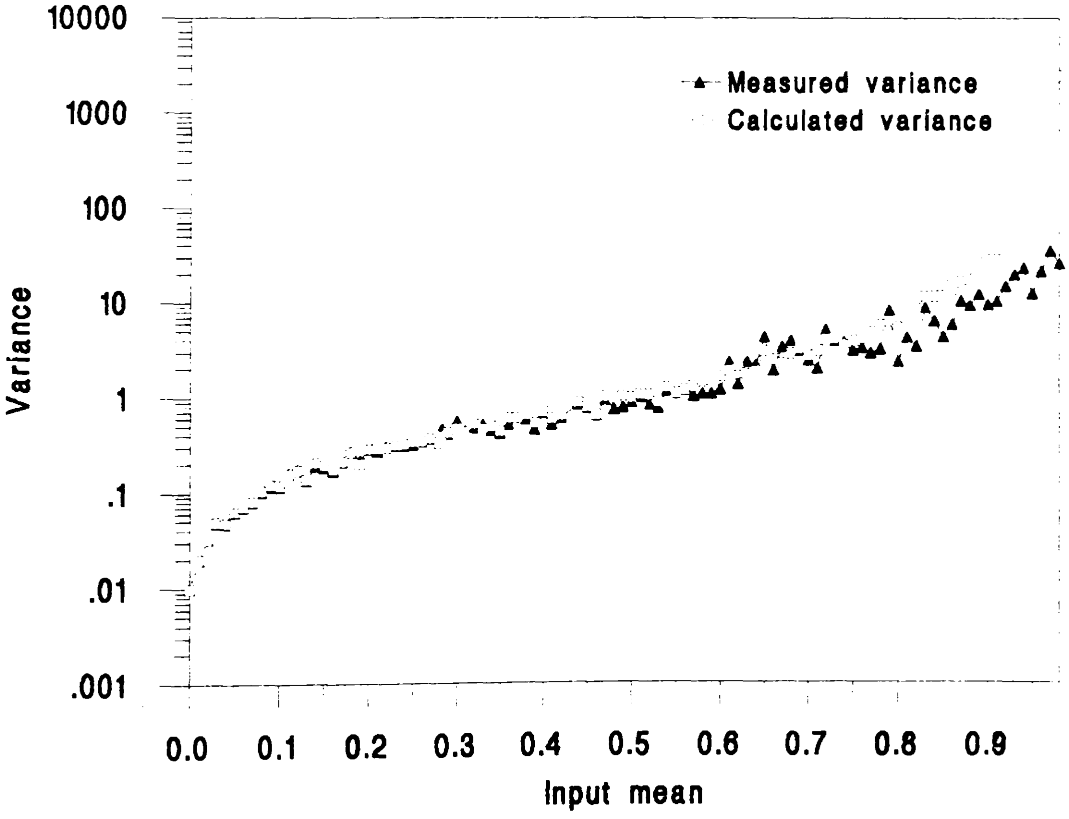

FIGURE 27. The graph of Figure 25 with logarithmic y-axis.

Chapter 6. Simulation of an Ethernet System

In this chapter, the Timed Petri Net simulation method is applied to the simulation of an Ethernet system. Ethernet is a widely-used local-area networking system employing a link-level protocol known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD.)

The basic CSMA protocol was first used in packet radio networks[31] linking together communicating stations spread over several widely dispersed campuses. The system was refined by Metcalfe and Boggs[32] with the addition of the collision detection capability which improved the throughput of the protocol under medium-to-high traffic loads, leading to the specification of the Ethernet protocol by Xerox, Intel and Digital Equipment Corporation[33].

Performance issues in Ethernet design were investigated at an early stage by Shoch and Hupp[34] and various formal analyses and performance analysis methods have since been applied to the system. More recently the use of stochastic Petri Net models has been advocated to analyse the performance of the Ethernet protocol, e.g. Gressier [35] and Ajmone Marsan et al [36].

The work described in the paper by Ajmone Marsan et al[36] formed a useful starting point for the simulation presented in this chapter, as the authors provide a working stochastic Petri net model for the Ethernet protocol along with usable values for the timed portions of the net. The first section of the chapter deals with the conversion of this model into a form which is usable by the current Petri net simulator and the reproduction of the performance indices involved. This exercise presents a practical application of some of the issues described in Chapter 2, in particular the equivalence between timed-transition and timed-place models. The second section attempts to extend the simulation to model the behaviour of the Ethernet under high traffic loads.

The Ethernet transmission strategy

The Ethernet is a broadcast network in which all of the attached stations have equal access to the transmission channel. In the initial state, any station is free to transmit packets on to the channel as long as the network is idle. Each station is equipped with the means to detect whether the channel is idle and also when a packet collision occurs, which may happen if two or more stations attempt to transmit simultaneously. If a station detects a collision during the course of transmitting a packet, the *back-off* strategy is used to resolve the conflict. First, the station will transmit a *jam* signal on to the channel, which ensures that any other stations involved in the collision are aware that the collision has taken place. The station then waits a random length of time before attempting to transmit again. This random delay reduces the probability that the competing stations will attempt to re-transmit simultaneously.

The Stochastic Petri Net Model of the Ethernet

A complete description of the original stochastic net model is given in the paper by Ajmone Marsan et al.[36], but some of the more important features will be discussed here. Figure 28 shows the Petri net diagram for one station on the Ethernet. The model used in this chapter consists of six stations all of which are identical except for the two stations at either end of the bus which lack some of the elements modelling propagation of messages along the bus. This model uses timed transitions of which there are two types. Those represented by the black-filled rectangles are deterministically timed, meaning that the delay assigned to the transition remains constant throughout the simulation, while those represented by the empty rectangles are exponentially timed, in which case the delay varies randomly with an exponential distribution. The empty squares represent instantaneous transitions as usual. The exponentially-timed transitions are used to model the *back-off* process, which represents the exponential back-off and retransmit strategy of the CSMA/CD protocol and the *think* process which represents the (unspecified) behaviour of the station between transmitting packets. The variation in the *think* delay controls the rate at which each station will attempt to transmit packets on to the network. The use of an exponentially distributed firing time for the *think* places fulfils the usual assumption of Poisson arrivals at the transmission queue.

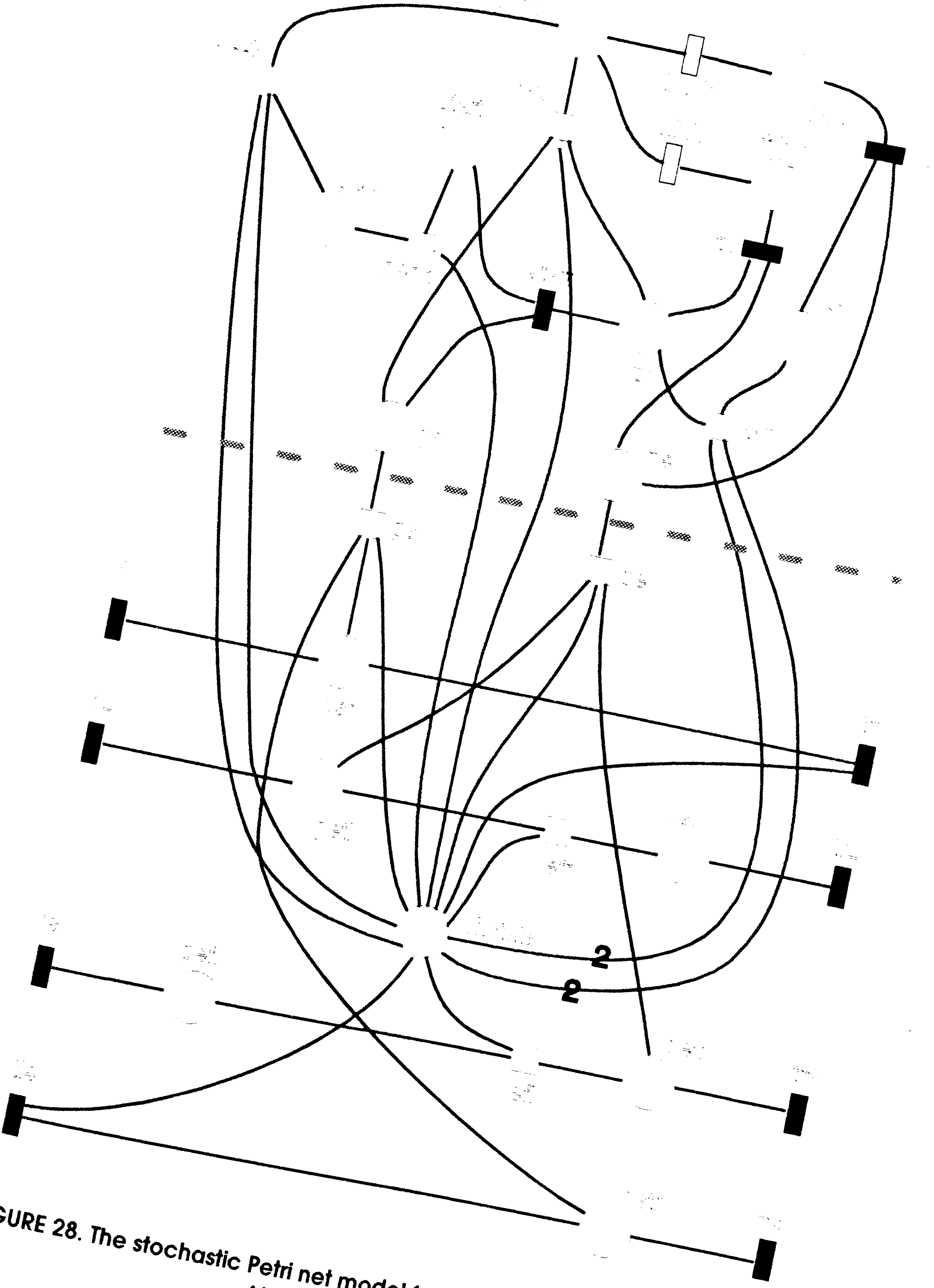

FIGURE 28. The stochastic Petri net model for a single Ethernet station (after Ajmone Marsan et al.)

The net components above the thick horizontal line in Figure 28 model the activity of the station itself, while those below the line model the propagation of packets along the bus. A short description of the function of each element of the net follows.

1.) Net elements belonging to the station

a.) Places

think	a token in this place signifies that the station is processing data and is not attempting to transmit.
sense	this place represents the station testing to see if the channel is idle.
pers	this represents the waiting state while the channel is busy.
start	the start of transmission.
tx	denotes that the station is in the process of transmitting a packet.
jam	a collision has been detected and the station is transmitting a jam signal.
back	the station is in the *back-off* state.
ps	propagate start-of-transmission to the other stations of the net.
pe	propagate end-of-transmission to the other stations.

b.) Transitions

think	exponentially-timed transition which simulates the station's processing activity.

idle this transition fires when the channel is idle and the station is sensing the state of the net.

busy fires when the station is in the *sense* state and the channel is busy.

pend when a station is waiting (token in the *pers* place) this transition remains disabled until the channel becomes free.

start a deterministically-timed transition representing the start-of-transmission delay.

tx a deterministically-timed transition representing the time taken to transmit a data packet.

coll this transition fires when the station is transmitting (token in place *tx*) and a collision is detected on the channel. A collision is represented by two or more tokens in place *chstate*, signifying that two or more stations are trying to transmit simultaneously.

jam deterministically-timed transition representing the sending of the jam signal.

back exponentially-timed transition representing the random back-off delay.

2.) Net elements belonging to the channel

a.) Places

pstl a token here begins propagation of the start of this station's transmission to the station on the left.

pstr starts propagation of start-of-transmission to the station on the right.

petl starts propagation of the end-of-transmission to the station on the left.

petr starts propagation of end-of-transmission to the station on the right.

pefl signifies that the station to the left has propagated an end-of-transmission.

pefr signifies that the station to the right has propagated an end-of-transmission.

chstate

represents the state of the channel. The number of tokens in this place represents the number of stations which are attempting to transmit at any given time.

b.) Transitions

ps propagate start-of-transmission.

pe propagate end-of-transmission.

efr fires when the station to the right propagates an end-of-transmission. Removes a token from place *chstate* to indicate that there is one less station transmitting.

efl similar to *efr* but handles the station to the left.

tl1-tl4, tr1-tr4

deterministically-timed transitions which represent the propagation delay of the channel.

The Timed Petri Net model of the Ethernet

The timed Petri net model used in this chapter was derived from the stochastic net model described in the previous section. Where possible, a direct translation of the net was made on a place-for-place and transition-for-transition basis and for a detailed description of most of the places and transitions the reader is referred to the descriptions given above for the stochastic net model. This translation was possible in the case of normal (untimed) places and instantaneous transitions, but the translation of the timed transitions (both deterministic and exponential) of the stochastic model into equivalent timed-place representations required some slight modification of the net structure. In particular the mechanism whereby the transmission of a packet may be interrupted by the detection of a collision required some extra work using the firing priority which is implicit in the present simulator. This will be discussed later.

The timed Petri net model for a single station on the Ethernet is shown in Figure 29. As in the case of the stochastic model described above, the net used in the simulation consists of six such stations, identical except for the two stations at the ends of the bus. The deterministically timed transitions ***start***, ***tx*** and ***jam*** have been replaced by normal instantaneous transitions and the time element has been moved to the corresponding places ***start***, ***tx*** and ***jam***. Similarly the exponentially-timed transitions ***back*** and ***think*** have been replaced with instantaneous transitions and the time moved to the randomly-timed places ***back*** and ***think***. These places have exponentially distributed delays, so that the behaviour of the net is preserved. The deterministically-timed places are inscribed with the legend $\tau = n$, where n is the delay associated with the place. Similarly the random-delay places are inscribed with the legend $\mu = n$ where n is the *mean* delay associated with the place. All of the timed places have parallel normal places associated with them so that whenever a token enters a timed place, another token also enters the associated normal place. This is purely for the purpose of gathering statistics - the time spent by a token in the normal place matches the delay time of the associated timed place and the simulator does not include unavailable tokens in the measurement of statistics. This means that during the period when a token in a timed place was unavailable the time-spent variable for the place would not be updated, leading to erroneous time-spent values for that place.

The transmission mechanism

Since no mechanism exists in the present simulator for pre-emption of place delays, a rather convoluted process was contrived to model the interruption of a packet transmission when a collision is detected. When transmission of a packet begins, one token is placed into each of the places ***tx*** and ***tx active***. This latter place acts as an enabling place for the transition ***coll***, which fires in the case of collision detection. When this happens the token is removed from place ***tx active*** and a new token is deposited in place ***tx kill***. This disables the transition ***tx*** and enables transition ***tx disable*** so that when the delay of place ***tx*** expires the token in that place is removed and deposited in place ***trash*** thus aborting the transmission. The delays assigned to the jam, backoff and think mechanisms ensure that the transmission delay will elapse before another token is deposited in place ***tx***.

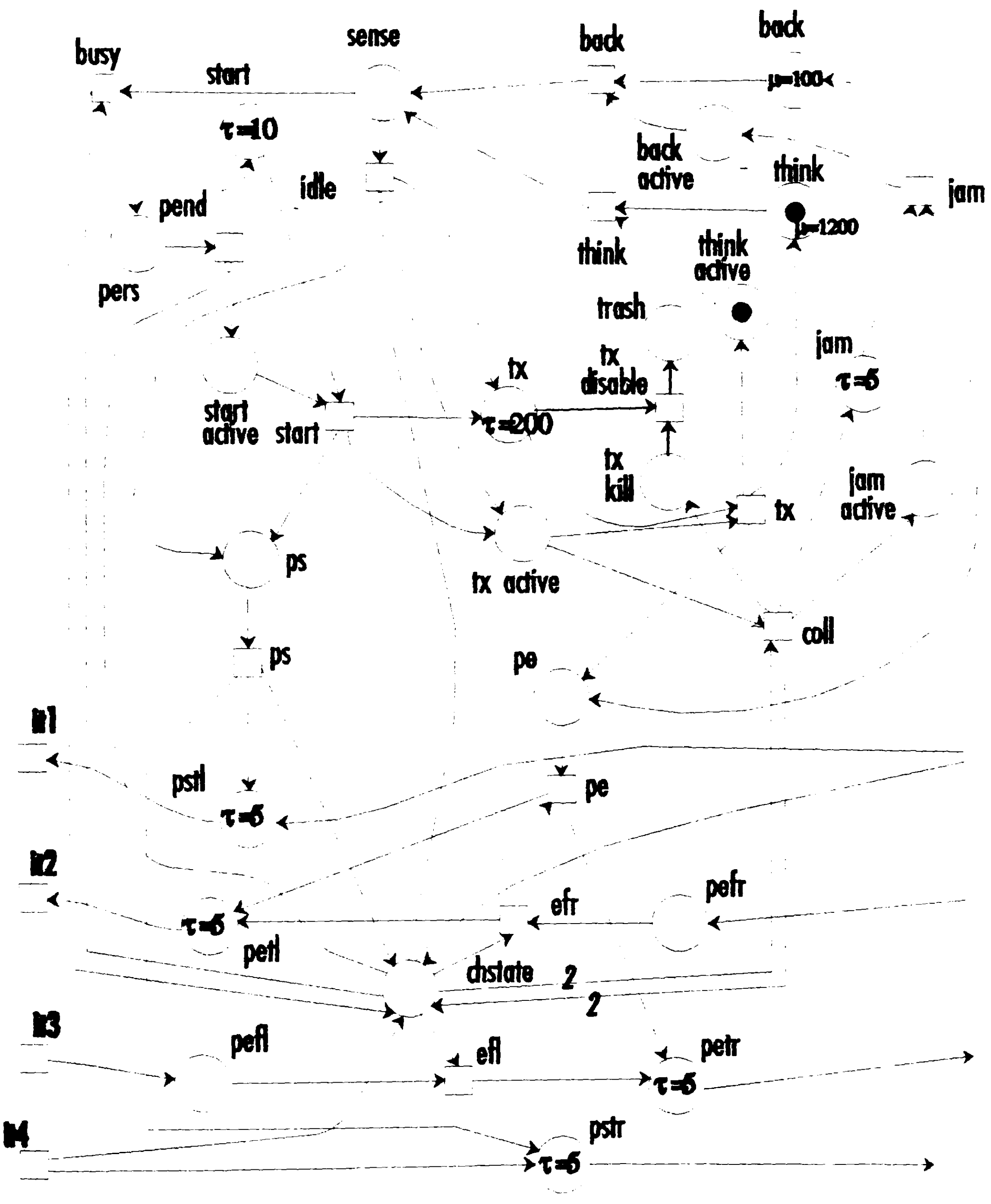

FIGURE 29. Timed Petri net representation of a single Ethernet station.

The propagation delays associated with the ethernet channel and which were implemented by the timed transitions ***tl1*** - ***tl4*** and ***tr1*** - ***tr4*** in the stochastic model are associated in the present model with the deterministically-timed places ***pstl***, ***petl***, ***petr*** and ***pstr***. The transitions themselves have been replaced in the present model with instantaneous transitions it1-it4.

Time parameters

The time parameters for the net were taken from the stochastic model and are shown in table 2. This shows the value of each delay in microseconds along with the place in the timed net which implements the delay.

Parameter	Delay	Associated place
Channel propagation	30μs	***pstl,petl,pstr,petr***
Packet transmission	200μs	***tx***
Jamming delay	5μs	***jam***
Start delay	10μs	***start***
Mean backoff delay	100μs	***back***
Mean processing delay	1200μs	***think***

TABLE 2. Timing parameters for the Ethernet simulation.

In order to check the timed Petri net model against the results of the stochastic model, a mean processing delay of 1200μs was chosen for each of the ***think*** places of the net. This corresponds to the probability of $\frac{1}{6}$ which was assigned to the corresponding exponentially-timed transitions of the stochastic model.

The simulation run

The simulation was run for a period of 120000 instants, each of which represented 1μs of real time. This was found to be long enough for the system to reach a steady state. The computer used was a Sun 4/360 and the total processing time for the run was approximately 10 minutes.

The output from the simulator was read into a spreadsheet package where the various performance indices could be computed from the raw output data.

Performance indices

In the paper by Ajmone Marsan et al, the following performance indices were calculated:

$$THINK = \sum_{i=1}^{6} E\{\#think_i\}$$

$$BACK = \sum_{i=1}^{6} E\{\#back_i\}$$

$$WAIT = \sum_{i=1}^{6} E\{\#wait_i\}$$

$$START = \sum_{i=1}^{6} E\{\#start_i\}$$

$$JAM = \sum_{i=1}^{6} E\{\#jam_i\}$$

$$CHBUSY = P\{\#chbusy_1 \geq 1 \text{ or } \#chbusy_2 \geq 1 \text{ or} \ldots \#chbusy_6 \geq 1\}$$

An explanation of each of these and its equivalent with regard to the present timed Petri net model is given now.

- *THINK*

  This is the steady-state average number of stations processing but not actively transmitting data. This is calculated by summing the mean markings for all of the ***think_active*** places in the net.

- *Back*

  This is the average number of stations in the back-off state after detecting a collision during transmission and is formed by summing the mean markings of the ***back_active*** places.

- *Wait*

  This is the mean number of stations waiting for the channel to clear. This is formed from the sum of the mean markings of the ***pers*** places.

- *Start*

  This represents the mean number of stations starting transmission, i.e. preparing to transmit after having detected that the channel is idle. It is formed from the sum of the mean markings of the ***start_active*** places.

- *Jam*

  The mean number of stations enforcing a jam after detecting a collision. This is formed from the sum of the mean markings of the ***jam_active*** places.

- *Chbusy*

  Channel busy. This is the probability that any of the chstate places of the net contains one or more tokens. This was measured by using the **zero stat_counter** associated with each of the ***chstate*** places of the net. At the end of the simulation, the number of instants during which each place was empty was subtracted from the simulation run time. The chbusy parameter was calculated by taking the mean of this quantity across all of the places and dividing by the simulation run time.

Comparison of performance indices

Table 3 shows the values of the performance indices described above for both the stochastic model and the present timed net model.

Performance index	Stochastic model	Timed Petri net model
THINK	3.84	3.38
BACK	1.08	1.02
WAIT	0.36	0.73
START	0.035	0.087
JAM	0.014	0.0505
CHBUSY	0.73	0.74

TABLE 3. Performance indices for the stochastic and timed Petri net simulations.

It can be seen that the two models are in agreement on most of the indices, including the most important one, that of ***channel busy***. This gives an indication of the degree of saturation of the Ethernet channel which is the most important limiting factor in Ethernet performance under high loads.

The discrepancies between the results for the two models possibly arise from the different methods used to implement the random elements of the models. This could be investigated further by performing the simulation of the stochastic model using the appropriate simulation program. Unfortunately this was not possible at the time of writing.

Simulation of the Ethernet under high loads

Once it had been verified that the timed Petri net model of the Ethernet behaved in a similar manner to the stochastic model, it was decided to evaluate the performance of the Ethernet under varying load conditions.

The six-station Ethernet model described in the previous sections was used, but the packet length was reduced to 128 bytes or 102μs. The reason for this choice was that this was the packet length used in the original measurements made by Shoch and Hupp in 1979.

The simulation method used was to create a net specification file containing all the net parameters except the mean processing delay (the mean delay for the ***think*** places.) A Unix shell script was written which would generate a specification file for each of the required values and run the simulation accordingly. The mean processing delay was varied between 50μs and 6000μs in increments of 50μs.

The **output_special_data()** function of the simulator was used to produce a series of results for each simulation run which were appended to a single output file so that the results of all the individual simulation runs could be processed together.

Results

The limiting factor on the performance of an Ethernet under high load is the packet throughput in the presence of a large number of collisions. To this end two quantities were measured: the probability that the channel was busy, i.e. that a packet was being transmitted over the network and the number of packets transmitted per second. Each of these quantities was measured against the cumulative transmission probability of all the stations. This was the sum of the probabilities that each station would transmit a packet at a given instant.

Figure 30 shows the channel busy probability plotted against this cumulative transmission probability. It can be seen that the channel usage approaches 1 as the cumulative transmission probability approaches a value of 6, i.e. each station's individual transmission probability approaches 1.

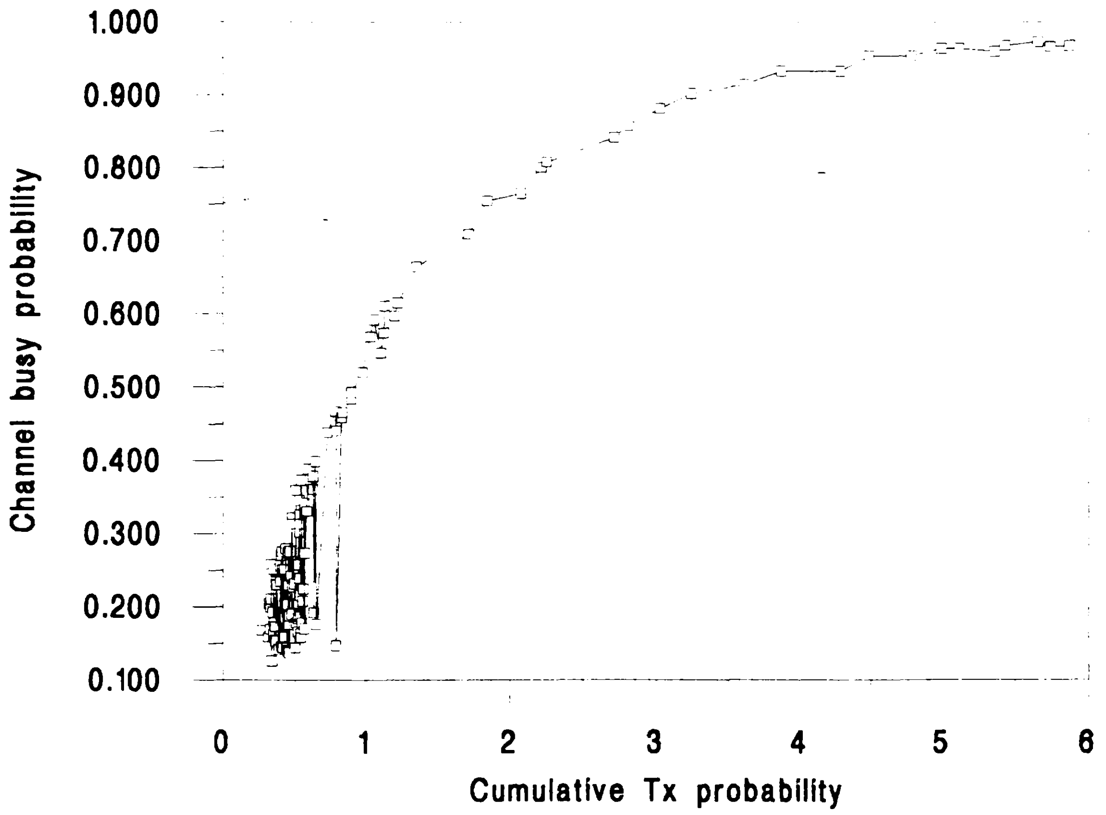

FIGURE 30. Graph of channel busy probability for the Ethernet simulation

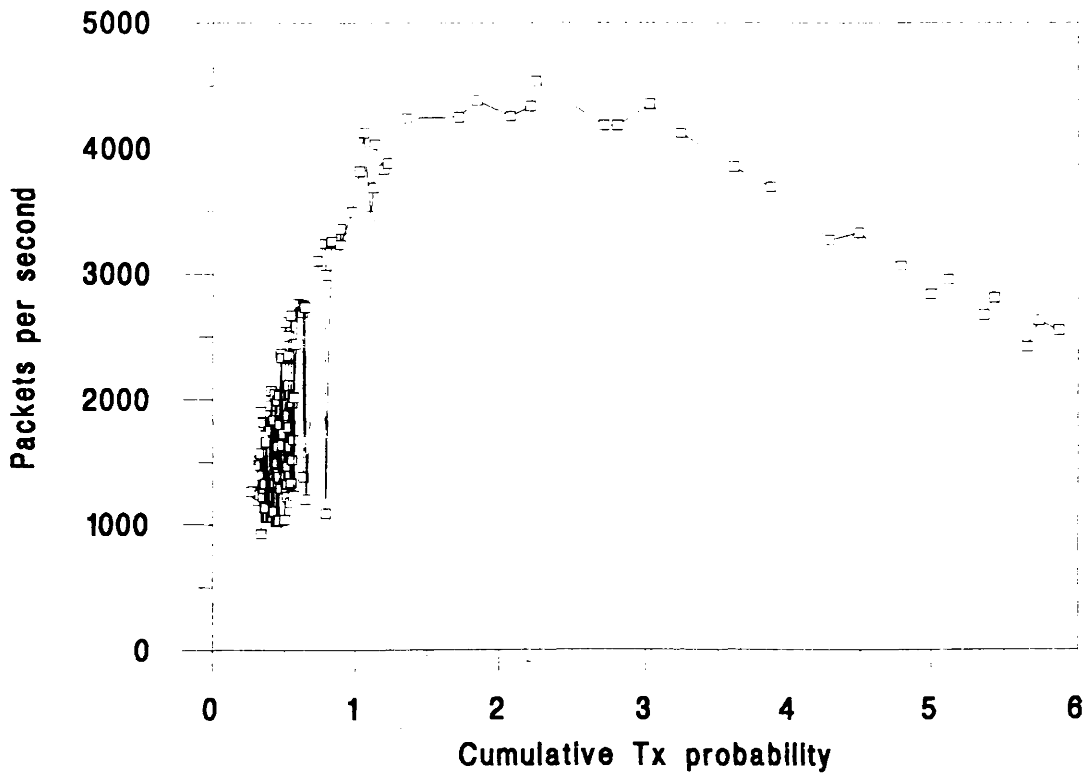

FIGURE 31. Graph of packets per second against cumulative transmission probability for the Ethernet simulation.

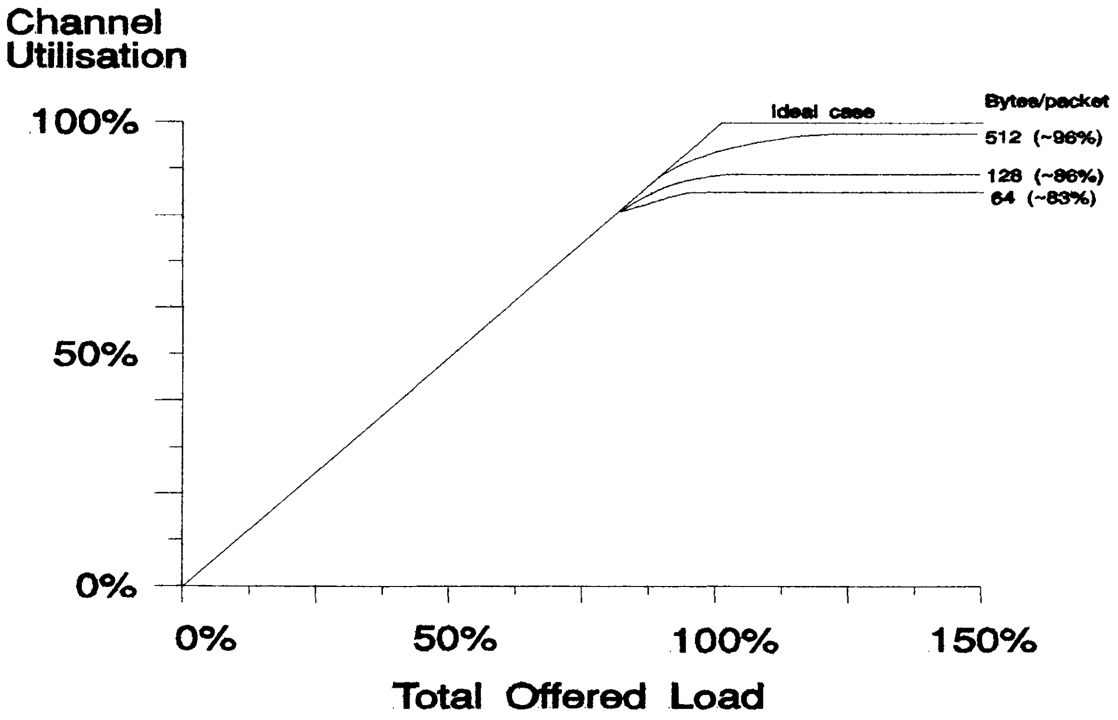

FIGURE 32. Shoch & Hupp's measurement of the experimental Ethernet under high load.

Figure 31 shows the number of packets delivered per second plotted against the cumulative transmission probability. It can be seen that the packet throughput drops off at a cumulative transmission probability of around 3, which represents an per-station probability of around 0.5 or a 50% load on the network.

In contrast, Figure 32 shows the results measured by Shoch & Hupp on an experimental Ethernet and presented in their 1979 paper[34]. There is an obvious difference between the two sets of results. The Petri net model shows channel utilisation falling off as the load increases beyond 50%, while Shoch and Hupp claim increasing utilisation for a load greater than 80%. This may be due in part to the way in which the transmission probabilities were calculated during the simulation, as the method used did not take account of stations being in the backoff, jamming or waiting states. The probability was calculated simply by dividing the total time spent in the transmitting state (time spent in the ***tx_active*** places) by the total runtime of the simulation. It would have been preferable to subtract from the runtime the total time spent in the backoff, jam and wait states, but these figures were not readily available from the simulator output. Another difference between the simulated Ethernet and the real network is that the simulation does not modify the mean backoff delay to match the number of collisions detected as is the case with the real network. However, in spite of these discrepancies the simulator can provide a useful measure of overall Ethernet system performance.

Chapter 7. Conclusions

The overall aim of this thesis was to develop and evaluate a technique of simulation using timed Petri nets. In order to evaluate the results produced it is helpful to consider separately various aspects of the work which was undertaken.

The Petri net model

The aim in designing the present Petri net model was to extend the basic Place/Transition class of Petri net to include an element of time and also to include some other extensions which would make the model more suited to simulation exercises. From the results of the three simulations presented earlier, it is clear that the resulting net model performs adequately in all three cases. Leaving aside the question of timed-place versus timed-transition models for the moment, it was felt that the other extensions succeeded in increasing the simulating ability of the model without significantly increasing the visual complexity of the net diagrams. In particular, the omega-weighted arcs and the multi-weighted arcs proved especially useful in modelling the queueing elements of the simulated systems. As these arcs appear in the diagrams as relatively simple variations on the normal arc, their functions are easily understood in the context of the net diagram.

The random-input and random-delay places which were introduced in order to allow the statistical events also fitted well into the net model. Once again these simply appear as variations on the normal place and their functions are easily comprehensible in the diagrams.

Use of the timed-place model

The timed-place model proved to be an extremely viable alternative to the timed-transition type. The degree of equivalence of the two models was highlighted by the relative simplicity of converting the timed-transition model of the Ethernet system into the timed-place scheme. However in the case of the interruptible transmission interval in the Ethernet system, the timed-place scheme proved less successful. The solution to this problem, although it worked satisfactorily, was rather clumsy and it would appear that there is no direct method of providing such a facility using the timed-place model. This, however, remains a topic for further research.

The timed Petri net simulation program as a performance evaluation tool

The results provided by the simulation program were immediately useful in all three simulation exercises. In the case of the Cart-computer system in Chapter 4, the objective was simply to measure the overall run-time of the system and to measure the time between the occurrences of certain events. This was easily accomplished from the tables of results produced by the simulation program. The decision as to which particular net parameters to measure depended to a large extent on the work done by Torn[12] and it was felt that the usefulness of this seat of measurements in this instance justified the choice.

In the cases of the single-terminal queueing model and the Ethernet system, the simulation program proved itself capable of simulating the steady-state of the systems, and the resulting graphs provided useful estimates of the performances of these systems. In particular, the Ethernet simulation may be used to estimate the performance limits of various real-world networks by varying the parameters of the simulation such as the number of stations, the network length or the size of packet transmitted.

Suitability of the object-oriented model in implementing the simulation program

Given the desire to test the applicability of an object-oriented language to writing a Petri net simulator, the C++ programming language proved to be a good choice, especially as the current program was derived from an older version written in C. The object oriented paradigm was shown to be appropriate for the manipulation of Petri nets within the program.

Comparison with other simulation methods

In terms of its operation and the results produced, the Petri net simulation method described in this thesis is similar to other discrete-event simulation methods, such as those based on special-purpose languages, e.g. SIMULA and SIMSCRIPT. The main difference is that the Petri net method allows the translation from the real-world system to the simulated model to be done graphically. Once the appropriate level of abstraction has been chosen, the net model can be constructed in a fashion analogous to the design of a logic circuit. Indeed one avenue of development for the present simulator would be to add an input facility, whereby the net could be drawn graphically and the translation to the internal data structures of the simulator performed automatically. This would remove the need to translate the net diagram into the textual description which is currently required.

It has been shown that the simulation program described in this thesis, along with its associated class of timed Petri nets has proved capable of simulating systems with a variety of characteristics. Using the simulator it was possible to perform simple time-interval measurements on a system, model the steady-state behaviour of a queueing system and perform simulations using stochastic net techniques. This generality, along with the usefulness of the results obtained, show that the simulator represents a useful general-purpose performance evaluation tool.

References

1 Petri, C.A. Kommunikation mit Automaten. Translation: C.F. Greene, Supplement 1 to Tech Report RADC-TR-65-337, Vol 1, Rome Air Development Center, Griffiss Air Force Base, N.Y., 1965.

2 Peterson, J. L. Petri Net Theory and the Modelling of systems. Prentice-Hall, Englewood Hall, N.J. 1981

3 Reisig, W. Petri Nets, an Introduction. Berlin, Springer Verlag, 1985.

4 Ramchandani, C. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. PhD Thesis, M.I.T. September 1973.

5 Sifakis, J. Performance Evaluation of Systems using nets. Lecture Notes in Computing Science. Springer Verlag. Vol 84. Advances in Petri Nets 1985, Ed. Rozenberg. pp 307-319.

6 Zuberek, W.M. Timed Petri Nets and Preliminary Performance Evaluation. 7th Annual Symposium on Computer Architecture, 1980, pp 88-96.

7 Razouk, R.R. and Phelps, C.V. Performance Analysis using Timed Petri Nets. Protocol Specification, Testing and Verification, Vol IV. Ed. Yemini, Strom and Yemini. Elsevier Science Publishers B.V. IFIP, 1985. pp 561-576.

8 Holliday, M.A. and Vernon, M.K. A Generalised Timed Petri Net Model for Performance Analysis. Proceedings of the International Workshop on Timed Petri Nets, Turin. I.E.E.E. Computer Society Press, September 1985. pp 181-190.

9 Pagnoni, A. Stochastic Nets and Performance Evaluation. Lecture Notes in Computing Science. Springer Verlag. Vol 254. Advances in Petri Nets 1986 Part I, Ed. W. Brauer, W. Reisig and G. Rozenberg. pp 460-478.

10 Ajmone Massan, M. and Chiola, G. On Petri Nets with Deterministic and Exponentially distributed Firing Times. Lecture Notes in Computing Science. Springer Verlag. Vol 266. Advances in Petri Nets 1987, pp 132-145.

11 Torn, A.A. Simulation Graphs: A General Tool for Modelling Simulation Designs.
Simulation Vol 37. No. 6. 1981. pp 187-194.

12 Torn, A.A. Simulation Nets, a Simulation, Modelling and Validation Tool.
Simulation Vol 45. No. 2. 1985. pp 71-75.

13 Alanch, P. Benzabour, K. Dolle, F. Gillet, P. Rodrigues, P and Valette, R. PSI: a Petri Net based Simulator for Flexible Manufacturing Systems.
Lecture Notes in Computing Science. Springer Verlag. Vol 188. Advances in Petri Nets 1984, Ed. Rozenberg. pp 1-14.

14 Nelson, R.A. Haibt, L.M. and Sheridan, P.B. Casting Petri Nets into Programs.
I.E.E.E. Transactions on Software Engineering, Vol. SE-9, No. 5, September 1983.

15 Schiffner, G. and Godberson, H. Function Nets: A Comfortable Tool for Simulating Database System Architectures.
Simulation Vol. 46, No. 5, 1986 pp 201-210.

16 Bauman, R. and Turano, T.A. Production based Language Simulation of Petri Nets.
Simulation Vol. 47 No. 5, 1986, pp 191-198.

17 Zuberek, W.M. M-Timed Petri nets, Priorities, Preemtions and Performance Evaluation of Systems.
Lecture Notes in Computing Science. Springer Verlag. Vol 222. Advances in Petri Nets 1985, Ed. Rozenberg. pp 478-498.

18 Petri, C.A. "Forgotten" Topics of Net Theory.
Lecture Notes in Computing Science. Springer Verlag. Vol 254. Part II. Advances in Petri Nets 1986, Ed. W. Brauer, W. Reisig and G. Rozenberg. pp 500-514.

19 Chin, T.M. and Willsky, A.S. Stochastic Petri net Modelling of Wave Sequences in Cardiac Arrhythmias.
Computers in Biomedical Research, Vol 22, (1989) pp 136-159.

20 Stroustrup, B. The C++ Programming Language. Addison Wesley. Reading, Massachusetts. 1986.

21 Scher, A.M. and Young, A.C., Frequency Analysis of the Electrocardiogram. Circulation Research,8, pp. 344-6, March 1960.

22 Watts, M.P. and Shoat, D.B. Trends in Electrocardiograph Design. Journal of the

Institution of Electronic and Radio Engineers, Vol 57, No. 4, pp 140-150, July/August 1987.

23 Macfarlane, P.W. (1969a). Computer Studies in Electrocardiography. Ph.D. Thesis, University of Glasgow, October, 1969.

24 Macfarlane, P.W. (1969b) A modified Axial Lead System for Orthogonal Lead Electrocardiography.
Cardio. Res. 3, 510.

25 Taylor, P. (1974) Computer Analysis of Cardiac Arrhythmias. Ph.D Thesis, University of Glasgow, March 1974.

26 Watts, M.P. and Macfarlane, P.W. 3-lead electrocardiogram transmission over Post Office telephone lines.
Medical and Biological Engineering and Computing, Vol 15, May 1977, pp 311-318

27 Information processing systems - Open System Interconnection - Basic Reference Model. International Standard ISO/DIS/7498, 1983.

28 Tanenbaum, Andrew S. (1981), Computer Networks, Prentice-Hall software series. pp 189-192

29 DECnet Digital Network Architecture: Digital Data Communications Message Protocol DDCMP Specification Version 4.0
1-March-1978
Digital Equipment Corporation, Maynard, Massachusetts 01754.

30 Konheim A.G. & Chu W.
On the Analysis and Modeling of a Class of Computer Communication Systems
IEEE Transactions on Communications
Vol COM-20, No. 3, June 1972

31 L. Kleinrock and F.A. Tobagi, "Packet Switching in Radio Channels: Part 1 - Carrier Sense Multiple Access Modes and their Throughput-Delay Characteristics," IEEE Transactions on Communications COM-23(12) pp. 1400-1416 (December 1975)

32 R.M. Metcalfe and D.R. Boggs, Ethernet: Distributed Packet Switching for Local Computer Networks, Communications of the ACM, Vol 19, No. 7, July 1976

33 Xerox-Intel-Digital Equipment Corporation, The Ethernet - A Local Area Network - Data Link Layer and Physical Layer Specifications V1.0

34 J.F. Shoch and J.A. Hupp, Measured Performance of an Ethernet Local Network. CACM, 1980, Vol 23 No. 12, Dec 1980, pp 711 - 721.

35 E. Gressier, A Stochastic Petri Net Model for Ethernet, Proc. International Workshop on Timed Petri Nets, IEEE, Torino, Italy (July 1985)

36 M. Ajmone Marsan, G. Chiola and A. Fumagalli, An Accurate Performance Model of CSMA/CD Bus Lan, Lecture Notes in Computing Science. Springer Verlag. Vol 266. Advances in Petri Nets 1987, Ed. Rozenberg. pp 146-161.

Appendix 1. Net specification file for the terminal-computer system.

```
;
; Spec file for single terminal simulation.
;
net Single_terminal_simulation
    {
    run_time 10
    }

random_place terminal
    {
    type random
    capacity 100
    marking 0
    delay 0
    distribution 0.5 10.0
    }

transition send_unit
    {
    input terminal omega

    output channel omega
    }

place channel
    {
    type normal
    capacity 500
    marking 0
    delay 0
    }

place cpu_idle
    {
    type normal
    capacity 2
    marking 1
    delay 0
    }

transition service_unit
    {
    input channel normal 1
    input cpu_idle normal 1

    output cpu_idle normal 1
    output units_serviced normal 1
    }

place units_serviced
    {
    type normal
    capacity 1000
    marking 0
    delay 0
    }
```

Appendix 2. Net simulator output for the Multiplexed system.

Timing values in the following tables are in milliseconds.

```
Netstat V3.0

                        Net Simulation Statistics
                        -------------------------

Simulation name: mds2   38 Places and  41 Transitions.

Simulation description:
Multiplexed system with no additional queueing delays
Initial marking:
  0  15  15  20  15  15  15  15   0   0   0   0   0   0   0   0   0   1
0   0   0   0   0   0   1   1   0   1   0   0   0   0   0   0   0   0
1   0
Final marking:
  0   0   1   1   1   1   1   1   1   0   0   0   0   0   0   0   0   0
0  15   0   0   0   0   1   1   0   1   0   0   0 110   0   0   0   0
1   1
Terminal marking:
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255
Time limit:  40000

Simulation terminated at global time 24898  Simulation time 24633
Termination code -1

Simulation runtime:  5:18:25

Place Statistics
----------------


 Marking data
 ------------
 Number      Name          Type   Min   Max      Mean Variance  % Zero
 ---------------------------------------------------------------------
    1        Queue_active n      0    10    9.4991   4.7010   4.8757
    2          Queue_tail n      0    15    0.0818   0.7443  98.5100
    3                 Qp2 n      0    15    5.6427  17.2443   0.2169
    4                 Qp3 n      0    20    6.2895  19.6278   0.2169
    5                 Qp4 n      0    20    6.9208  21.2639   0.2169
    6                 Qp5 n      0    20    7.5983  28.1245   0.2169
    7                 Qp6 n      0    20    8.2290  28.0888   0.2169
    8                 Qp7 n      0    20    8.8450  27.4305   0.2169
    9          Queue_head n      0    20    9.4694  25.8165   0.2169
   10                 Qe1 n      0     2    0.0022   0.0043  99.8916
   11                 Qe2 n      0     2    0.0033   0.0054  99.7831
   12                 Qe3 n      0     2    0.0033   0.0054  99.7831
   13                 Qe4 n      0     2    0.0033   0.0054  99.7831
   14                 Qe5 n      0     2    0.0033   0.0054  99.7831
   15                 Qe6 n      0     2    0.0033   0.0054  99.7831
```

```
16                 Qe7 n     0     2   0.0033    0.0054  99.7831
17          Re-queueing n     0     1   0.0008    0.0008  99.9237
18        Start_process n     0     1   0.0026    0.0026  99.7389
19     Datagram_started n     0     1   0.0009    0.0009  99.9116
20             Trashcan n     0    15  10.1608   21.7593   0.0643
21     Queue_head_empty n     0     1   0.3087    0.2134  69.1313
22              Polling n     0     1   0.0009    0.0009  99.9116
23      Packets_waiting n     0     5   1.9306    2.0735  22.8644
24     Packet_available n     0     1   0.0044    0.0044  99.5582
25          Packet_done n     0     1   0.0413    0.0396  95.8673
26              Tx_idle n     1     1   1.0000    0.0000   0.0000
27       Sending_packet n     0     1   0.0044    0.0044  99.5582
28              Rx_idle n     1     1   1.0000    0.0000   0.0000
29      Packet_received n     0     1   0.0044    0.0044  99.5582
30        Receiving_ACK n     0     1   0.0044    0.0044  99.5582
31    Waiting_for_reply n     0     1   0.6715    0.2206  32.8487
32         Global_count n     0   110  55.5319 1026.4484    0.9639
33       Reply_received n     0     5   1.9306    2.0735  22.8644
34        Datagram_sent n     0     1   0.0009    0.0009  99.9116
35         Decode_check n     0     1   0.0003    0.0003  99.9719
36     Sending_confirm. n     0     1   0.0003    0.0003  99.9719
37        Datagram_done n     0     1   0.0042    0.0042  99.5783
38     Datagram_enabled n     0     1   0.9833    0.0164   1.6707
```

```
Timing data
-----------


                         Time spent
Number          Name   Min   Max       Mean     Nc     Fc    Lc
-------------------------------------------------------------------

   1     Queue_active   14  1187   187.9762    126      0 23455
   2       Queue_tail   13    14     6.8704     54      0 23468
   3              Qp2   12  1188   451.7273     55      0 23468
   4              Qp3   10  1188   451.7273     55      0 23466
   5              Qp4    8  1188   451.7273     55      0 23464
   6              Qp5    6  1188   451.7273     55      0 23462
   7              Qp6    4  1188   451.7273     55      0 23461
   8              Qp7    2  1188   451.7273     55      0 23459
   9       Queue_head   13  1188   365.3676     68      0 23457
  10              Qe1    1     1     0.5000     54      0 23468
  11              Qe2    2     2     0.6667     81      0 23467
  12              Qe3    2     2     0.6667     81      0 23465
  13              Qe4    2     2     0.6667     81      0 23463
  14              Qe5    2     2     0.6667     81      0 23461
  15              Qe6    2     2     0.6667     81      0 23460
  16              Qe7    2     2     0.6667     81      0 23458
  17      Re-queueing    1     1     0.5000     38      2 23454
  18    Start_process    1     3     1.2037     54      0 23456
  19 Datagram_started    1     1     0.5000     44      1 23456
  20         Trashcan    0     0     0.0000     15      2 20006
  21 Queue_head_empty 1098  1098   549.0000     14  15290 24546
  22          Polling    1     1     0.5000     44      7 23461
  23  Packets_waiting  873   873   145.5000    132      7 24330
  24 Packet_available    1     1     0.5000    220     72 24395
  25      Packet_done    1   167     4.6561    221      0 24546
  26          Tx_idle    0     0     0.0000      1      0     0
  27   Sending_packet    1     1     0.5000    220    158 24481
  28          Rx_idle    0     0     0.0000      1      0     0
  29  Packet_received    1     1     0.5000    220    223 24546
  30    Receiving_ACK    1     1     0.5000    220    224 24546
  31 Waiting_for_reply 152   152    76.0000    220     73 24546
  32     Global_count    0     0     0.0000    110    224 24546
  33   Reply_received  873   873   145.5000    132    224 24546
  34    Datagram_sent    1     1     0.5000     44   1092 24546
  35     Decode_check    1     1     0.5000     14  16380 24547
  36 Sending_confirm.    1     1     0.5000     14  16467 24633
  37    Datagram_done    1    69     2.3333     45      0 24633
  38 Datagram_enabled    1  1174   461.9434     53      0 23468
```

```
Transition Statistics
---------------------

                              Firings                T.B.F
Number                  Name Total  First  Last   Min   Max       Mean
-----------------------------------------------------------------------
    1                   Qt14     4      0 20009     0 20008  6669.6665
    2                   Qt15    37      0 21102     0  5807   586.1667
    3                   Qt16    44      0 22285     0  5807   518.2558
    4                   Qt17    46      0 23454     0  5804   521.2000
    5                   Qt18    34      0 23454     0  4624   710.7273
    6                   Qt19    34      0 23454     0  3446   710.7273
    7                   Qt20    35      0 23454     0  2355   689.8235
    8                   Qt21    26      2 23454     0  4631   938.0800
    9                    Qt1    27      0 23468     0  1181   902.6154
   10                    Qt2    27      0 23466     0  1181   902.5385
   11                    Qt3    27      0 23464     0  1181   902.4615
   12                    Qt4    27      0 23462     0  1181   902.3846
   13                    Qt5    27      0 23461     0  1180   902.3461
   14                    Qt6    27      0 23459     0  1181   902.2692
   15                    Qt7    27      0 23457     0  1181   902.1923
   16               Re-queue    19      3 23454     0  5808  1302.8334
   17        Replenish_queue     7  15290 23456  1179  2271  1361.0000
   18                    Qt8    27      0 23467     0  1181   902.5769
   19                    Qt9    27      0 23465     0  1181   902.5000
   20                   Qt10    27      0 23463     0  1181   902.4231
   21                   Qt11    27      0 23461     0  1181   902.3461
   22                   Qt12    27      0 23460     0  1180   902.3077
   23                   Qt13    27      0 23458     0  1181   902.2308
   24           Get_datagram    22      1 23455  1092  1179  1116.8572
   25          Shuffle_queue     4  23454 23454     0     0     0.0000
   26               De-queue    27      0 23456     0  1181   902.1539
   27        Select_re-queue    19      2 23454     0  5808  1302.8889
   28              Set_empty     7  15290 23456  1179  2271  1361.0000
   29               Set_full    15      2 20006  1092  5808  1428.8572
   30         Start_datagram    22      7 23461  1092  1179  1116.8572
   31        Get_next_packet   110      8 24330   217   311   223.1376
   32            Send_packet   110     73 24395   217   311   223.1376
   33         Receive_packet   110    159 24481   217   311   223.1376
   34          Receive_reply   110    224 24546   217   311   223.1376
   35               Send_ACK   110    224 24546   217   311   223.1376
   36     Terminate_datagram    22   1092 24546  1092  1179  1116.8572
   37           Start_decode     7  16380 24546  1179  2271  1361.0000
   38               Continue    15   1092 21096  1092  5808  1428.8572
   39      Send_confirmation     7  16381 24547  1179  2271  1361.0000
   40            End_message     7  16467 24633  1179  2271  1361.0000
   41        Enable_datagram    27      0 23468     0  1181   902.6154
```

Appendix 3. Net Simulator Output for the Non-multiplexed System

Timing values in the following tables are in milliseconds.

```
Netstat V3.0

                        Net Simulation Statistics
                        -------------------------

Simulation name: fcfs   31 Places and  32 Transitions.

Simulation description:
First_come_first_served_system_with_no_queueing_delays
Initial marking:
  0  15  15  20  15  15  15  15   0   0   0   0   0   0   0   2   0   0
0   1   1   0   1   0   0   0   0   0   0   0   0
Final marking:
 10   0   0   0   0   0   0   0   0 199  28   0   0   0   0   0   0   0
0   1   1   0   1   0   0   0 110   0   0   0   0
Terminal marking:
255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255
255 255 255 255 255 255 255 255 255 255 255 255 255
Time limit:  50000

Simulation terminated at global time 12103  Simulation time 11924
Termination code  1

Simulation runtime:  1:26: 7


Place Statistics
----------------

 Marking data
 ------------
 Number      Name          Type   Min   Max     Mean Variance  % Zero
 -----------------------------------------------------------------
    1        Queue_active n      0    10   9.9989   0.0090   0.0083
    2          Queue_tail n      0    15   0.0161   0.2414  99.8926
    3                 Qp2 n      0    15   2.0510  26.5603  86.3268
    4                 Qp3 n      0    20   4.0896  44.6969  72.7611
    5                 Qp4 n      0    20   6.7998  69.3478  59.1953
    6                 Qp5 n      0    20   8.8347  68.0565  45.6295
    7                 Qp6 n      0    20  11.5387  53.5252  27.6024
    8                 Qp7 n      0    20  13.5736  32.9462  14.0367
    9          Queue_head n      0    20   0.0091   0.1445  99.9422
   10                 Qe1 n      0   199  35.3320 1756.8444   0.0991
   11                 Qe2 n      0    33   0.0924   1.7202  99.1490
   12                 Qe3 n      0    17   0.0340   0.3202  99.4382
   13                 Qe4 n      0     9   0.0105   0.0507  99.6861
   14                 Qe5 n      0     5   0.0041   0.0109  99.8017
   15                 Qe6 n      0     3   0.0024   0.0044  99.8513
```

```
16                Qe7 n       0     2   0.0020    0.0033  99.8678
17            Polling n       0    20   0.0091    0.1445  99.9422
18    Packets_waiting n       0    20   7.3257   22.8590   8.0387
19   Packet_available n       0     1   0.0091    0.0090  99.0912
20        Packet_done n       0     1   0.0276    0.0268  97.2406
21            Tx_idle n       1     1   1.0000    0.0000   0.0000
22     Sending_packet n       0     1   0.0091    0.0090  99.0912
23            Rx_idle n       1     1   1.0000    0.0000   0.0000
24    Packet_received n       0     1   0.0091    0.0090  99.0912
25      Receiving_ACK n       0     1   0.0091    0.0090  99.0912
26  Waiting_for_reply n       0     1   0.8815    0.1044  11.8473
27       Global_count n       0   110  54.7257 1016.6667    0.9666
28     Reply_received n       0    20   7.3257   22.8590   8.0387
29      Datagram_sent n       0     1   0.0006    0.0006  99.9422
30       Decode_check n       0     1   0.0006    0.0006  99.9422
31    Sending_confirm n       0     1   0.0006    0.0006  99.9422
```

```
Timing data
-----------


                                   Time spent
Number                   Name    Min    Max       Mean      Nc      Fc     Lc
------------------------------------------------------------------------------
    1            Queue_active      0      0     0.0000       2       0      1
    2              Queue_tail     13     13     6.5000       2       0     11
    3                     Qp2     11   1644   413.7500       4       0   1636
    4                     Qp3      9   1644   549.5000       6       0   3261
    5                     Qp4      7   1644   617.3750       8       0   4887
    6                     Qp5      5   1644   658.1000      10       0   6512
    7                     Qp6      3   2184   730.2500      12       0   8672
    8                     Qp7      1   2184   743.2143      14       0  10297
    9              Queue_head      1      1     0.5000      14       0  10298
   10                     Qe1      0      0     0.0000     101      10  11924
   11                     Qe2      2     32     0.9364     110       8  11924
   12                     Qe3      2     32     0.8947      76       7  11924
   13                     Qe4      2     16     0.8261      46       5  11924
   14                     Qe5      2      8     0.7500      32       3  11924
   15                     Qe6      2      4     0.6923      26       1  11924
   16                     Qe7      2      2     0.6667      24       0  11924
   17                 Polling      1      1     0.5000      14       7  10304
   18         Packets_waiting   1513   2053    95.1367     117       7  11803
   19        Packet_available      1      1     0.5000     220      17  11813
   20             Packet_done      1     27     1.5113     221       0  11909
   21                 Tx_idle      0      0     0.0000       1       0      0
   22          Sending_packet      1      1     0.5000     220     103  11899
   23                 Rx_idle      0      0     0.0000       1       0      0
   24         Packet_received      1      1     0.5000     220     113  11909
   25           Receiving_ACK      1      1     0.5000     220     114  11909
   26       Waiting_for_reply     97     97    48.5000     220      18  11909
   27            Global_count      0      0     0.0000     110     114  11909
   28          Reply_received   1513   2053    95.1367     117     114  11909
   29           Datagram_sent      1      1     0.5000      14    1612  11910
   30            Decode_check      1      1     0.5000      14    1618  11916
   31          Sending_confirm     1      1     0.5000      14    1627  11924
```

```
Transition Statistics
---------------------
                              Firings                  T.B.F
Number                   Name Total  First  Last    Min   Max      Mean
---------------------------------------------------------------------------
   1                     Qt14     2      0     1      1     1    1.0000
   2                     Qt15     2      0     1      1     1    1.0000
   3                     Qt16     2      0     1      1     1    1.0000
   4                     Qt17     1      0     0      0     0    0.0000
   5                     Qt18     1      0     0      0     0    0.0000
   6                     Qt19     1      0     0      0     0    0.0000
   7                     Qt20     1      0     0      0     0    0.0000
   8                     Qt21     0     -1    -1      0     0    0.0000
   9                      Qt1     1     11    11      0     0    0.0000
  10                      Qt2     2      9  1636   1627  1627 1627.0000
  11                      Qt3     3      7  3261   1627  1627 1627.0000
  12                      Qt4     4      6  4887   1627  1627 1627.0000
  13                      Qt5     5      4  6512   1627  1627 1627.0000
  14                      Qt6     6      2  8672   1627  2162 1734.0000
  15                      Qt7     7      0 10297   1627  2162 1716.1666
  16                      Qt8   100     10 11924      0  2155  120.3434
  17                      Qt9    65      8 11924      0  2159  186.1875
  18                     Qt10    34      7 11924      0  2160  361.1212
  19                     Qt11    19      5 11924      0  2161  662.1667
  20                     Qt12    12      3 11924      0  2162 1083.7273
  21                     Qt13     9      1 11924      0  2162 1490.3750
  22             Get_datagram     7      1 10298   1627  2162 1716.1666
  23            Start_process     7   1627 11924   1627  2162 1716.1666
  24           Start_datagram     7      7 10304   1627  2162 1716.1666
  25          Get_next_packet   110      8 11803    107   129  108.2110
  26              Send_packet   110     18 11813    107   129  108.2110
  27           Receive_packet   110    104 11899    107   129  108.2110
  28            Receive_reply   110    114 11909    107   129  108.2110
  29                 Send_ACK   110    114 11909    107   129  108.2110
  30       Terminate_datagram     7   1612 11909   1627  2162 1716.1666
  31             Start_decode     7   1613 11910   1627  2162 1716.1666
  32             Send_confirm     7   1619 11916   1627  2162 1716.1666
```