SERVICE AND METASTABILITY PERFORMANCE OF ARBITERS

by

Lindsay Kleeman

(B.E. Hons.I, B.Math. Hons.I)

Supervisor : Professor Antonio Cantoni

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

at

The University of Newcastle

New South Wales, 2308

Australia

August 1986

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Signed : L. Kleeman

ACKNOWLEDGEMENTS

I wish to sincerely thank my supervisor Professor Tony Cantoni for his guidance, fruitful ideas, motivation and uncompromising insistence on a rigorous approach to research. His vigour in uncovering hidden assumptions and exploring fundamental issues of importance has been greatly appreciated. My supervisor's wife, Stephnie, and children, Michael and Simon, are warmly thanked for welcoming me into their household during our stay in Pisa, Italy.

My grateful appreciation is also given to:

- the Australian Computer Research Board for the financial support of a Postgraduate Scholarship and research grants relevant to the thesis;
- all postgraduate students and academic staff in the Department for creating a stimulating, friendly and scholastic environment;
- the technical staff of the Department, in particular Ron Goodhew, Fred Martinez, Peter McLaughlan, Fred Huang and Lloyd Jennings for their helpful and friendly assistance;
- the secretarial staff of the Department for their help in preparing papers and organisational matters;
- Ildiko de Souza for helping to type this thesis and Wanda Lis for preparing diagrams within the thesis;
- the Department for generously financing my travel expenses to the Performance '86 and Sigmetrics 1986 conference in the USA;
- the Istituto di Elettronica e Telecomunicazioni, Universita di Pisa for their assistance during my visit in 1984.

Finally, I would like to thank Louise, for her love, support and understanding during the whole course of my Ph.D.

ABSTRACT

The thesis reports the results of a study of issues related to the performance of arbiters. Arbiters are digital circuits which are inherently susceptible to metastable behaviour because of the asynchronous nature of their inputs. The thesis examines a number of fundamental issues concerning metastability in digital circuits and establishes their relevance to the performance of arbiters and synchronisers. Furthermore, the thesis presents new models and new analytical tools for evaluating the service and utilisation performance of arbiters.

The question of the unavoidability of metastability is examined in the thesis and previous results are extended to enable direct application to circuits with realistic waveforms. It is also shown in the thesis that redundancy and masking techniques are ineffectual in improving the performance of synchronisers with respect to metastability.

A general model for analysing the metastable performance of digital systems, called the aperture model, is developed in the thesis. One application of the aperture model is to the evaluation of the performance of various schemes designed for improving the metastable performance of synchronisers. The development of the aperture model is an important step towards the analysis of the rate of occurrence of failure due to metastability in arbiters.

The thesis identifies a new class of arbiters referred to as batched arbiters. This batching mechanism can be combined with primary service disciplines, such as fixed priority and round robin to generate new service disciplines. Models are developed for these arbiters which include many practical circuit parameters not previously considered. The modelling has been carried out for both asynchronous and clocked arbiters.

New analytical tools for the analysis of the service and utilisation performance of arbiters are developed in the thesis. The techniques are based on imbedded Markov chains. Arbitrary service time distributions can be handled by the analysis techniques developed, and the performance under light and heavy request loadings is derived. The analysis techniques are applied to both asynchronous and clocked arbiters, and a variety of service disciplines.

The aperture model for metastability of digital circuits is combined with the analysis techniques for the arbiter models adopted to derive the metastable failure rate performance of various arbiters. Furthermore, the impact of certain internal arbiter functions, such as the request resetting mechanism in clocked arbiters is evaluated.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION, HISTORY AND MOTIVATION

This thesis is primarily concerned with the modelling and analysis of the performance of arbiters. In a digital system, an arbiter is a circuit that processes a set of requests for a resource to generate one and only one acknowledge, which grants access to the resource. The characteristics of arbiters are key factors in determining the overall performance of digital systems due to the central role of arbiters in primitive resource allocation. Two aspects of the performance of arbiters are examined in detail in this thesis. From the traditional viewpoint of service and utilisation performance, the thesis develops new models and analysis techniques. From a reliability performance viewpoint not often considered, the thesis develops new models and analysis techniques for evaluating the effects of metastability, a phenomenon inherent to arbiters due to the asynchronous nature of request inputs.

Considerable attention has been given in the literature to the problem of designing efficient arbiters [B.1, B.4, C.1, C.5, F.1, I.2, I.3, K.3, K.13, L.4, M.2, M.9, P.1, P.4, R.1, S.1, T.1, T.3]. However, considerably less attention has been given to the evaluation of the performance of arbiters. Most of the work reported in the literature describes Monte-Carlo simulation approaches for evaluating the service and utilisation performance of arbiters [B.1, H.2, K.13, M.5, S.1], and a small number of papers develop analytical tools [M.5, M.9]. A factor important in determining arbiter performance is the service discipline. The service discipline defines the order in which pending requests are serviced. A range of disciplines are considered in the literature. These

include fixed priority [B.1, B.3, C.17, H.2, I.2, K.13, M.9], round robin [B.1, B.4, C.5, C.17, K.13, M.2], first come first served [B.1, K.14, S.1] and dynamic priority disciplines [B.1, F.1, L.4, T.1]. Furthemore, an important new class of arbiters, termed batched arbiters, is identified and a suitable model is developed in the thesis. A batched arbiter employs a lock out mechanism on arriving requests when pending requests exist, to prevent persistent high priority requesters hogging the resource [C.2, C.5, T.1, T.2]. In the thesis, the batching mechanism is extended to generalise existing disciplines.

Based on an ideal model for a FCFS arbiter and exponential service and re-request time distributions, standard queueing theory results can be applied to analyse the service and utilisation performance [G.2, K.12]. Except for multi-level FCFS [M.9], the performance of other disciplines has been evaluated using Monte-Carlo simulation [B.1, H.2, K.13, M.5]. An incomplete and somewhat dubious analysis of a fixed priority arbiter is presented in [H.2]. The thesis develops an analysis technique applicable to a wide range of batched and non batched disciplines. The analysis techniques developed are based on imbedded Markov chains [G.2, K.12], which do not seem to have been applied previously, and yet prove to be valuable for the analysis of practical arbiter systems. Furthemore, the service time distributions in this analysis are not limited to the exponential distribution. In fact, the service distributions may be arbitrary, including deterministic service times. The relative merits of different analysis approaches and the applicability of the underlying assumptions are discussed in the thesis.

Although the existence of practical arbiter circuit parameters, such as propagation delays, is recognised in the literature [B.1, C.1, G.4, K.13, S.1, T.3], the analytical models employed in the analyses of the

performance of arbiters do not incorporate these non ideal circuit parameters. Based on the examination of a number of practical arbiter circuits, the thesis formulates models which incorporate these non ideal paramaters. Practical arbiters may be clocked [B.1, C.2, C.17, F.1, G.4, H.2, I.2, K.13, L.4, M.9] or asynchronous [C.1, C.5, C.12, C.13, M.2, P.4, S.1, T.3]. Analytical techniques for evaluating the performance of clocked arbiters do not appear in the literature, although some Monte-Carlo simulations with realistic modelling exist [G.4, H.2]. The thesis develops realistic models for clocked arbiters. The extended models for both asynchronous and clocked arbiters mentioned above are amenable to the analysis technique of imbedded Markov chains developed in the thesis.

Metastability in a digital system is a malfunction corresponding to the circuit state remaining indefinitely between two stable states, that can produce ambiguous state interpretation and other erroneous effects within the circuit. While the existence of problems due to metastability is recognised in some of the literature on arbiters [B.2, C.1, C.5, C.8, C.12, C.13, I.1, K.3, K.13, L.4, M.9, P.1], to the author's knowledge there has been no modelling and analysis of metastability induced failure rate of arbiters. However, the study of the metastability of primitive digital circuits such as flip-flops, Schmitt triggers and inertial delays [C.4, C.7, C.8, C.9, E.1, F.2, H.3, H.4, K.3, L.1, L.2, M.3, M.4, P.3, R.2, U.1, V.1] has received increased attention after confused beginning [C.6]. In fact, integrated circuits manufacturers are becoming more aware of the importance of the problem of metastable behaviour in digital system design and the problem is addressed in recented data books [I.1, L.3]. Much of the recent literature on VLSI recognises the importance of this phenomenon [G.3, M.7].

A number of papers [F.2, H.3, L.1, R.1] on the modelling of metastability in flip-flops, synchronisers and other primitive elements allude to the formulation of a model which is further developed by the author for use in the analysis of more complex circuits containing the primitive elements. In particular, the thesis applies the model to the analysis of the metastability performance of both batched and non batched arbiters. The model developed in the thesis is called the aperture model. In view of the fundamental nature of the metastability problem in digital systems and in particular arbiters, the thesis also contains a broad overview of metastability.

The lack of understanding of metastability [C.6, F.3, M.4, W.3] motivated work directed towards a rigorous examination of the fundamentals of the problem. A significant step towards dispelling any myths, for example, that a perfect metastable-free arbiter with asynchronous request inputs can be physically realised, was made with the work of Hurtado and Marino [H.4, M.3, M.4]. However in the theory developed, the set of functions over which the system inputs can range necessarily contain functions which are unlikely to be found in practical circuits, such as those with discontinuities or unbounded derivatives. It was not clear to what extend the unavoidability was related to the inclusion of these possibly impractical input functions. In the thesis, it is shown that the inclusion of these unrealistic input functions is not necessary in order to establish the unavoidability of metastability. Examples of sets of more realistic input functions are given that necessarily contain input functions that give rise to metastability in digital circuits with a "decision" mechanism.

Once accepted as unavoidable, the design problem is to reduce the probability of metastability to an acceptable level, and a great deal of

attention has been given to this problem [A.1, C.4, C.6, C.7, C.8, C.9, C.14, C.15, C.16, E.1, F.2, F.4, K.3, M.3, M.4, M.9, P.3, R.1, S.6, S.7, V.1]. Although experimental data has been produced [C.7, C.16, R.1, R.2], the quantitative evaluation of the schemes developed has received less attention [H.3, L.5, R.1]. The performance of various schemes suggested in the literature are examined from an analytical viewpoint in the thesis.

Redundancy and masking techniques are well established as a means for improving system reliability when physical component failures and transient noise errors are considered [K.1, K.2, M.6, M.8, W.1, W.2, W.4]. It is shown in the thesis that these same techniques are ineffective for improving the reliability when metastable failure is considered. This result establishes a fundamental difference in nature between metastability and permanent or transient hardware component failure.

1.2 CONTRIBUTIONS OF THE THESIS

The main contributions of the thesis are as follows:

(i) The thesis identifies a new class of arbiters referred to as batched arbiters and develops models for the analysis of arbiters which incorporate the batching mechanism. The thesis also describes a mechanism for generating new arbitration disciplines by combining the batching mechanism with primary disciplines, such as fixed priority and round robin.

(ii) Models for arbiters which account for many practical arbiter circuit parameters not previously considered in the literature are developed. The modelling has been carried out for both asynchronous and clocked arbiters.

(iii) New analytical tools for the analysis of the service and utilisation performance of arbiters are developed. The analysis techniques are based on imbedded Markov chains. Arbitrary service time distributions are permitted by the analysis techniques developed. The analysis enables limiting results under light and heavy request loadings to be derived for performance measures. Futhermore, the techniques are applied to both asynchronous and clocked arbiters and a variety of service disciplines.

(iv) The thesis examines a number of fundamental issues concerning metastability in digital systems and specifically its relevance to the reliability performance of arbiters:

(a) A general model for analysing metastable reliability performance of digital circuits, called the aperture model, is developed.

(b) A quantitative evaluation of schemes designed for improving metastable reliability in synchronisers is presented.

(c) The thesis extends previous results on the unavoidability of metastability in digital systems to enable direct application of the results circuits with practical digital input signals.

(d) It is shown that redundancy and masking techniques are ineffective in improving the metastable reliability of synchronisers.

(v) An important contribution of the thesis is the analysis of the performance of arbiters from the viewpoint of metastability induced failure. The analysis techniques are applied to both asynchronous and clocked arbiters The impact of certain internal arbiter

functions, such as the request resetting mechanism in clocked arbiters, is evaluated.

1.3 OUTLINE OF THE THESIS

The thesis is organised as follows:

In Chapter 2, arbiters and their performance measures are introduced and defined. An overview of the relevant literature is presented in order to put the work presented in the thesis in perspective. This chapter provides a basis and framework for the discussion and analysis of arbiters presented in later chapters. Various service disciplines and the concept of batched disciplines are introduced and practical circuit implementations given.

Metastable behaviour is a problem encountered in many digital circuits with asynchronous inputs. Chapter 3 introduces metastability and presents an overview of developments in the area. Original results on the unavoidability of metastable behaviour are presented. The efficacy of masking and redundancy techniques applied to synchronisers is examined. In addition, the aperture model for analysing metastable behaviour of general digital circuits is developed. Various techniques suggested for improving the metastable reliability of synchronisers are analysed using the aperture model.

In Chapter 4, approaches for analysing the service and utilisation performance arbiters are reviewed and a new technique based on imbedded Markov chain is introduced. The relative merits of the approaches are examined. The arbiter models which form the basis of the analysis are developed in this chapter.

In Chapter 5, the analysis approach of imbedded Markov chains is applied to fixed priority batched arbiters and the transition probability

matrix of the Markov chain is derived. The transition matrix is shown to be irreducible and primitive. Performance measures are derived from the unique limiting probabilities and service and metastable reliability performance measures are considered. Analytical limiting results for the light and heavy request loading are derived. Finally, numerical results obtained from the Markov analysis are presented and compared to Monte-Carlo simulation results.

Chapter 6 has the same structure as Chapter 5. The chapter contains corresponding analysis and results for non batched fixed priority arbiters.

In Chapter 7, clocked fixed priority batched arbiter configurations are examined. Three different resetting strategies for an arbiter design are considered. Models for analysing their performance are developed. As in Chapters 5 and 6, an approach based on imbedded Markov chains is employed to analyse the performance of the three configurations. Finally, numerical results obtained from the Markov analysis are presented and compared to Monte-Carlo simulation results.

Chapter 8 contains a selection of results comparing the performance of various batched and non batched arbitration disciplines which have not been amenable to analytical approaches. The numerical results presented are obtained by Monte-Carlo simulation techniques.

Finally, Chapter 9 contains conclusions of the thesis and directions of future research.

CHAPTER 2

ARBITERS

2.1 INTRODUCTION

The aim of this chapter is to introduce and motivate the study of arbiters conducted in this thesis and also to provide an overview of the relevant literature. Furthermore, it provides a framework and basis for the discussion and analysis presented in subsequent chapters, and introduces the various arbitration disciplines together with terminology, notation, and implementations. Areas of interest in relation to arbiter performance are discussed. In particular, service performance and reliability performance are distinguished and defined.

The organisation of this chapter is as follows: In Section 2.2, an ideal arbiter is defined and input/ouput constraints are presented. Section 2.3 briefly gives examples of applications of arbiters in systems, to provide some practical motivation for a study of arbiters. In Section 2.4, different disciplines employed in arbitration circuits are introduced and example implementations are described. Various classifications are also defined in this section, such as centralised/decentralised, clocked/non clocked, and batched/non batched. In Section 2.5, correctness proving of arbitration circuits is discussed and an example is given of an "incorrect" arbiter to motivate the discussion. After functional correctness of an arbitration circuit has been established, the performance of a model of its behaviour in response to requests can be evaluated. Section 2.6 qualitatively introduces performance measures used in the thesis and discusses their practical meaning. Section 2.7 discusses aspects of the reliability performance of arbiters, dividing

reliability into hardware related failures and metastable behaviour induced failures. The final section summarises the chapter and presents conclusions.

2.2 DEFINITION OF AN IDEAL ARBITER

Before proceeding with the definition of an arbiter, a few terms are introduced: A *requester* is an autonomous source of requests for exclusive access to a shared common *resource*. The shared resource cannot be accessed by multiple requesters simultaneously, but must be allocated to one requester at a time. The allocation of the resource in response to a request can also be referred to as servicing of the request, granting of the resource or acknowledging the request. An important example of a shared resource in a computer system, particularly multiprocessor systems, is a shared bus [B.6, C.11, E.2, H.2, K.14, M.1, M.5, T.3, T.4, W.5]. Obviously, information can only be transferred when only one user drives the bus at a time and consequently bus access sequencing must occur among conflicting users. Additional, more detailed examples can be found in the next section.

In order that mutually exclusive access to the resource be ensured, requests must be arbitrated to determine which request is given access. Other pending requests must then wait. A requester is signalled that it has been given access to the resource by an acknowledge signal [P.4] (also referred to as a grant signal elsewhere [C.17]). The function of an arbiter is to control the acknowledge signals, as indicated in the following definition.

An *arbiter* is a digital circuit that processes a set of requests to generate *one and only one* acknowledge corresponding to an asserted request. The arbiter drops an acknowledge only after the corresponding

request is dropped. Apart from constraints on the arbiter outputs, constraints are imposed on the arbiter inputs, namely requesters [P.4, C.17]. A requester only requests when its acknowledge is not asserted and only drops a request after it receives an acknowledge. Once a request is lodged, the requester is committed to holding the request until it is serviced by the arbiter. Also after dropping its request, a requester must wait until the arbiter has responded by dropping the corresponding acknowledge signal before the requester can proceed to lodge another request. A fully interlocked handshaking protocol [T.3] is established by these constraints.

Figure 2.1 shows an arbiter with k request inputs, denoted Req 1, Req 2, .., Req k, and the corresponding k acknowledge outputs, denoted Ack 1, Ack 2, .., Ack k. Requester h lodges a request by asserting Req h, as shown in Figure 2.2, and the arbiter responds later by asserting Ack h to grant the resource to requester h. When requester h has completed accessing the resource, it drops Req h and waits for Ack h to drop before it can request again. The constraints stated above for the arbiter circuit and requesters are summarised concisely below:

$$\uparrow \text{Req h} \implies \text{Ack h} = 0 \qquad (2.1)$$

$$\uparrow \text{Ack h} \implies \text{Req h} = 1 \qquad (2.2)$$

$$\downarrow \text{Req h} \implies \text{Ack h} = 1 \qquad (2.3)$$

$$\downarrow \text{Ack h} \implies \text{Req h} = 0 \qquad (2.4)$$

$$\text{Ack } h = 1 \implies \text{for all } i,\ i \neq h \quad \text{Ack } i = 0 \qquad (2.5)$$

$$\text{Req } h = 1 \implies \text{there exists } i,\ \text{Ack } i = 1 \qquad (2.6)$$

where $\Longrightarrow$ denotes implication, $\uparrow$ denotes a 0-1 transition and $\downarrow$ a 1-0 transition. The first four constraints (2.1-4) correspond to the arrows in Figure 2.2 labelled from (1) to (4) respectively and have already been discussed. The constraint (2.5) states that only one Ack output is asserted at a time, whilst (2.6) states that if a request is pending then there is an Ack asserted. In a non ideal arbiter a delay occurs between the assertion of a request and assertion of an acknowledge but is ignored here in the ideal case. Note that (2.2), (2.4), (2.5) and (2.6) are output or functional constraints (i.e. constrain the Ack outputs).

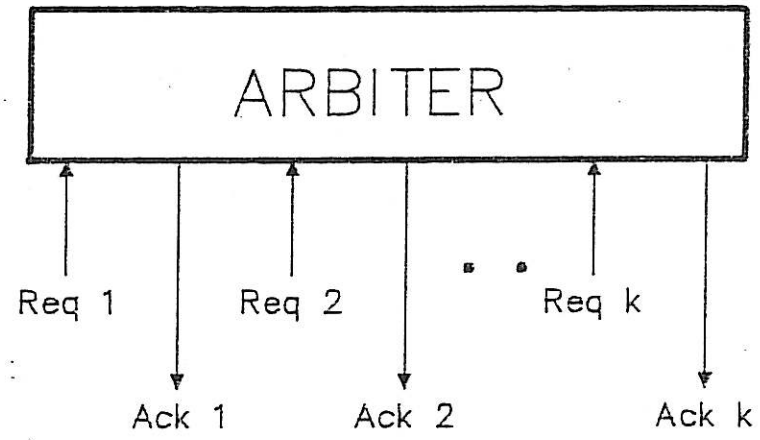


FIGURE 2.1 General Arbiter.

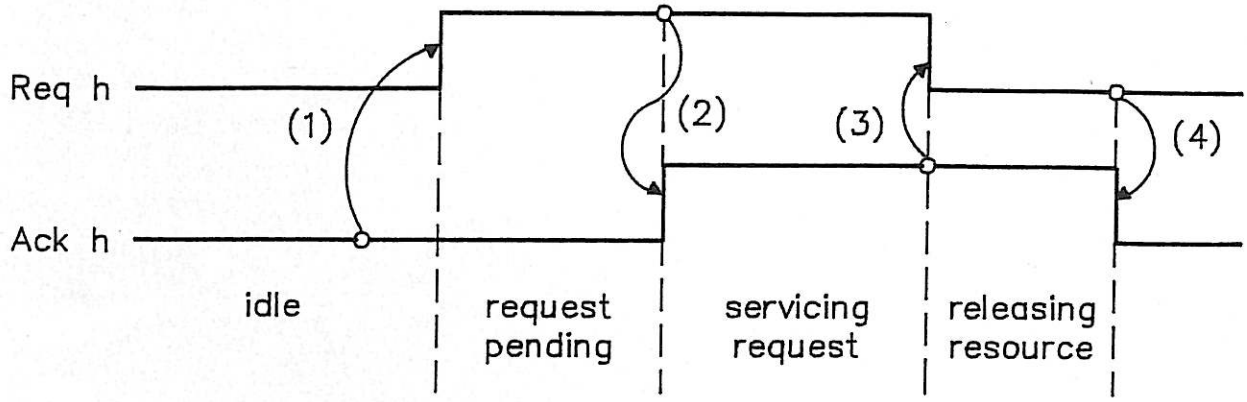


FIGURE 2.2 Request Acknowledge Signal Protocol.

Constraints (2.1) and (2.3) are input or domain constraints (i.e. constrain request inputs). However, no restriction is placed on the timing of the assertion of a request when Ack = 0. This allows requests to occur *asynchronously* in time. In particular, requests can occur arbitrarily close to one another. The ideal arbiter resolves requests irrespective of their relative timing. As will be seen in Chapter 3, this is indeed an *ideal* situation since physically realisable arbiters cannot achieve perfect request resolution within a bounded time under all request timings due to the possibility of metastable behaviour.

2.3 APPLICATIONS OF ARBITERS IN SYSTEMS

Arbiters form a basic building block of computer systems, in particular multiprocessing systems [B.1, C.2, C.5, I.2, L.4, M.1, M.9, P.4, S.1, T.3]. With the advent of cheaper and readily available processors, multiprocessor systems became an attractive alternative to

achieve goals ranging from high processing power to high reliability [B.6, E.2]. New interconnection architectures for processors, memory and I/O devices involve a degree of sharing of the basic hardware resources in a system, such as bus systems [T.3], common memory [C.3, M.5], dedicated arithmetic and vector processors, dynamic memory with automatic refresh control [R.1], I/O interfaces and processors themselves [C.2]. In order to control the access to shared resources, fast arbitration circuits are employed. Because speed is critical in the allocation of hardware resources such as buses, memories and I/O devices, arbiters are implemented as dedicated hardware. The functional level of allocation of these resources is more primitive than software operations performed in the kernel of an operating system and hence it is not efficient to employ software resolution of contention. Indeed, implementations of kernels assume implicitly that arbitration exists, for example, in dual port or shared memory, in order that higher level synchronisation mechanisms can be built up, such as semaphores [H.1].

In this section two applications of arbiters are described: allocation of buses; and interrupt distribution. Detail is kept to a minimum and only basic details of the structures are described.

2.3.1 Bus Arbitration

Much of the literature on arbiters concentrates on bus arbitration [B.1, F.1, M.1, T.3]. Buses provide a unified modular and well ordered interconnection structure for a system. The advantages of shared bus systems are widely recognised and a detailed discussion of bussing structures can be found in [C.11, T.3].

A general structure for a single shared bus with arbitration is shown in Figure 2.3. Request and acknowledge lines are incorporated in

the bus itself. The arbiter shown in Figure 2.3 is centralised, meaning all request lines and acknowledge lines must fan into and out from the arbiter respectively. In the next section decentralised arbitration schemes will be discussed that provide a more modular and reliable structure.

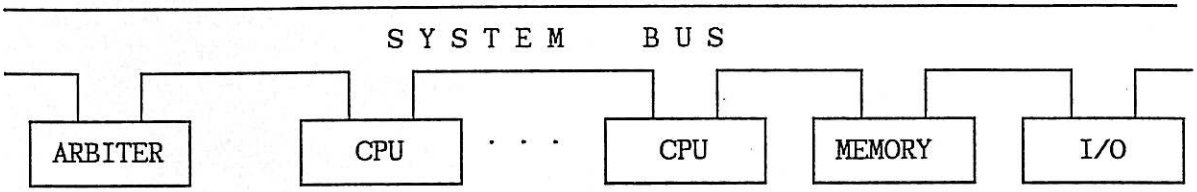


FIGURE 2.3 Centralised Bus Arbiter.

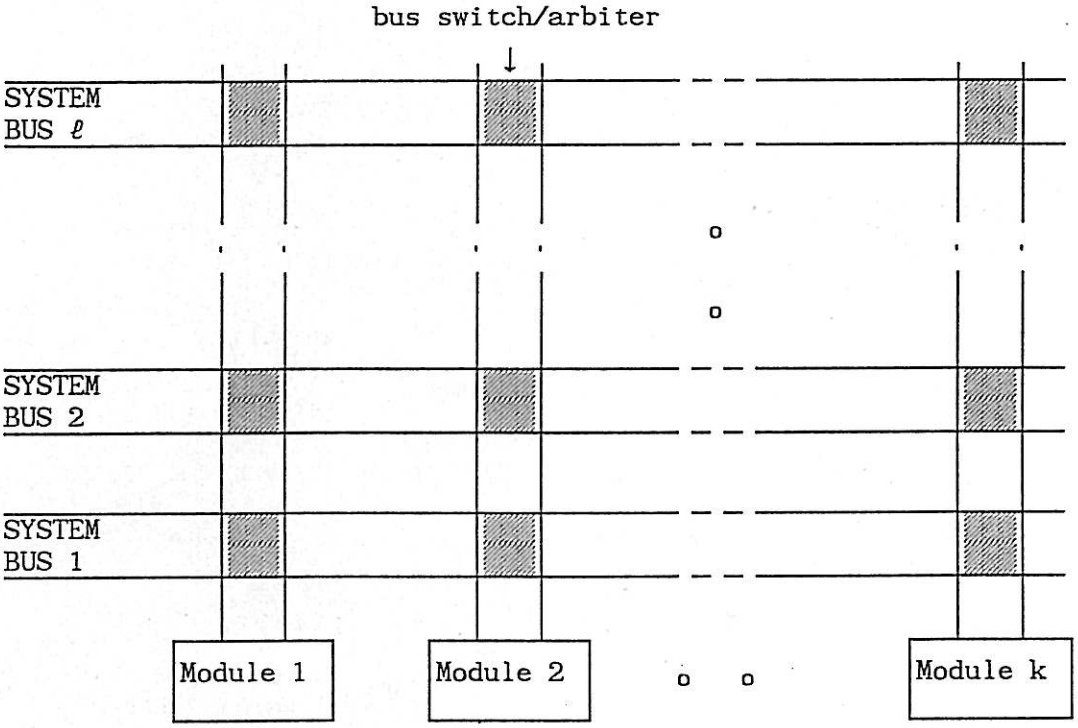


FIGURE 2.4 Multiple Shared Bus System.

A more general situation of a multiple bus system is shown in Figure 2.4. Examples of multiple shared bus systems are POLYBUS [M.1, M.2], C.mmp [W.5], Pluribus [K.2] and iAPX432 [T.4]. To obtain a communication path between a slave module M_s and a master module M_m ,M_m must obtain access to module buses s and m and also to any system bus. This can be achieved using 3 types of arbiters:

(a) **Module Bus Arbiter (MBA)** - resolves requests for the module bus from devices within the module and external modules communicating with the module.

(b) **System Bus Arbiter (SBA)** - Each system bus has its own system bus arbiter which resolves requests from modules for the bus.

(c) **Multiple Bus Available Arbiter (MBAA)** - This arbiter is associated with each module. Its function is to select one of possibly many system buses that acknowledge the module's request for a system bus, and reset the remaining system bus requests from its module.

The interconnect structure for arbiters SBA and MBAA is shown in Figure 2.5 for one module and many system buses. A module requests a system bus by asserting Req of the MBAA, in Figure 2.5, resulting in the MBAA asserting all Bus Req signals to the SBA's. The SBA's respond by asserting Bus Avail signals when their corresponding system bus is free. The MBAA selects a unique available system bus by arbitrating the Bus Avail signals and asserting one Bus Ack. All other corresponding Bus Req signals are reset by the MBAA.

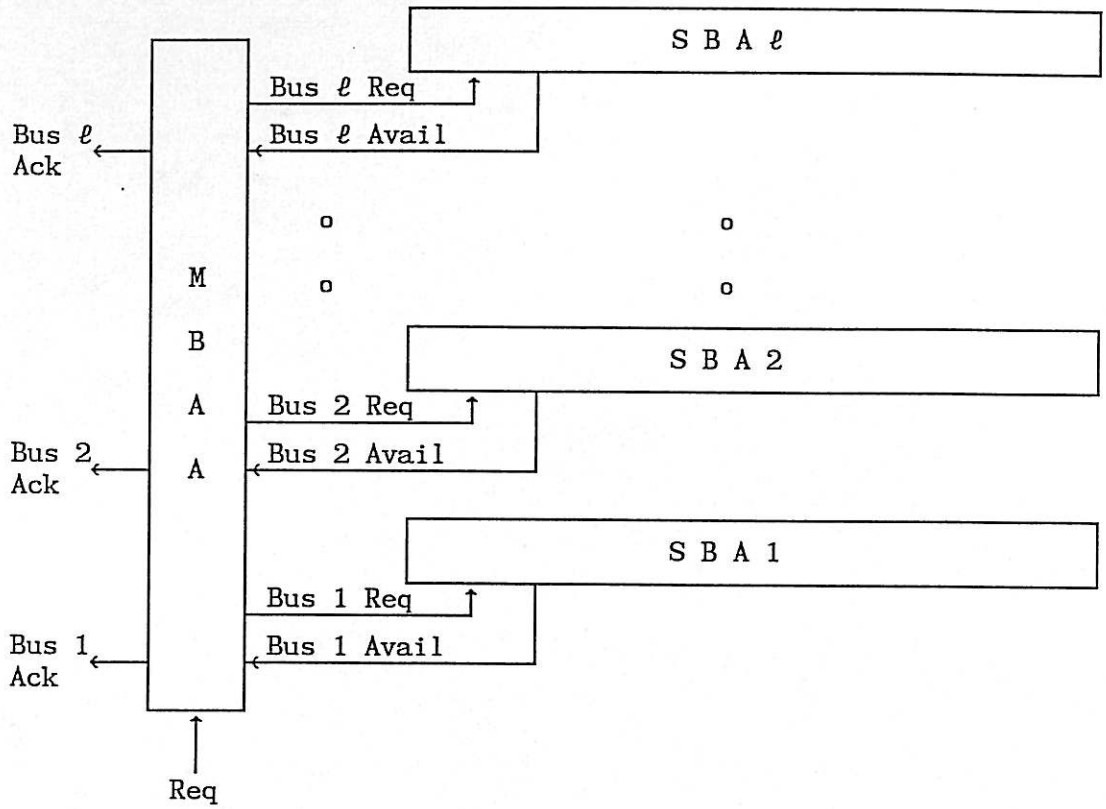


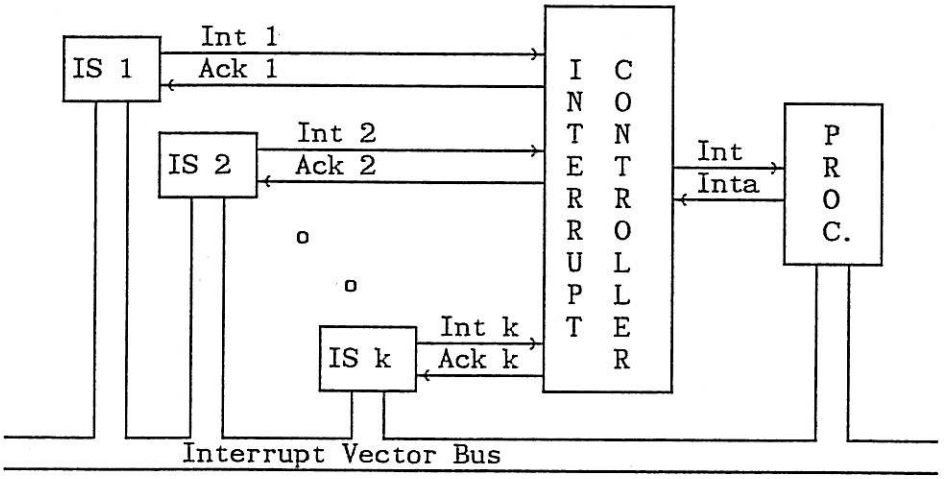
FIGURE 2.5 Multiple System Bus Arbitration

(One module circuit only).

Note that a Bus Req h can be reset before Bus Avail h is asserted in an SBA and Bus Avail h can drop before Bus Ack h is asserted in an MBAA. This violates the signal protocol described in the previous section, namely a request must be held until the corresponding Ack is asserted. The request protocol can be preserved by delaying resetting of Bus Req h until Bus Avail h and Bus Ack h are both asserted, but this incurs unnecessary delay and holding of buses. Another solution is to carefully design the arbiters to tolerate resetting of unserviced requests.

2.3.2 Interrupt Distribution

Interrupts allow processors to respond to asynchronous events without continuously checking event status (busy waiting). An interrupt source requires mutually exclusive access to the processor, and consequently multiple interrupts must be arbitrated. The arbitration is performed by an interrupt controller [I.3] as shown in Figure 2.6.

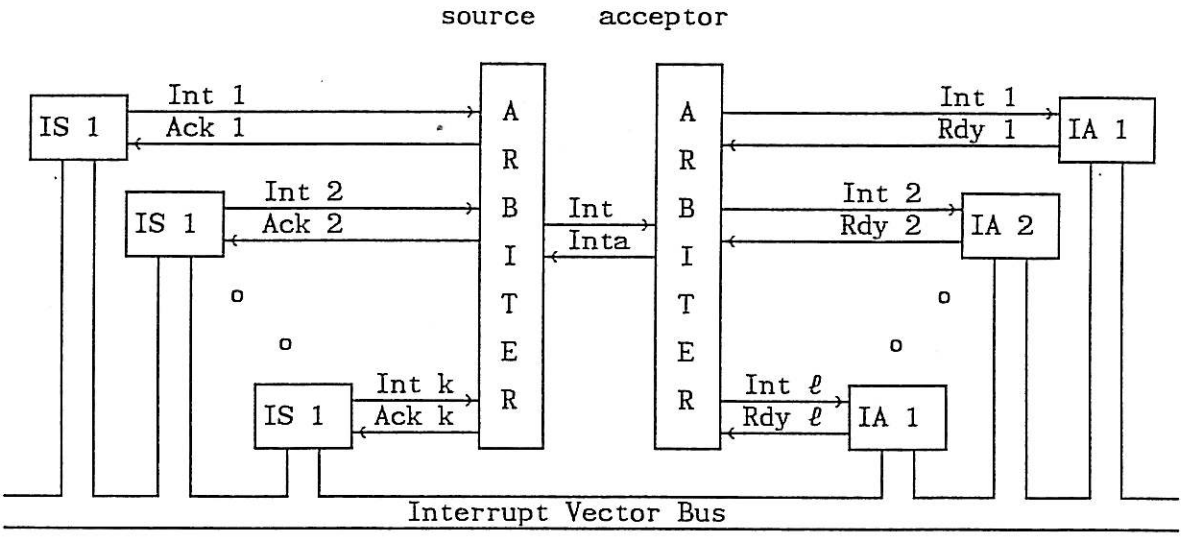


IS=Interrupt Source

FIGURE 2.6 Interrupt Controller For Single Processor.

The interrupt controller has inputs Int 1, Int 2, .. , Int k from k Interrupt Sources (IS) and corresponding acknowledge outputs Ack 1, Ack 2, .. , Ack k to indicate to the IS that its interrupt vector should be placed on the Interrupt Vector Bus (IVB). The controller interrupts the processor via the Int line and the processor responds when ready to accept an interrupt by asserting Inta. The controller forms Int as the OR function of all Int i, i=1, .. , k and conditions the Ack i outputs on Inta from the processor.

The case of a single Interrupt Acceptor (IA), for example the processor in the previous example, can be generalised to multiple IA's and multiple IS's [C.2] as shown in Figure 2.7.



IS=Interrupt Source, IA=Interrupt Acceptor

FIGURE 2.7 Multiple Interrupt Distribution to Multiple Interrupt Acceptors.

An IA signals it is primed to accept an interrupt by asserting the Ready line. Should at least one interrupt be pending from an IS, the acceptor arbiter selects one IA and assigns it to the IS selected by the source arbiter. The interrupt vector is transferred via the IVB (which contains the necessary control lines for transfer protocol). Since the IVB is tied up for only the short time necessary to transfer an IS's identity, only one IVB is employed. The interrupt distribution could be achieved using multiple IVB's, similar to the multiple bus example previously, should the IVB loading demand it.

From the previous application examples, it is clear that the characteristics of arbiters are crucial for system overall performance. The fundamental importance of the arbiter in the system architecture has been highlighted.

2.4 ARBITRATION DISCIPLINES AND IMPLEMENTATIONS

Arbiters have been characterised so far in this chapter by their mutually exclusive selection process on the pending requests. The discipline and manner in which a request is selected from a number pending is the subject of this section. Examples of implementation structures are also given.

2.4.1 General Model of Arbitration Disciplines

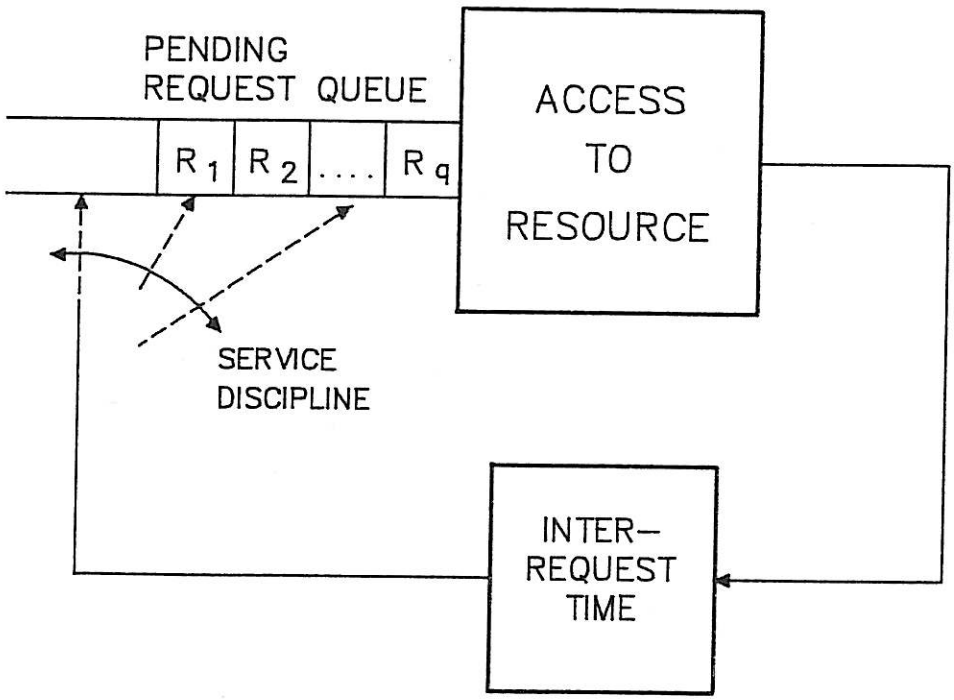


FIGURE 2.8 General Service Discipline Model.

The arbitration process can be viewed in a manner depicted in Figure 2.8. Requests are placed on a queue of pending requests. The position in the queue of a newly arriving request is determined by the service discipline, and access to the resource is given to the request at the head of the queue when the current service is completed. That is, the discipline is non preemptive.

The service discipline can be dependent on many parameters. For example, the discipline can be a function of requester identity, previous service and requesting history, the arrival times of requests, or even pseudo random events such as the physical location of a rotating grant signal [M.1].

2.4.2 First Come First Served (FCFS)

FCFS is the simplest and most natural discipline to define in terms of the general model in Figure 2.8. Arriving requests are placed on the end of the queue, and so requests are serviced in order of arrival.

It is shown in [S.1] that FCFS is the optimal service discipline in the sense that it minimises the standard deviation of the waiting times of all requests provided all requester statistics are identical. The proof of this result is based on showing that the standard deviation increases every time a request is swapped ahead in the queue with an earlier arriving request.

Approximate FCFS can be implemented in a distributed [S.1] and centralised [L.4, S.1] manner, but more hardware is necessary than most other disciplines. In [S.1], the FCFS implementation is based on assigning ticket numbers to requests, with the arbitration for the next ticket number achieved through a secondary discipline. It is the speed of the secondary arbitration that determines how closely the overall discipline approximates FCFS. In order that the ticket assignment scheme is as fair as possible, the secondary arbitration must be unbiased and as fair as possible. Sharma and Ahuja [S.1] study the effect on the overall arbiter performance with various dynamic priority disciplines used for the secondary arbitration. Dynamic priority disciplines are discussed in

Section 2.4.4. The ticket assignment scheme effectively shifts the basic arbitration process to the secondary arbiter and, thus, the approximate FCFS arbiter relies on the fast operation of a simpler arbitration policy. Hence, the study of simple policies that are amenable to fast implementation is of importance to FCFS arbiter design.

Another approximate implementation of FCFS [L.4] is a centralised clocked scheme. Requests are synchronised to a clock and then resolved within a request priority resolution module which can be programmed to implement FCFS. The order of arrival of requests after synchronisation is stored using k(k-1) R-S flip-flops, where k is the number of requesters. When two or more requests arrive within the same clock period they are ordered using differing delays (labelled t_{Zi} in [L.4]) in each synchronised request line. The fixed delays effectively implement a fixed priority secondary arbitration policy, resulting in a biased total arbitration scheme.

2.4.3 Fixed Priority Arbitration Discipline

Each requester is assigned a fixed unique identity from 1 to k where k is the number of requesters. It is assumed that requester h has priority over h+1 (h=1, ..,k-1) and so requester 1 has the highest priority, although sometimes the opposite convention is adopted.

The arbitration discipline is to service the highest priority pending request. In Figure 2.8, this corresponds to always inserting an arriving request in the queue to maintain $R_1 > R_2 > .. > R_q$. With this priority structure, persistent high priority requests can lock out low priority requests. This is a well known problem with fixed priority arbiters [B.1] but they still find wide application [B.3, I.2, T.3]. Systems are often composed of greatly different requester characteristics and their treatment is often suited to a priority structure such as is

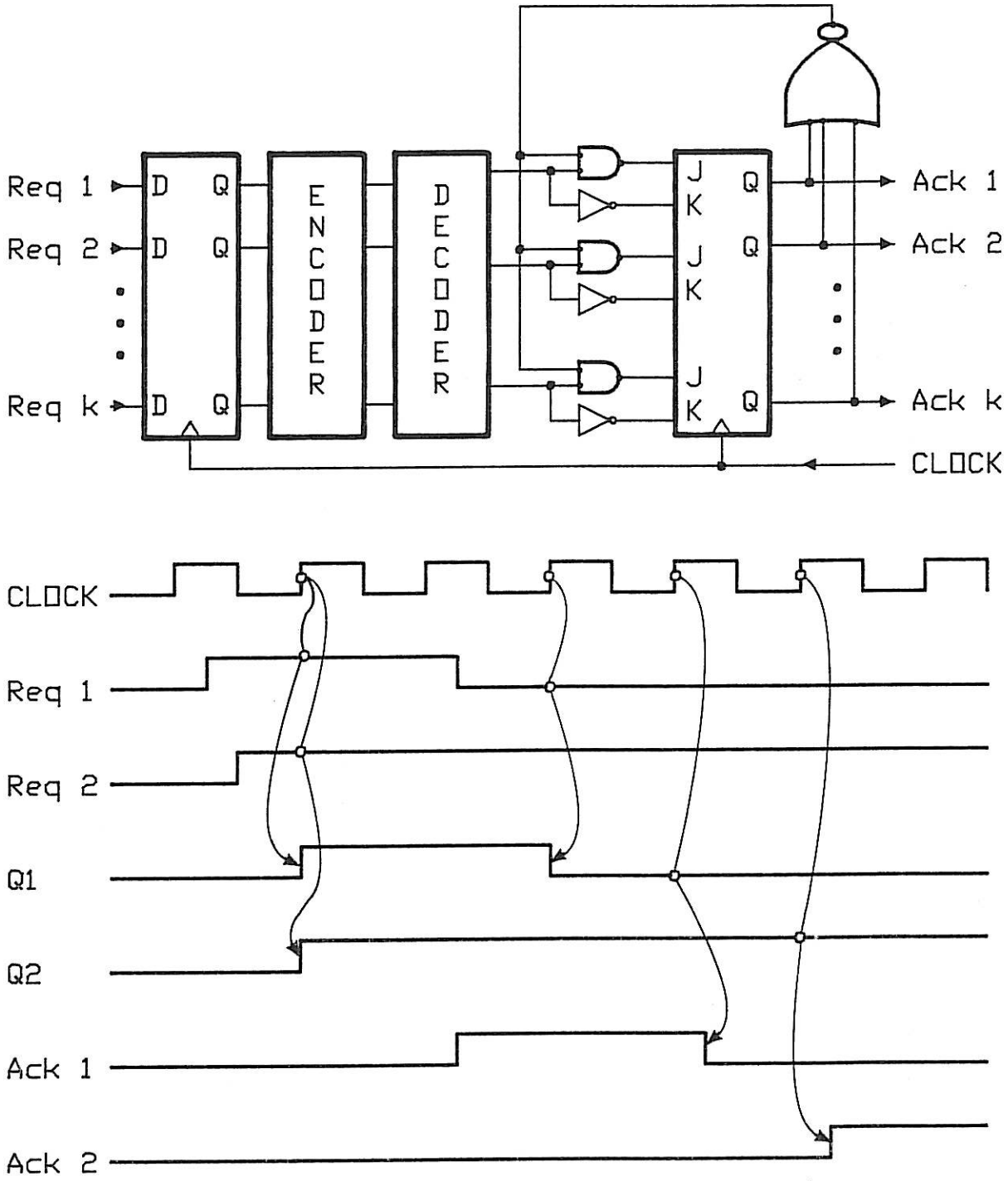


FIGURE 2.9 Centralised Clocked Fixed Priority Arbiter.

present in the PDP-11 based systems. Time critical interrupts, such as from USARTS on a high baud data link, should take priority over a non critical slow interrupt, such as a keyboard.

Figure 2.9 shows an implementation of a centralised clocked fixed priority arbiter. The arbiter can be divided into three parts:

(i) The request synchroniser consisting of clocked D-type flip-flops;

(ii) Priority resolution circuit of encoder-decoder combination which selects the highest priority synchronised request;

(iii) The output latch/non preemption circuit - once an Ack output is asserted only $1 \rightarrow 0$ Ack changes can occur. This prevents a higher priority request overriding the service of a lower priority request already in progress.

The arbiter in Figure 2.9 is centralised and consequently is difficult to expand to incorporate more requesters beyond a preset initial limit imposed by the designer. The circuit utilises fast parallel priority resolution circuitry in the encoder exploiting the centralised nature of Req inputs. This has the disadvantage of requiring many Req and Ack lines running to the requesters throughout the system.

In order to overcome the disadvantages of a centralised arbiter, a decentralised version can be employed. The decentralised example shown in Figure 2.10 is an asynchronous or non clocked digital circuit. Priority resolution is performed using a Daisy Chain [B.3, T.3] which threads through each arbiter module. When a module receives an asserted Daisy In (DI) signal it passes it on to Daisy Out (DO) if the module does not require the resource. If the module has a stored request it blocks the daisy chain. Thus, the daisy propagates from the highest priority module

to the closest module with a request latched. The wired-or lines $\overline{\text{Req Pend}}$ and $\overline{\text{Busy}}$ control the resetting of the daisy chain after a service and the prevention of preemption by a high priority request during the servicing of a low priority request. The circuit details of a module are presented in Section 4.3.1.

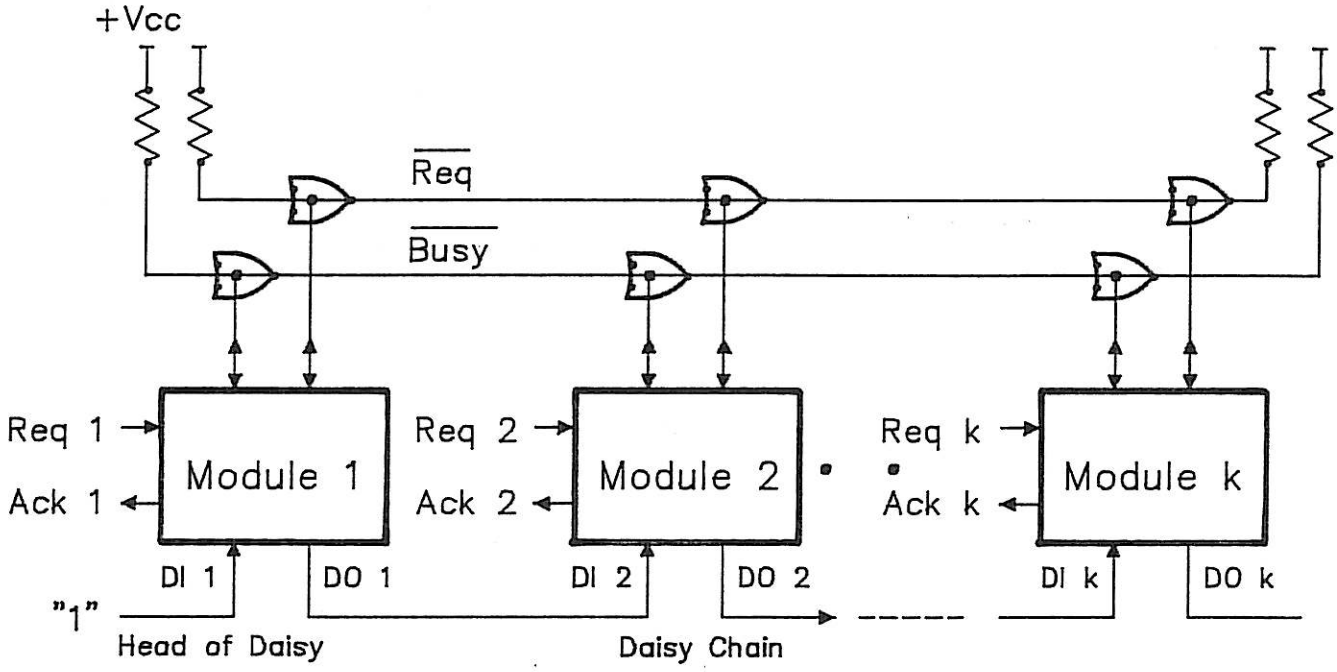


FIGURE 2.10 Decentralised Unclocked Daisy Chained Fixed Priority Arbiter - Module Interconnections.

The decentralised arbiter has several important features:

(i) It is modular and can be easily expanded by adding further modules. A limit to expansion may be imposed by timing constraints depending on the implementation.

(ii) The number of global lines is small.

(iii) The priority is a function of physical positions of requesters and resolution can be slow due to the serial nature of the daisy chain. Parallel resolution on a

distributed basis is however possible as used in the synchronous backplane interconnect of the VAX 11/780 [B.3] and IEEE Future Bus [T.1]. These schemes require additional interconnections to distribute priority information.

The circuitry of Figure 2.10 does not rely on a clock for its operation. As will be discussed in more detail in Chapter 7, this has the advantages of greater speed and less synchronisation events within the arbiter. Also implementing fault tolerance is simpler for asynchronous circuits. However, extra care in design is needed to avoid races, hazards [U.1] and to satisfy all timing constraints required by components (some of which are impossible to satisfy due to the asynchronous nature of requests, but this problem also occurs in clocked arbiters as is discussed in Chapters 3 and 7).

2.4.4 Dynamic Priority Arbitration Disciplines

The fixed priority arbiter of the previous section treats requests in an unfair or asymmetrical manner which can be detrimental to the performance in some applications where no preference for particular requesters can be justified. Dynamic allocation of priorities to requesters can overcome this problem if the allocation is symmetric with respect to requesters. After the arbitration decision to service a request is completed, priorities are permuted in a manner that favours no requester on an average basis when requesters are statistically identical. Algorithms discussed in this section are round robin, next robin and least recently used (LRU).

Let $p_n : \{1,2, \ldots,k\} \longrightarrow \{1,2, \ldots,k\}$ be a permutation mapping from the set of requesters to the set of priorities on which the arbitration of

the n^{th} service is based. If $p_n(h) < p_n(i)$, requester h has priority over requester i for the n^{th} service. Each dynamic allocation algorithm is characterised by the mapping p_n. The permutation mapping is also called the ***discipline state*** of the dynamic priority scheme. All the dynamic priority disciplines discussed here have Markovian discipline states since the n^{th} state depends only on the $(n-1)^{th}$ and request/service behaviour during the $(n-1)^{th}$ service.

Round robin is defined for $n > 1$ by

$$p_n(h) = \begin{cases} p_{n-1}(h-1) & , \ 1 < h \leq k \\ p_{n-1}(k) & , \ h = 1 \end{cases}$$

and for $n = 1$ (2.7)

$$p_1(h) = h \qquad , \ 1 \leq h \leq k$$

The priority rotates every service, independently of request behaviour. This scheme is symmetric with respect to all requesters and hence favours none.

Next robin [B.1, B.4, I.3] is dependent on the requester serviced at time n-1, denoted s. The scheme is defined as follows. For $n > 1$

$$p_n(h) = \begin{cases} h - s + k & , \ 1 \leq h \leq s \\ h - s & , \ s < h \leq k \end{cases}$$

and for $n = 1$ (2.8)

$$p_1(h) = h \qquad , \ 1 \leq h \leq k$$

The last requester serviced is assigned the lowest priority and the

remaining requesters follow in order modulo k. Next robin can be implemented in a decentralised form [B.4] as shown in Figure 2.11. The last module to assert its Ack signal becomes the head of the daisy chain and hence lowest priority. That is, the daisy starts propagating from the head module, giving all other modules the opportunity to block the daisy chain before the head module receives an asserted Daisy In.

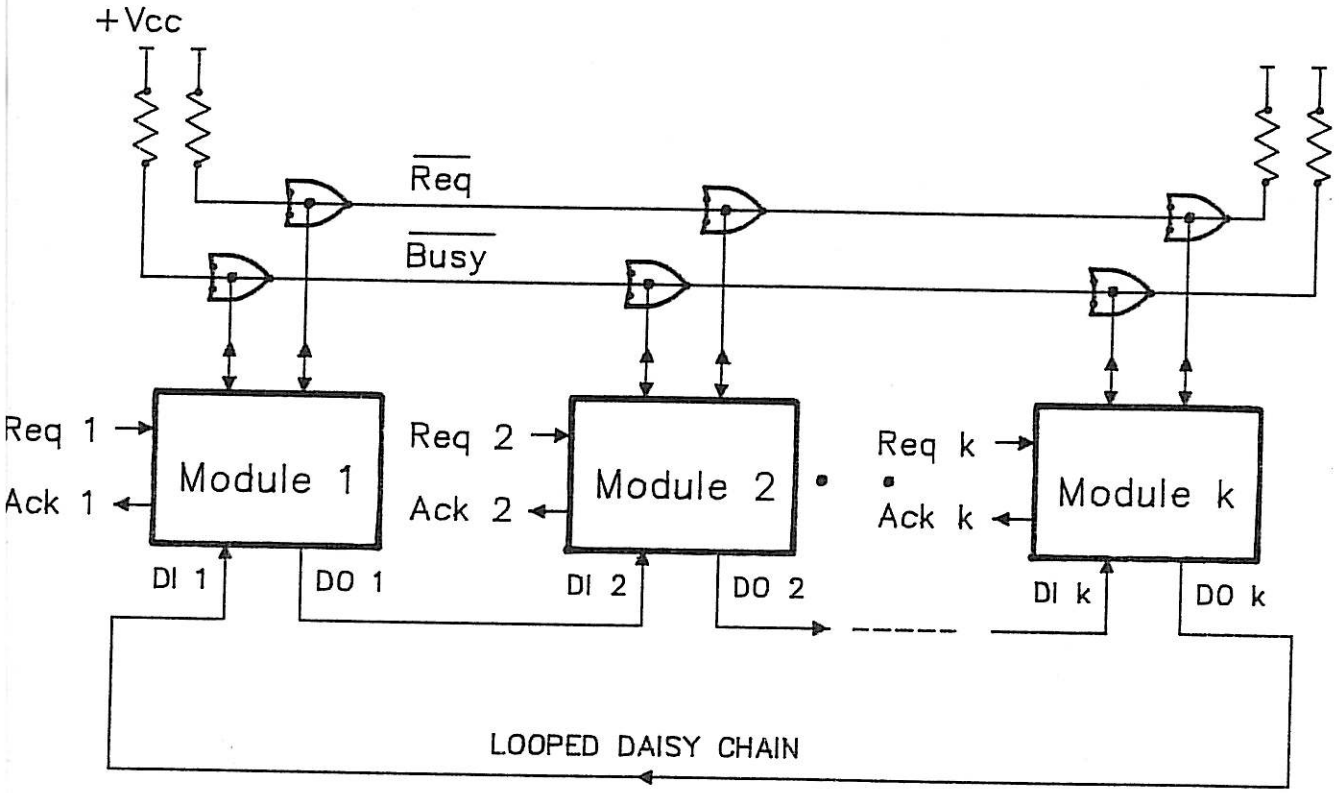


FIGURE 2.11 Daisy Chain Implementation of Next Robin.

The LRU dynamic priority algorithm assigns priorities in order of least recent accesses to the resource, with highest priority given to the requester not accessing the resource for the longest time. After the servicing of requester R_s, priorities are reassigned as follows [B.1] (refer also to Table 2.1):

for $n > 1$

$$p_n(h) = \begin{cases} k & , h = R_s \\ p_{n-1}(h) & , p_{n-1}(h) < p_{n-1}(R_s) \\ p_{n-1}(h) - 1 & , p_{n-1}(h) > p_{n-1}(R_s) \end{cases}$$

and for $n = 1$ (2.9)

$$p_1(h) = h \quad , 1 \leq h \leq k$$

PRIORITY	$(n-1)^{th}$ SERVICE (R_s serviced)	n^{th} SERVICE
1	R_1	R_1
2	R_2	R_2
:	:	:
s-1	R_{s-1}	R_{s-1}
s	R_s	R_{s+1}
s+1	R_{s+1}	:
:	:	R_k
k	R_k	R_s

TABLE 2.1 LRU Dynamic Priority Mapping

That is, R_s is assigned lowest priority, having most recently used the resource, and requesters with previous priorities lower than R_s's previous priority are upgraded in priority and those with higher priorities are unchanged. The mapping is invertible and hence priorities remain unique.

After at most k-1 requesters receive services the priorities will be in order of least recently used, independently of the initial priorities. This can be seen as follows: If two requesters have been serviced, their priorities will be in order of least recently used for the following reasons: Whilst a requester is not serviced its relative priority does not change with respect to other requesters not serviced. In the time after the last of the two requesters is serviced, their relative order of priorities does not change and so the last requester serviced must have a lower priority than the other, because the last requester serviced was allocated the lowest priority after being serviced. Applying this to all pairs of serviced requesters gives the least recently used priority scheme. At most one requester is not serviced at all, and it must receive the highest priority because its priority can never be lowered and all the other requesters were placed below it after servicing.

The LRU, and other dynamic priority schemes, can be implemented centrally using sequential logic in place of the encoder-decoder in Figure 2.9. A decentralised scheme using a daisy chain cannot be employed for LRU because of the serial order of priorities changes dynamically, however distributed parallel priority resolution such as employed in FUTUREBUS and FASTBUS arbitration [T.1, T.2] along with sequential logic for dynamic priority allocation within modules [F.1] presents a practical alternative.

2.4.5 Batched Disciplines

The concept of batched disciplines is essentially new and has not been previously identified or defined in the literature to the author's knowledge. Some specific arbiter designs [C.1, C.2, C.5] have employed disciplines classified as batched in this section.

Batched arbitration disciplines can be described in terms of the general model in Figure 2.8 by placing a dummy request, called a **batch marker**, at the end of the queue whenever a batch marker reaches the front of the queue (i.e. is "serviced"). Initially, when no requests are pending, a batch marker is placed at the end of the queue, and from then on, the queue will always contain exactly one batch marker. The function of a batch marker is to restrict the service discipline so that it acts only on the queue after the batch marker. Reordering of the queue when a new request arrives cannot then disadvantage or change the ordering of batched requests (i.e. those in front of the batch marker). In an ideal batched arbiter, the service time of a batch marker is zero, however non zero batch marker service times can be used to represent inter-batch time durations of a more realistic arbiter model, as adopted in Chapter 4.

The batching concept can be implemented by a lock out mechanism on requests occurring after a batching point (i.e. immediately after "servicing" of the batch marker). When the next batching point occurs, requests pending are stored within the arbiter for later servicing. These stored requests are termed a *batch* of requests and correspond to those requests ahead of the batch marker in the request queue of Figure 2.8. Only batched requests are serviced, and the order of servicing within the batch is a function of the particular batched discipline. A batching point occurs a short time after a batch of requests has been serviced or in the case of the arbiter *idling* (i.e. no requests pending), a short time

after the first request occurs. The batching concept was motivated by the desire to prevent hogging of the shared resource by high priority persistent requests in fixed priority arbiters.

Batched disciplines can be classified as either *derived* or *non derived* A derived batched discipline is obtained by applying the batching concept to any existing discipline. That is, the original discipline is restricted so that requests cannot be reordered across the batch marker. Batched requests only are serviced in the order determined by the original discipline over which batching is applied, and those requests that are not yet batched are treated as though they are not yet present. It is interesting to note that FCFS and batched FCFS are the same discipline.

It is possible to define *non derived* batched disciplines which are not defined as batched versions of previous disciplines. A non derived batched discipline still has the property that requests occurring after a batching point are locked out until all the batched requests have been serviced. The order of servicing of batched requests in a non derived batched discipline may be a function of the position of the batch marker. Two examples of non derived disciplines are non derived batched forward round robin [C.5] and non derived batched reverse round robin. These are dynamic priority batched disciplines which only reassign priorities when a batching point occurs (i.e. when the batch marker reaches the front of the request queue), as distinct from non batched dynamic priority disciplines which reassign priorities every service. Non derived batched forward round robin is defined to reassign priorities by rotating them in a forward direction after every batch with the same permutation mapping as round robin in (2.7). Non derived batched reverse round robin rotates priorities in the reverse direction at the end of servicing a batch of

requests by assigning requester 2's previous priority to requester 1's priority and so on as in the following definition of $pb_n(h)$, the priority of requester h during servicing of the n^{th} batch. For $n > 1$

$$pb_n(h) = \begin{cases} pb_{n-1}(h+1) & , 1 \leq h \leq k-1 \\ pb_{n-1}(1) & , h = k \end{cases}$$

and for $n = 1$ (2.10)

$$pb_1(h) = h \quad , 1 \leq h \leq k$$

When labelling batched disciplines they will be assumed to be derived disciplines unless otherwise stated.

An example of a decentralised batched fixed priority arbiter is shown in Figure 2.12. A daisy chain implements the fixed priority resolution, with the module nearest to the head of the chain, module 1, receiving highest priority. The common line forms the wired-or function of latched requests (i.e. those that have been batched) and when asserted locks out new requests from being latched. After the servicing of a batch of requests, the common line resets and each module latches pending requests and resets the daisy chain. Once the common line is asserted due to a latched request, the daisy chain is enabled and the batch of requests is serviced in order of the modules along the daisy chain. More details can be found in Section 4.3.1.

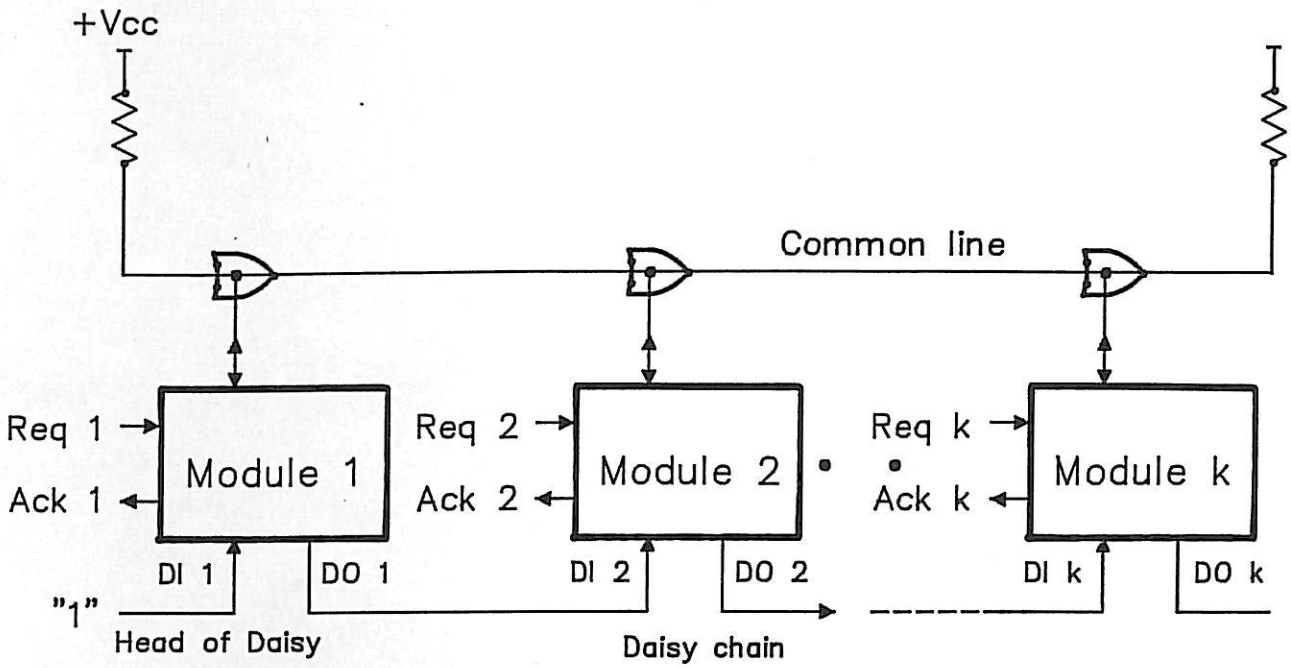


FIGURE 2.12 A Decentralised Batched Fixed Priority Arbiter.

The $\overline{\text{Busy}}$ line in the non batched version of Figure 2.10 is unnecesary in the batched version of Figure 2.12 since the common line is sufficient to distinguish batches. Thus, it can be seen that in this basic example of a fixed priority arbiter, the batching implementation is in fact simpler that the non batched version. In any general distributed arbiter, batching can be incorporated using a distributed wired-or line, which is asserted if a request has been latched, and request latches enabled when the wired-or line is unasserted. For example, the non derived forward [C.5] and reverse round robin batched disciplines can be implemented using a priority daisy chain connected in a loop, a wired-or common line and another daisy chain to rotate the priority daisy chain head at the end of servicing a batch of requests.

In the proof [S.1] that FCFS minimises the standard deviation of waiting times (when service times are constant and request statistics are identical), it is shown that the standard deviation of waiting times

increases every time a request is swapped ahead in the queue with an earlier arriving request. One may try to extend this result to a comparison between batching and non batching versions of the same discipline, argueing that the batch marker acts to reduce the number of swaps performed by the discipline and hence the standard deviation. This is true in most cases, but not all, as borne out by simulation results in Chapter 8. The reason being that, given the same request stream, the queue order and discipline state seen by an incoming request may be different, and also the discipline changes the arrival process in terms of the distribution of request identities (but not queue length). Thus comparison can only be made between disciplines where one of them makes no discrimination on the basis of requester identity (e.g. FCFS).

2.5 CORRECTNESS PROVING

Arbitration circuits require careful design because of the asynchronous nature of their inputs and strict output requirements outlined in Section 2.2. The difficulty lies in the possibility of inputs changing while the circuit state has not yet recovered from a previous input change. This difficulty in the design process can be alleviated by synchronising the request inputs to a clock and then designing a synchronous circuit to arbitrate the resulting synchronous requests. However, asynchronous circuits, where no internal clock synchronism is present, have speed advantages over synchronous circuits irrespective of any differences in metastable reliability which are discussed in Chapter 7. Once a design, either synchronous or asynchronous, has been formulated it needs to be tested for all possible input sequences and device parameter variations before confidence can be placed in it.

An example of a FCFS asynchronous arbiter design with three request inputs is shown in Figure 2.13. The three R-S flip-flops, FF1, FF2 and FF3, which are assumed to be set when both R and S are asserted, store the arrival order of requests as follows: FF1 is reset if Req 2 arrives before Req 1; FF2 is reset if Req 3 arrives before Req 1 ; FF3 is reset if Req 3 arrives before Req 2. The circuit functions perfectly well when request input changes are sufficiently separated to prevent marginal triggerring of the flip-flops. Ignoring metastable behaviour, which is discusssed in Chapter 3, two inputs changing together result in a valid state transition when the appropriate flip-flop settles to either logic state. However, a problem arises when all three inputs change together as shown in the timing diagram of Figure 2.13. Since all three flip-flops are marginally triggered by the attempt to decide on the request arrival order, they could settle to any of the eight possible states, six of which are valid and the remaining two are inconsistent. The inconsistent states correspond to circular queue orderings (1→2→3→1 and 1→3→2→1, where → denotes "arrived before") and result in deadlock with no Ack asserted, An example is shown in the timing diagram of Figure 2.13.

This example is quite simple in hindsight, but for a more complex arbiter not all state transitions may be as easily understood. Additional difficulties arise in verifying that all device timing constraints are met, such as minimmum pulse widths on flip-flops. Some constaints *cannot* be met as discussed with regard to metastable behaviour in Chapter 3. Correctness proving, similar to that employed in some computer software, can be applied to hardware design. Axioms can be formulated for devices, input protocols and connections, with theorems developed and proved rigorously within a first order predicate calculus framework. This

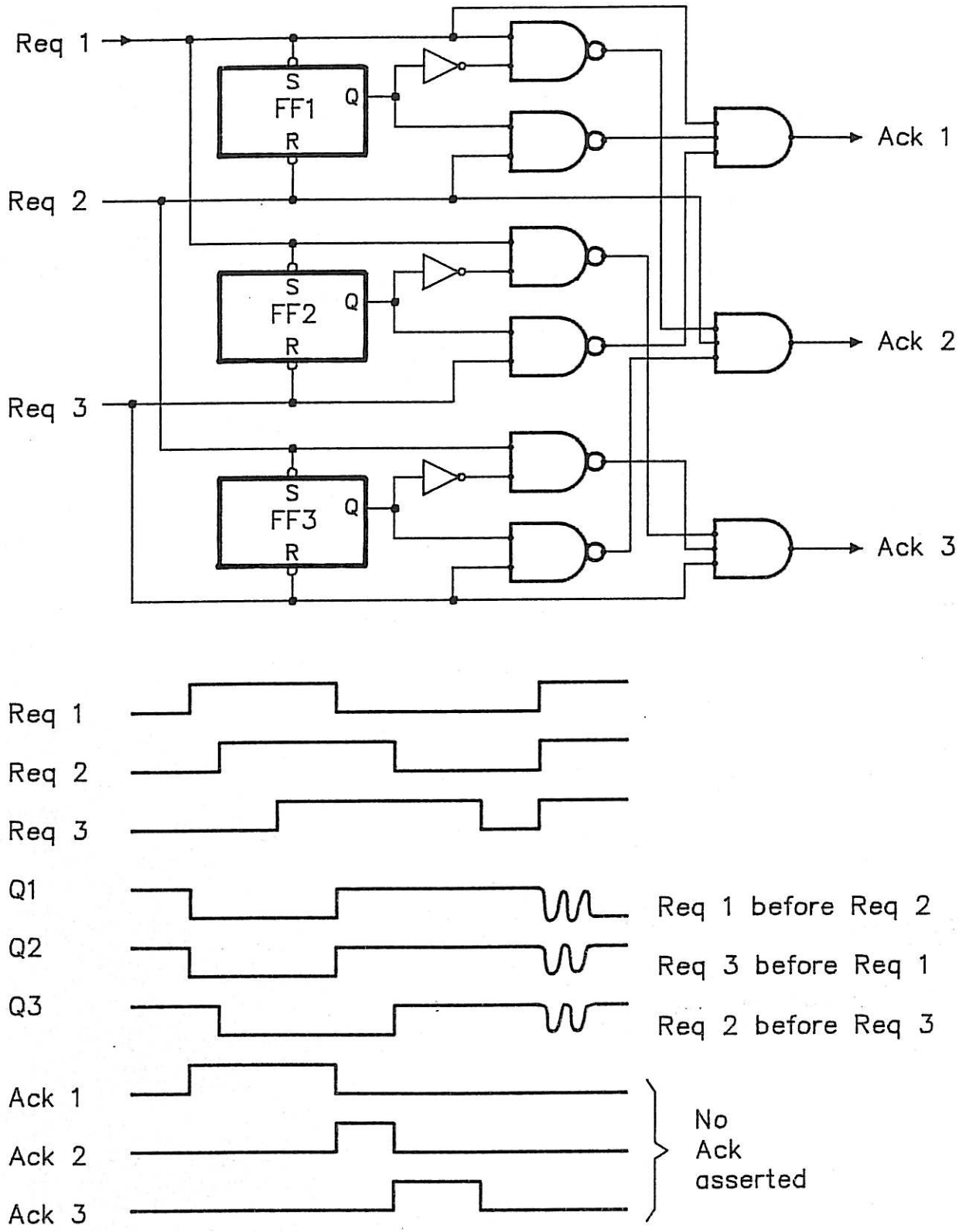


FIGURE 2.13 An Example of a FCFS Three Input Arbiter.

approach is used in [B.2] to formally show equivalence of the arbiter, synchroniser, latch and inertial delay. Naturally, in the development of axioms, modelling assumptions are made for devices and interconnection structures that may only be approximate or even erroneous. One may model an arbiter, synchroniser, latch and inertial delay to be free of metastable behaviour and acknowledge that the assumption has been made [B.2]. Another example of simplified modelling that is often overlooked in digital design is that of the wired-or open collector driven common line [G.5]. To achieve a distributed OR function, open collector drivers are connected to a distributed line which is pulled high at both ends by resistors. It is often assumed that the line is low whenever at least one open collector driver is active. This may not always be true, since when a driver becomes inactive it redistributes the current in the line which can cause transient glitches to be seen in the line [G.5].

The process of correctness proving can be extremely tedious to perform by hand, and semi-automated theorem provers have been developed [W.4, S.3, S.4] which are claimed to be a practical development tool, especially in areas that require high reliability such as fault tolerant computers. It is pointless designing a circuit to be extremely reliable when the design itself cannot be verified to be correct in a rigorous manner. From the author's own experience [K.8] and others [W.4], the exercise of proving correctness can identify subtle design flaws that may never be discovered by conventional hardware testing, and yet can cause later problems which are possibly intermittent and difficult to trace.

Correctness proving is not pursued further in this thesis and the interested reader is referred to specific work done by the author [K.8] and others [W.4, B.2, S.3, S.4].

2.6 PERFORMANCE MEASURES

Once an arbiter design has been established to be functionally correct, the behaviour of the arbiter can be predicted from a model based on the functional description. By analysing the model under certain excitation assumptions, performance characteristics can be obtained. The excitation of the arbiter is controlled by the request inputs. The Req h input controls the length of the *re-request time* (i.e. the time from Ack h going low to Req h asserted high, labelled "idle" in Figure 2.2) and the *service time* (i.e. the length of time both Req h and Ack h are high). The performance of an arbiter is relative to the assumptions regarding the re-request time and service time distributions for each requester.

Two aspects of arbiter performance are considered in this thesis: service performance and reliability performance. These are qualitatively introduced separately, with the discussion of reliability performance deferred to Section 2.7. Precise mathematical definitions of the performance measures are given in Chapters 5 and 6.

2.6.1 Service Performance

The service performance is based on the request to service response of the arbiter. Service performance measured in terms of waiting times are discussed with reference to times between transitions of the request acknowledge protocol of Figure 2.2. Apart the re-request and service times, which are controlled by the requester, the time durations of transitions from (1) to (2) of Figure 2.2, called the **waiting time**, and from (3) to (4), called the **release time**, are determined by the arbiter. Performance measures can be derived from waiting times, such as:

(i) the mean waiting time over all requests, **MWT** ;

(ii) the mean waiting time for requester h, **MWT(h)** ;

(iii) the standard deviation of waiting times **STDW** ;

(iv) the standard deviation of waiting times for requester h, **STDW(h)** .

MWT is independent of the arbitration discipline provided all requesters are statistically equivalent and the arbitration discipline is ideal in the sense that no logic delays or decision times occur [B.1, S.1]. For an ideal arbitration discipline, MWT is a measure of the contention between requests or the request loading (i.e. the inverse of the mean re-request time relative to the mean service time). MWT(h) allows comparisons between requesters' waiting times to be made and can indicate the degree of fairness of a discipline, such as fixed priority. Disciplines which treat requesters in a symmetric fashion, such as FCFS, round robin schemes and LRU, have MWT(h)=MWT for all requesters h, provided each requester's request properties are the same. Two disciplines that treat requests symmetrically may still be judged on their service performance. The distribution of waiting times of the disciplines may be different, even though their means coincide. The probability of a requester waiting longer than a certain time may be a useful measure of performance. Ideally, the probability of waiting longer than *any* time would proviode a good basis for comparison, amounting to obtaining the entire distribution function of waiting times. However, in practice this is difficult to obtain, and a less accurate measure is often used, namely STDW [S.1, B.1]. When the spread of waiting times is considered a good measure of fairness, STDW may be a measure of fairness in this sense, where a discipline with a smaller STDW is considered fairer that one with a higher STDW. Disciplines with a smaller STDW result in more constant waiting times for

requests, and tend to treat requests more evenly in that sense. The notion of fairness is not, however, uniformly agreed upon. The LRU discipline is considered the fairest in [F.1, K.13], even though LRU does not minimise STDW [S.1]. In an analogous social situation to an arbiter, "queueing" (i.e. FCFS) is considered fairest when people are waiting for service in a shop, and those people who "jump" the queue are frowned upon as being unfair (a variant occurs when someone claims to require quick service and is allowed to "fairly" receive service ahead of others). A definition of fairness is avoided in this thesis due to its application dependence and also because other more precisely understood measures, such as those mentioned above, can be used instead.

Waiting time measures of performance are useful in applications sensitive to response times (e.g. interrupt distribution), but another measure is useful in throughput applications, namely the proportion of time allocated to requester h, **PROP(h)**, which is the ratio of the sum of service times of requester h to the total time elapsed. When the requester properties, in particular service times, are the same for all requesters, PROP(h) indicates the degree of preferential throughput treatment given to requesters. The idle time of an arbiter. IDLE, is the proportion of time not allocated to any requester and is a function of request loading and efficiency of an arbiter. For a given request excitation, a less efficient arbiter will have a higher IDLE than a more efficient arbiter. Inefficiency can be due to slow arbitration logic (e.g. a long release time) or in the case of a time division multiplexed arbiter [B.1], due to unfilled time slots.

2.7 RELIABILITY OF ARBITERS

Arbiters play a central and vital role in a digital system, being responsible for resource allocation involving a large proportion of the system. The malfunction of an arbiter can cause the entire system to crash. Many possible malfunctions of an arbiter could result in system wide failure or local module failure, some of which are listed below:

(i) Multiple acknowledgements: more than one requester attempts simultaneous access to a shared resource. For example, with a system bus, information could be corrupted and bus drivers damaged with electrical noise propagating through the system.

(ii) No acknowledgements: the arbiter does not allocate the resource, despite the presence of pending requests. The system may stop running if no reconfiguration occurs. In the multiple bus arbitration example discussed in Section 2.3.1, a system bus arbiter (SBA) not issueing any acknowledges merely reduces the number of available system buses. The system does not crash, but gracefully degrades in performance [M.2]. Such a malfunction in a *module* bus arbiter would result in at least the failure of the entire module.

(iii) A module of a distributed arbiter failing: The effect is application dependent. Daisy chain arbiters are vulnerable to the daisy chain malfunctioning due to a faulty module [C.5]. Distributed dynamic priority resolution schemes employed in recent buses [C.10, T.1, T.2] are vulnerable to modules permanently driving priority lines, deadlocking the arbiter. In distributed arbiters some module faults may only affect the local module, and it is suggested [C.5, F.1, M.2] that distributed arbiters offer greater reliability over centralised designs.

After a design has been shown to be functionally correct, two aspects of unreliability can be considered. Unreliability due to hardware component and interconnection failure, and noise interference problems are classified as *hardware failures*. Another type of failure can occur in arbiters as a result of metastable behaviour within the arbiter and is called *metastable failure*. Metastable behaviour is due to storage elements of a digital circuit, such as flip-flops, bistables, latches and synchronisers, being marginally triggerred by critical request timing. The marginal request timing causes a state of indecision within the storage elements that lasts an indefinite time period. Metastable behaviour is discussed in detail in Chapter 3 where it is established that the problem is unavoidable in certain digital circuits with asynchronous inputs, arbiters being an example.

Hardware failure reliability can be improved by established techniques of hardware redundancy to mask faults and achieve a level of fault tolerance [K.1, W.1, W.2, W.4]. For example, triple modular redundancy (TMR) can be used to mask single hardware faults by employing majority voting circuits on the outputs of three replicated modules working in parallel with the same inputs. A faulty module output is masked by the other two fault free outputs that agree. The voter circuit selects the majority output value, which is the correct one when at most one faulty module exists. In order to be able to compare the outputs of the modules, they must operate with some degree of synchronism with respect to one another so that the correct outputs are available at the same time. It is the synchronisation aspect that can present problems, especially with independently clocked synchronous circuits as modules [D.1].

The author has conducted an investigation into the possibility of improving metastable reliability by applying redundancy and masking techniques. Original work presented in Chapter 3 and in [1] shows that the probability of metastable failure of a synchroniser cannot be improved by redundancy and masking techniques that are effective against hardware failure. This establishes a fundamental difference in nature between hardware failure and metastable failure. It is in the area of metastable failure that reliability performance of arbiters is analysed in this thesis.

Metastable behaviour can occur in the request synchroniser in a synchronous arbiter, for example in Figure 2.9. Should a request occur near the sampling edge of the clock, the flip-flop synchronising the request may enter a metastable state giving rise to malfunctions later in the circuit if the metastable state lasts long enough. The situation is different in an asynchronous arbiter, such as the decentralised batched fixed priority arbiter shown in Figure 2.12. Metastable behaviour can be induced during the latching of requests for a batch, should a request occur just when further requests are in the process of being locked out at the start of the servicing of a batch of requests. Within all arbiters, sequential decision logic must be present to choose the next request to service. Timing of requests can always occur that places the decision logic in a state between allocating the resource to either of two requesters. A more detailed description of the mechanisms involved is presented in the next chapter.

2.8 CONCLUSIONS

This chapter has defined an arbiter and the associated request/acknowledge protocol, with a view to isolating the essential features of an arbiter to be discussed in this thesis. The fundamental primitive nature of arbiters in digital systems has been demonstrated through application examples and discussion. Various disciplines, classifications and performance measures of arbiters have been introduced, in order to set up the framework and perspective for later work in this thesis. The service and reliability performance measures have been discussed in a qualitative manner to motivate analyses pursued later. Reliability performance has been divided into two fundamentally different classes, hardware reliability and metastable reliablity, with the latter being the subject of the next chapter.

CHAPTER 3

METASTABLE BEHAVIOUR

3.1 INTRODUCTION

Many digital circuits with asynchronous inputs, such as arbiters, are susceptible to failure even when all components are fault free. Fault free circuits may fail due to metastable behaviour when the input timing results in a marginal triggering of circuit state storage devices, such as flip-flops. Metastable behaviour of a digital circuit is a malfunction corresponding to the circuit state remaining indefinitely between two stable states, that can produce ambiguous state interpretation and other erroneous effects within the circuit.

This chapter has two main objectives. One is to provide an overview of the developments in the study of metastable behaviour relevant to arbiter design and analysis, and the other is to present original contributions in the area by the author. Although the results are directly relevant to arbiters, their applicability to digital circuits and dynamic systems in general is naturally evident throughout the chapter.

The original contributions include reformulation and generalisation of results on the unavoidability of metastable behaviour to enable practical application of the results to a wide class of circuit input functions [3].

Also original results are presented which show that redundancy and masking techniques applied to synchroniser circuits (e.g. flip-flops, latches and bistables) are ineffectual in improving the probability of metastable failure [1]. This result is significant not only in realising that employing redundancy techniques to improve the metastable reliability of synchronisers is fruitless, but also because it shows there is a

fundamental difference in characteristics between metastable reliability and hardware reliability which *can* be improved by the same redundancy techniques.

The organisation of the chapter is as follows. Specific examples of metastable behaviour are given in Section 3.2, and Section 3.3 defines metastable behaviour and describes its characteristics. Section 3.4 presents new results on the unavoidability of metastable behaviour and applies them to arbiters. The importance of metastable behaviour in systems is described in Section 3.5, where broad design issues are discussed. In Section 3.6 modelling techniques for metastable behaviour are described that are later employed in this thesis for analysing metastable failure of arbiters. Techniques for improving metastable reliability of arbiters, and indeed any digital circuit susceptible to metastable behaviour, are described and analysed in Section 3.7, where original analyses are carried out for a Schmitt trigger synchroniser and redundant synchronisers. Finally, Section 3.8 summarises the conclusions of the chapter.

3.2 EXAMPLES OF METASTABLE BEHAVIOUR

An arbiter is a digital circuit which clearly demonstrates the metastable problems involved with processing asynchronous inputs. Consider the case of a two input arbiter, implemented as an asynchronous circuit, as shown in Figure 3.1. The arbiter employs the two NAND gates on the inputs to lock out the other request input when a request occurs. The arrival order of requests is stored on the cross coupled NAND gate flip-flop whose outputs are enabled a delayed time after a request occurs.

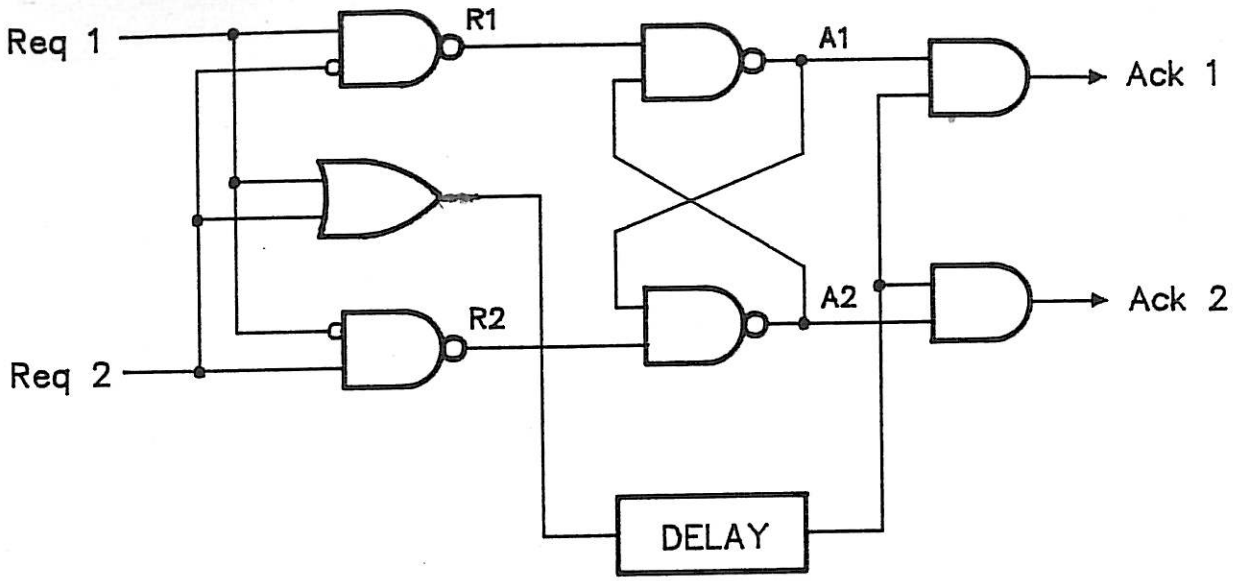


FIGURE 3.1 An Asynchronous Two Input Arbiter.

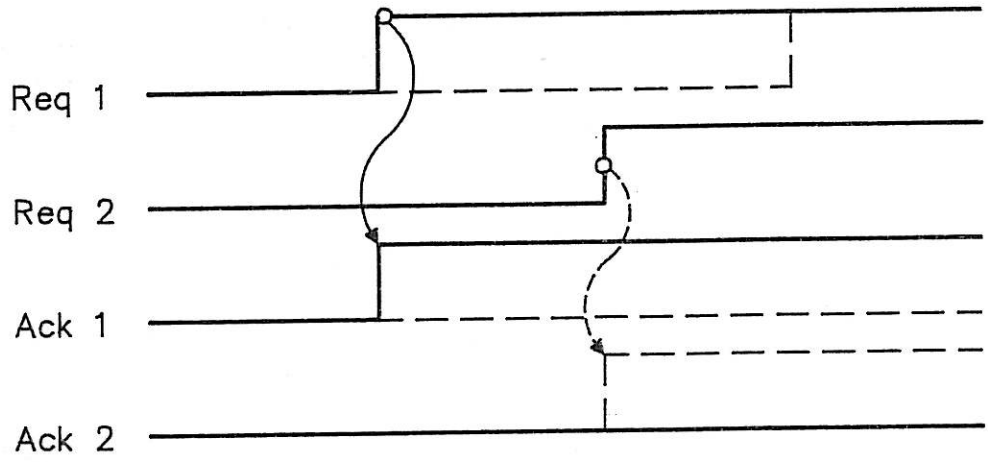


FIGURE 3.2 Timing Dependence of an Arbiter on Requests.

Consider the following two request scenarios as shown in Figure 3.2:

(i) Req 1 is asserted before Req 2, and Ack 1 shortly after.

(ii) Req 2 is asserted before Req 1, and Ack 2 shortly after.

Treating the time at which Req 2 is asserted as fixed, consider a continuous variation in timing of Req 1 from scenario (i) to (ii). At some point metastable behaviour will be induced corresponding to the arbiter being unable to decide which Ack to assert. As shown in Figure 3.3, when Req 1 and Req 2 are asserted approximately simultaneously, runt pulses are produced within the circuit at R1 and R2. The runt pulses, which are of insufficient width and separation to guarantee a valid state change, may marginally trigger the flip-flop and cause it to enter an unstable equilibrium state between the flip-flop being set and reset. The duration of the unstable equilibrium is unbounded, even in the presence of circuit noise. Ignoring noise effects for now, one may suggest that the probability of the flip-flop *exactly* reaching the unstable equilibrium point is zero and so no problem exists in practice. However, the flip-flop need not be exactly at the point of unstable equilibrium for the flip-flop voltage to stay outside the voltage ranges of the valid logic states for a period of time considerably longer than a normal propagation delay. A range of triggering energies with non zero probability can result in this anomalous flip-flop behaviour, known as a ***metastable behaviour***. It has been shown [C.16, V.1] that typical levels of circuit noise have an insignificant effect on the probability distribution of metastable behaviour duration.

The delay element in the arbiter circuit of Figure 3.1 delays the Ack outputs from being asserted to allow a metastable settling time. Since the duration of metastable behaviour is unbounded, there is still a

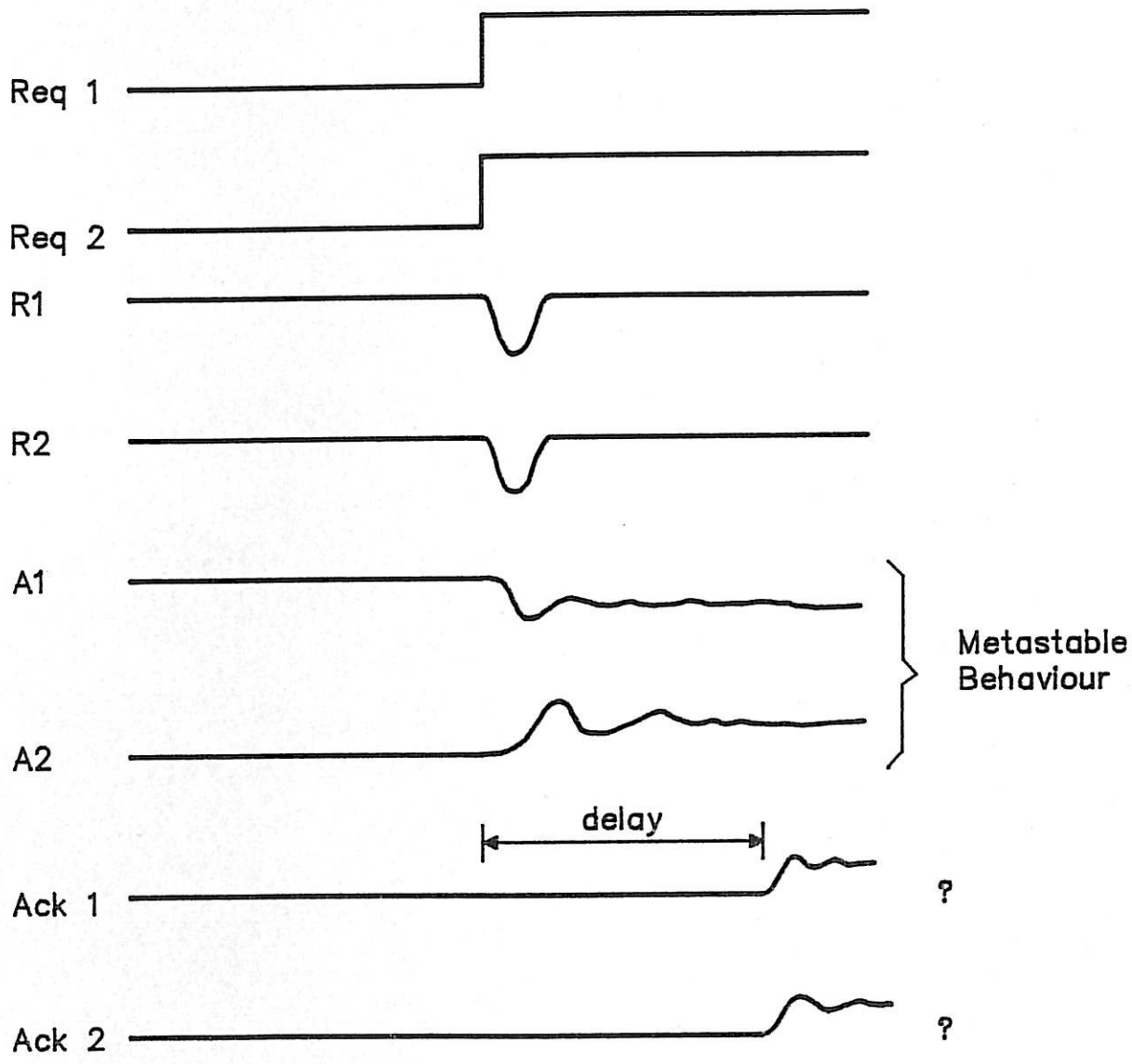


FIGURE 3.3 Arbiter Indecision and Metastable Behaviour.

possibility that the metastable behaviour of the internal flip-flop will manifest itself at the output in the form of prolonged undefined or oscillatory Ack signals, which may constitute failure of the arbiter. Techniques for extending the delay duration until metastable behaviour has settled are discussed in Section 3.7.

The arbiter circuit shown in Figure 3.1 is implemented as an asynchronous circuit. Arbiters which rely on a clock to sequence internal state changes can also exhibit metastable behaviour. Usually synchronous arbiters, for example the clocked fixed priority arbiter shown in Figure 3.4, synchronise the request inputs to the clock before the requests are utilised internally. The synchronisation is usually performed in a well defined interface circuit known as a *synchroniser*. Once synchronised, the requests are assumed to change only soon after a sampling clock edge, as shown in Figure 3.4. The synchronous circuit then can be designed so that all internal device timing constraints are satisfied [P.2]. For example, set-up and hold time constraints for a D-type flip-flop within the synchronous circuit can be met provided the clock period is sufficiently long.

Within a synchroniser metastable behaviour can occur when an asynchronous request occurs near a clock edge and violates set-up and hold time constraints of the synchroniser. When a synchroniser exhibits metastable behaviour, the entire circuit could malfunction for two reasons.

(i) The synchroniser output may no longer be a defined logic value and could be interpreted differently by different components within the synchronous circuit.

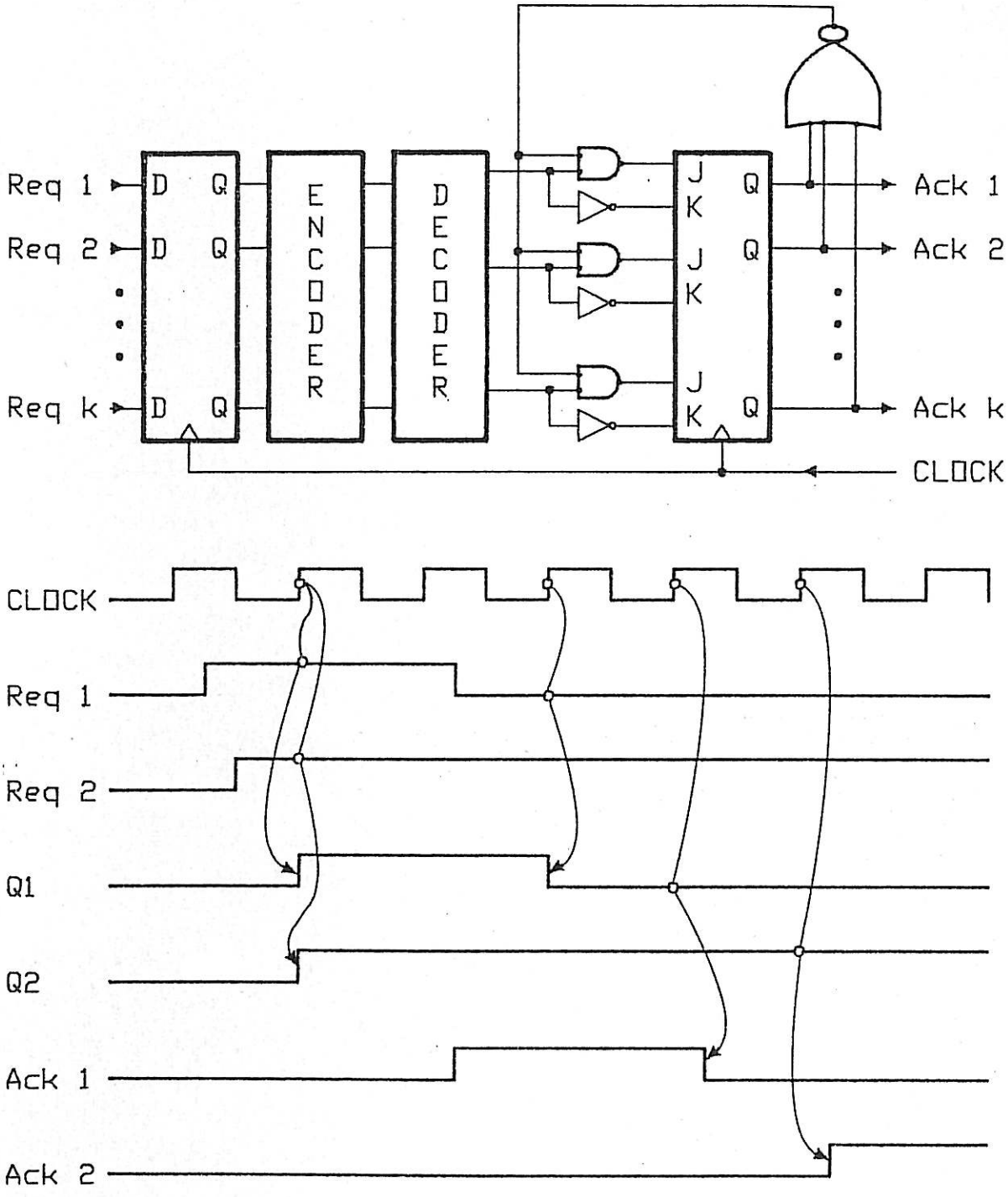


FIGURE 3.4 Clocked Fixed Priority Arbiter and Synchronous Requests.

(ii) The delay through the synchroniser can result in timing constraints of the synchronous circuit being violated.

The fundamental problem in both asynchronous and synchronous implementations of arbiters described above is that the arbiter must make a decision based upon the timing of requests which is asynchronous with respect to the other requests and possibly to an imposed clock reference. One can usually identify devices within the arbiter whose correct operation relies on constraining request timing. For example, the R-S flip-flop in Figure 3.1 requires a minimum set/reset pulse separation in order to function correctly. The D-type flip-flop synchroniser in Figure 3.4 may malfunction when requests occur within an interval of time around the sampling clock edge defined by the set-up and hold times.

Unger [U.1] has proposed a design technique for asynchronous sequential switching circuits with unrestricted input changes which uses a special state assignment and inertial delay elements in the state branches. An ideal inertial delay only passes pulses greater than a set width. All realiseable inertial delay elements require input constraints in order to perform as required. The input constraints may be violated by asynchronous inputs and metastable behaviour can occur, as demonstrated by Marino in [M.3].

3.3 METASTABLE STATE - DEFINITION AND CHARACTERISTICS

In this section, the behaviour of a flip-flop under marginal triggering conditions is considered. In Section 3.4 it is shown that similar behaviour can be exhibited by any device with at least two stable states and two input contingencies which drive the device to either stable state.

A flip-flop is said to be ***marginally triggered*** when its output fails to settle to a logically defined state ("0" or "1") within its normal maximum time delay. The normal maximum time delay is defined to be the maximum delay required for the output to reach a logically defined state under specified conditions on, for example, set-up and hold times. The acceptable maximum delay is effectively determined by the components of a logic family and the set-up and hold times appropriately specified. Marginal triggering can occur in a D-type flip-flop when the data input changes near the sampling clock edge, or in an R-S type flip-flop when reset and set pulses are released almost simultaneously or contain runt pulses.

Under marginal triggering conditions a flip-flop can enter a ***metastable state*** [C.4, C.16] which is characterised by the flip-flop state being in the vicinity of an unstable equilibrium state, with the flip-flop output lingering or oscillating in between the 0 and 1 voltage levels.

Catt [C.4] demonstrated qualitatively that an unstable equilibrium state exists in bistable flip-flops and can be induced by a critical marginal triggering of the flip-flop. Moreover, he claimed that a metastable state in a flip-flop can continue indefinitely, even in the presence of noise.

Various forms of output responses of marginally triggered flip-flops determined experimentally are presented in [C.8], where it is claimed that

for small propagation time to rise time ratio logic families the output *hovers* between 0 and 1 logic states before resolving the either logic state, whilst logic families which have large propagation time to rise time ratios exhibit *oscillatory* behaviour before resolving to either logic state.

3.4 FUNDAMENTAL UNAVOIDABILITY OF METASTABLE BEHAVIOUR

A common approach to designing sequential circuits with asynchronous inputs is to use a synchronous sequential circuit with a synchroniser as an interface to handle the asynchronous inputs. The problem of designing a perfect synchroniser (not subject to metastable behaviour) is equivalent to the problem of designing perfect circuits with asynchronous inputs, since each can be built from the other. Considerable attention has been given to the problem of designing reliable synchronisers and estimating their probability of failure due to metastable behaviour (also known as synchronisation failure) [C.4, C.7, C.8, C.16, E.1, F.2, F.4, H.3, I.1, K.3, K.5, K.6, L.1, L.2, L.3, L.5, M.3, M.4, P.3, R.1, R.2, S.6, S.7, V.1]. Attempts have been made to design a perfect synchroniser [C.6, F.3]. However, careful examination reveals that the designs are indeed subject to synchronisation failure. This is particularly evident in analyses in [C.6].

The problem of metastable behaviour is generally considered to be unavoidable in digital circuits which are dependent on the timing of asynchronous signals. Attempts have been made to establish mathematically [H.4, M.4] that metastable behaviour is a fundamentally unavoidable problem in a class of dynamic systems which includes digital systems that process asynchronous signals. Marino [M.4] adopts a general mathematical model for a digital system and then proves that metastable behaviour

occurs for a range of inputs when the inputs include all functions with bounded first derivatives that obey certain boundary conditions. That is, he requires to be included in the input function space, functions with unbounded or undefined second derivatives. Without these functions, his theorem cannot guarantee that metastable behaviour will occur.

It is questionable that inputs found in practical circuits can be considered to have unbounded or undefined second derivatives. Similar comments apply to higher derivatives. Original work of the author [3] is presented in this section to show that the inclusion of these possibly unrealistic input functions is not necessary to establish that metastable behaviour can occur. Practical input functions may have other properties which could further limit the size of the set of input functions. For example, the bandwidth may be limited or certain relationships may hold between components of the inputs. The question arises: What is a property of a general set of input functions that leads to a range of inputs causing metastable behaviour? It is shown in this section that the topological property of *connectivity* of the set of input functions answers this question. Section 3.4.1 reviews a number of mathematical preliminaries that are needed in later discussions. A theorem due to the author is then presented that extends Marino's result [M.4] followed by practical applications of the theorem to realistic input functions.

3.4.1 Mathematical Preliminaries

Central to the discussion is the definition of a connected set:

> A set C is *connected* if and only if there do *not* exist open sets D and E such that $D \cap E = \phi$, $C \cap D \neq \phi$, $C \cap E \neq \phi$ and $C \subset D \cup E$, where ϕ is the empty set.

To determine whether a set is open, a norm for the elements of the set can be used. Since sets of *functions* are considered here, the norm of a function needs to be defined. An example of a possible norm for bounded functions is the $\|\cdot\|'_\infty$ norm:

$$\| u \|'_\infty \triangleq \text{least upper bound } \{ \| u(t) \| : t \in R^+ \} \qquad (3.1)$$

where $\|\cdot\|$ is a norm on the range of u and R^+ is the set of non negative real numbers.

From a norm definition, an open sphere $S_r(u)$ centered at u can be defined:

$$S_r(u) \triangleq \{v \in X : \|v-u\|' < r\} \qquad (3.2)$$

where $(X, \|\cdot\|')$ is the normed linear space in which the sphere lies.

A point x is called an *interior* point of a set A if there exists $r > 0$, such that $S_r(x) \subset A$. The set of all interior points of A is called the interior of A, denoted $\text{int}(A)$. The set A is said to be *open* if every point of A is an interior point of A.

A point x is called a *boundary* point of A if every open sphere with centre x intersects both A and A' (=complement of A); that is, for any $r > 0$, $S_r(x) \cap A \neq \phi$ and $S_r(x) \cap A' \neq \phi$. The set of all boundary points of A is denoted $\text{bnd}(A)$.

Further mathematical background can be found in [S.5]. The notation and definitions follow [M.4] with some of the more important concepts and definitions reviewed here.

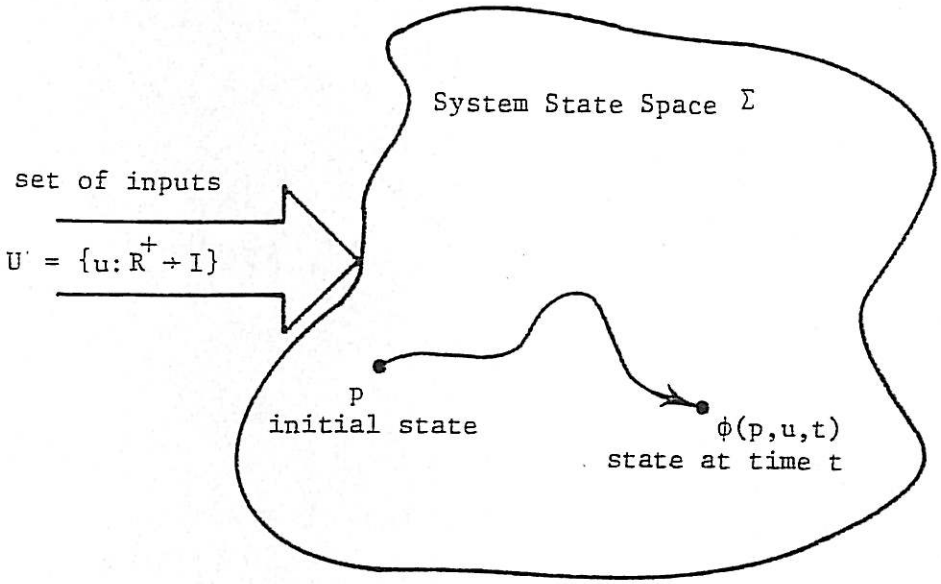


FIGURE 3.5 System Definitions.

Referring to Figure 3.5, the system model is described in terms of a state space Σ which is assumed to be a connected metric space. The set of input functions, U, consists of piecewise continuous functions $u: R^+ \to I$ where I is a metric space called the input space. The system behaviour is determined by the state transition function $\phi: \Sigma \times U \times R^+ \to \Sigma$. The value $\phi(p, u, t)$ represents the system state at time t, with input function u, and initial state p. The function ϕ is assumed to be non anticipatory (ϕ is dependent on u only up to time t)

and continuous with respect to the initial state and time. Also, ϕ satisfies

$$\phi(p, u, t+s) = \phi\left[\phi(p, u, t), u_t, s\right] \qquad (3.3)$$

for all $p \in \Sigma$, $u \in U$ and $t, s \in R^+$ where

$$u_t(s) = u(t + s). \qquad (3.4)$$

This means that all past history before t that determines future behaviour is stored in the system state $\phi(p, u, t)$, and also the system is time invariant.

Suppose the set of states L, and set of input values C satisfy $L \subset \Sigma$ and $C \subset I$. The set of idle input functions U_c is defined:

$$U_c \triangleq \{u \in U \mid u(t) \in C \text{ for all } t \in R^+\} \qquad (3.5)$$

L is *invariant for input range* C or C is the *idle range* for the invariant set L if

$$\text{for all } t \in R^+, \; p \in L \text{ and } u \in U_c \Rightarrow \phi(p, u, t) \in L \qquad (3.6)$$

The *region of attraction* (see Figure 3.6) of L for input u is

$$A(L, u) \triangleq \{p \mid \phi(p, u, t) \in L \text{ for some } t \in R^+\} \qquad (3.7)$$

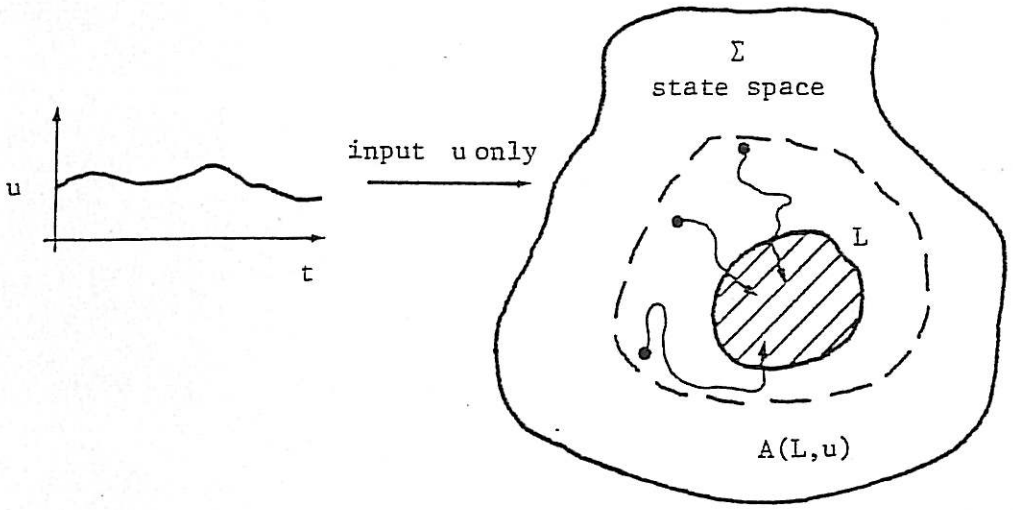


FIGURE 3.6 Region of Attraction A(L,u).

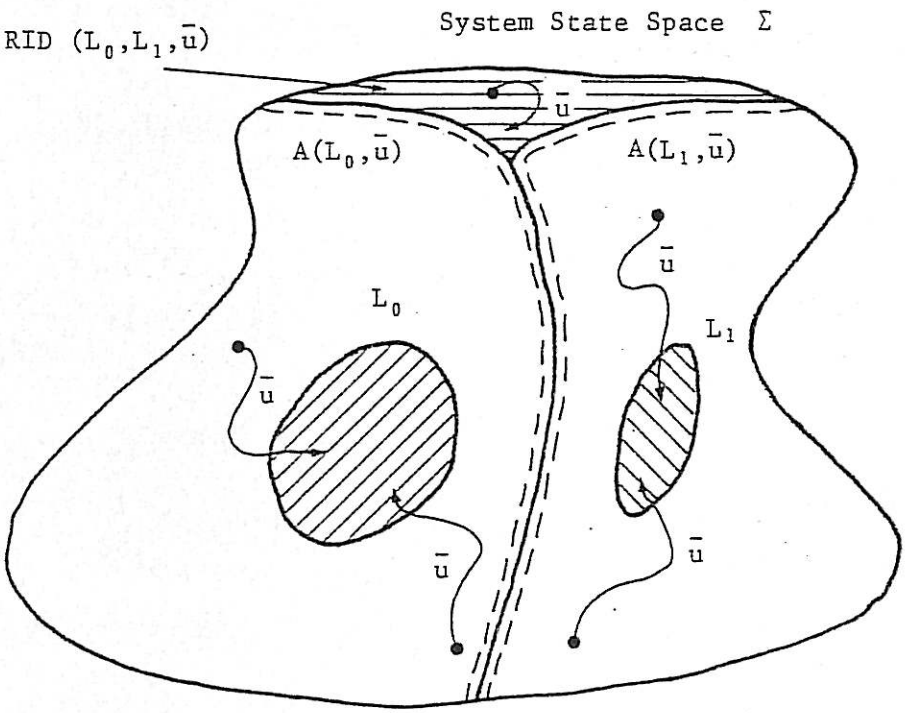


FIGURE 3.7 Region of Indecision RID(L_0,L_1,$\bar{u}$).

The region of attraction of L for input range C is

$$A(L, C) = \bigcap_{u \in U_c} A(L, u) \tag{3.8}$$

The set L is *stable* for input range C if L is invariant for C and there exist $r > 0$ such that $S_r(L) \subset A(L, C)$. Suppose that L_o and L_1 are non empty disjoint subsets of Σ, each of which is stable for input range C and suppose $\bar{u} \in U_c$. The *region of indecision* (see Figure 3.7) for $(L_o, L_1, \bar{u})$ is the set

$$RID(L_0, L_1, u) \triangleq \Sigma - \left[A(L_0, \bar{u}) \cup A(L_1, \bar{u})\right] \tag{3.9}$$

3.4.2 Statement of the Extended Theorem

Theorem 3.1 (see Figure 3.8)

Suppose $\bar{u} \in U_c$ and $p \in A(L_o, \bar{u})$. Let $\Gamma \subset U$ be a *connected* set of input functions, $u: R^+ \to I$, with the following properties:

There exists $t_1 \in R^+$, $\hat{u}_1, \hat{u}_2 \in \Gamma$ such that

(i) $\phi(p, \hat{u}_1, t_1) \in A(L_o, \bar{u})$

(ii) $\phi(p, \hat{u}_2, t_1) \notin A(L_o, \bar{u})$

(iii) for all $u \in \Gamma$, $u_{t_1} = \bar{u}$

Also, suppose ϕ is continuous with respect to u on Γ,

Then

there exists $u^* \in \Gamma$ such that $\phi(p, u^*, t_1) \in RID(L_o, L_1, \bar{u})$. Furthermore, for any $T > 0$, there exists $\epsilon > 0$ such that for $0 \leq t \leq T$

$$|u^* - u| < \epsilon \Rightarrow (p, u, t_1 + t) \notin L_0 \cup L_1 \tag{3.10}$$

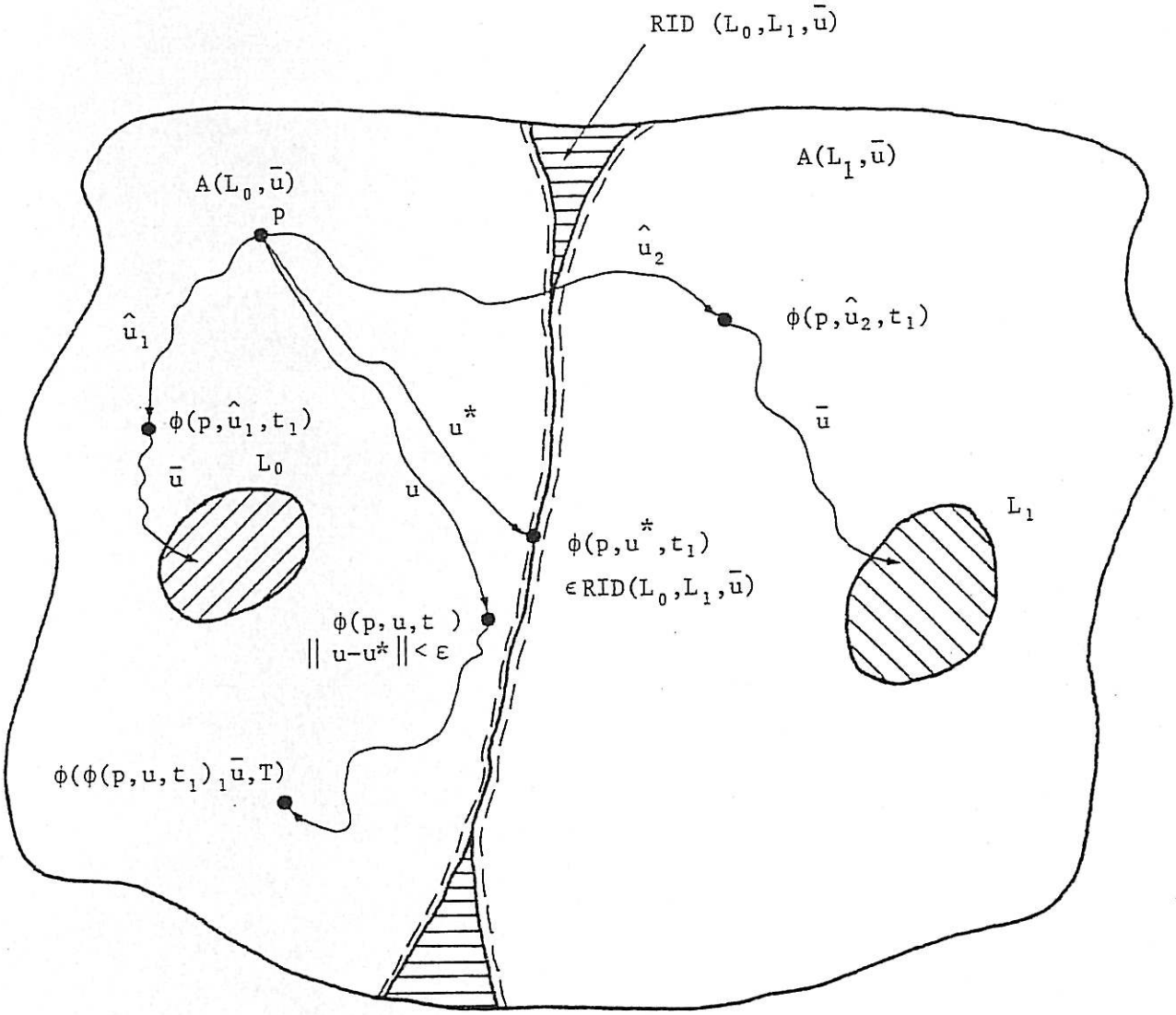


FIGURE 3.8 Illustration of Theorem 3.1.

Proof and comments

Refer to Appendix A.

3.4.3 Applications of Theorem 3.1

Theorem 3.1 can be applied to sets of inputs which are found in practical situations. For example, it is not necessary that inputs with unbounded or undefined second or higher derivatives be included in the set of possible inputs as is the case in [M.4]. The first example below demonstrates this by having an input space with constraints on an arbitrary number of derivatives.

Example 1

Let the system be a D-type flip-flop having the two stable states of "set" and "reset". The precise details of the state representation are not of concern here, only the fact that there are two stable states and the flip-flop satisfies the system axioms. The inputs to the flip-flop are the clock and D input. It is assumed that inputs take on values in the interval [0, 1], giving an input space of $I = [0, 1]^2$. Let the idle input $\bar{u}$ be the constant function (1,1). The idle range C could be defined as the disconnected set

$$C \triangleq \{(\text{clock},D) : \text{clock} \in [0, 0.2] \cup [0.8, 1],\ D \in [0, 1]\} \qquad (3.11)$$

Note that the idle functions U_c only include inputs with the clock maintaining a constant logic value, since inputs must be continuous. The set of possible input functions, Γ_n, is defined by:

$$\Gamma_n \triangleq \left\{ u = (u_1, u_2): R^+ \to [0,1]^2 \;\middle|\; i = 1 \text{ or } 2\text{: } u_i \text{ differentiable to order n;} \right.$$
$$\left| \frac{d^j u_i}{dt^j} \right| \leq B_j \,, \; j = 1, \ldots, n \,;$$
$$\left. u(0) = (0,0); \qquad u(t) = (1,1) \quad t \geq 1 \right\} \tag{3.12}$$

where $B_1 \ldots B_n$ are positive derivative bounds. It is assumed that Γ_n contains inputs $\hat{u}_1$ and $\hat{u}_2$ which set and reset the flip-flop as shown in Figure 3.9.

The set Γ_n is path connected (and hence connected [S.5]). Path connection is established by considering any two functions u^1 and u^2 in Γ_n. Define a path function $f:[0,1] \to \Gamma_n$ by $f(s) \triangleq (1 - s)u^1 + su^2$. This has the required properties $f(0) = u^1$, $f(1) = u^2$, f is continuous and $f(s) \in \Gamma$, for $s \in [0,1]$.

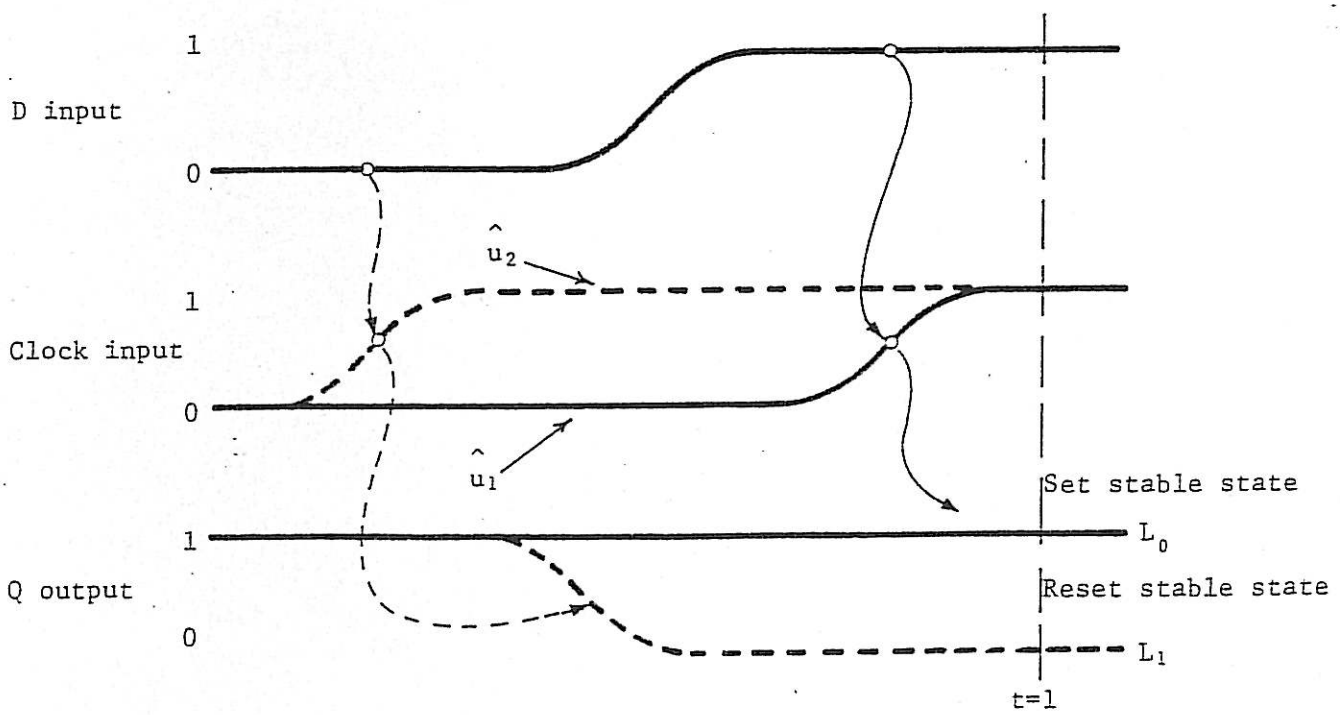


FIGURE 3.9 Examples of Inputs $\hat{u}_1$ and $\hat{u}_2$ (dashed).

Hence, Theorem 3.1 can now be applied to show that a subset of the functions Γ_n always exists that results in metastable behaviour persisting for a time T. The time T can be arbitrarily long (resulting in possibly small intervals of functions). This then shows that the inclusion of functions with unbounded or undefined second and higher derivatives is unnecessary for metastable behaviour to occur.

Example 2

This example is a two requester arbiter that is assumed to obey all the system axioms. Two stable states, L_0 and L_1, of the arbiter circuit correspond to the assertion of each Ack output. The input space is again assumed to be $[0,1]^2$. The idle input $\bar{u}$ corresponds to both Req 1 and Req 2 remaining at 1, that is $\bar{u}(t) = (1,1)$, and thus ideally the arbiter should be in either L_0 or L_1 and not $RID(L_0, L_1, \bar{u})$. The region of indecision, $RID(L_0, L_1, \bar{u})$, corresponds to the arbiter being exactly in a state of indecision forever. The set of input functions, Γ, has further conditions placed on it in this example, so that functions are monotonic and have a minimum and maximum rise time.

For fixed constants $0 < C_2 < C_1$, $0 < \epsilon < \frac{1}{2}$

$$\Gamma \triangleq \left\{ u = (u_1,u_2): R^+ \to [0,1]^2 \;\middle|\; \begin{array}{l} i = 1 \text{ or } 2\text{: } u_i \text{ differentiable;} \\ 0 \le \frac{du_i}{dt} \le C_1 \text{ ;} \\ \text{for } \epsilon \le u_i(t) \le 1 - \epsilon \text{ , } \frac{du_i}{dt} \ge C_2\text{;} \\ u(0) = (0,0) \text{ ;} \\ \text{for } t \ge 1,\ u(t) = (1,1) \end{array} \right\} \qquad (3.13)$$

Γ is assumed to contain request contingencies that result in either requester being serviced.

Γ is now shown to be path connected. The same path function employed in Example 1 cannot be employed here since the minimum rise time condition $\frac{du_i(t)}{dt} \geq C_2$ may not be satisfied for all functions on the path. This can occur when a path is required between two functions whose transition time intervals do not overlap. Instead, the path from u^1 to u^2 is broken up into three sections:

(i) u^1 to s^1

(ii) s^1 to s^2

(iii) s^2 to u^2

where s^1 and s^2 are straight line functions illustrated in Figure 3.10 and defined precisely below:

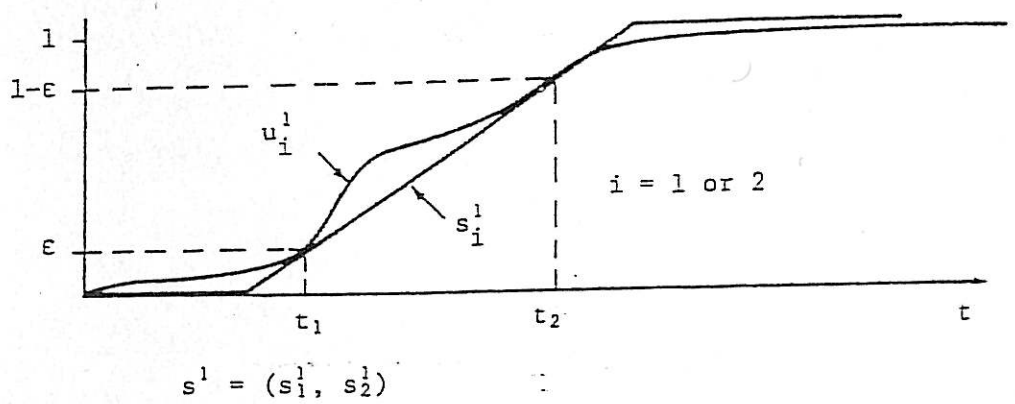


FIGURE 3.10 Definition of s_i^1 Function.

Since u_i is monotonically increasing for u_i in the interval $[\epsilon, 1 - \epsilon]$, there are unique points t_1, t_2 such that $u_i(t_1) = \epsilon$ and $u_i(t_2) = 1 - \epsilon$. The slope $\frac{1 - 2\epsilon}{t_2 - t_1}$ lies in the range $[C_2, C_1]$ thus the function $s^1 = \left[s_1^1, s_2^1\right]$ defined below is an element of Γ.

$$s_i^1(t) \triangleq \begin{cases} 0 & , \ 0 \le t \le t_1 - \dfrac{\epsilon(t_2 - t_1)}{(1 - 2\epsilon)} \\ (t-t_1)\left[\dfrac{1 - 2\epsilon}{t_2 - t_1}\right] + \epsilon & , \ t_1 - \dfrac{\epsilon(t_2 - t_1)}{(1 - 2\epsilon)} < t < t_2 + \dfrac{\epsilon(t_2 - t_1)}{(1 - 2\epsilon)} \\ 1 & , \ t \ge t_2 + \dfrac{\epsilon(t_2 - t_2)}{(1 - 2\epsilon)} \end{cases} \qquad (3.14)$$

The functions u^1 and s^1 are path connected with a path function $f(p):[0,1]\to \Gamma$ defined by:

$$f(p) \triangleq (f_1(p) \ , \ f_2(p)) \triangleq (1 - p)\, u^1 + p s^1 \qquad (3.15)$$

It is clear that $f(0)$ and $f(1)$ are u^1 and s^1 respectively. It remains to show that $f(p) \in \Gamma$ for $0 \le p \le 1$. The conditions of Γ are established in the sequence of definition in (3.13). Firstly, $f_i(p)$ (i=1 or 2) is differentiable with

$$\frac{df_i(p)}{dt} = (1 - p)\frac{du^1}{dt} + p\frac{ds^1}{dt} \qquad (3.16)$$

Now since $0 \le \dfrac{ds^1}{dt}, \dfrac{du^1}{dt} \le C_1$ then $0 \le \dfrac{df_i(p)}{dt} \le C_1$ for $0 \le p \le 1$. By the definition of s^1 , and conditions on u^1

$$\begin{array}{ll} 0 \le t < t_1 & f_i(p) < \epsilon \\ t_1 \le t \le t_2 & \epsilon \le f_i(p) \le 1-\epsilon \\ t_2 < t & 1-\epsilon < f_i(p) \end{array} \qquad (3.17)$$

It now follows that

$$\epsilon \leq f_i(p) \leq 1-\epsilon \quad \Rightarrow \quad t_1 \leq t \leq t_2$$

$$\Rightarrow \frac{du_i^1}{dt} \geq C_2 \cdot \frac{ds_i^1}{dt} = \frac{1 - 2\epsilon}{t_2 - t_1} \geq C_2$$

$$\Rightarrow \frac{df_i}{dt} \geq C_2 \tag{3.18}$$

The remaining conditions of (3.13), namely that at $t = 0$ $f_i(p) = 0$ and for $t \geq 1$ $f_i(p) = 1$, follow since they hold for both u^1 and s^1. This then completes the proof that u^1 and s^1 are path connected. By similar arguments it follows that s^2 and u^2 are path connected. All that remains is to show s^1 and s^2 are path connected. One *cannot* achieve this with the path function g:

$$g(p) \triangleq (1 - p)\, s^1 + p s^2 \tag{3.19}$$

since the condition of minimum slope may be violated along the path. Instead the following path function, $h(p) = (h_1(p), h_2(p))$ is employed, illustrated in Figure 3.11 and defined in (3.20) below.

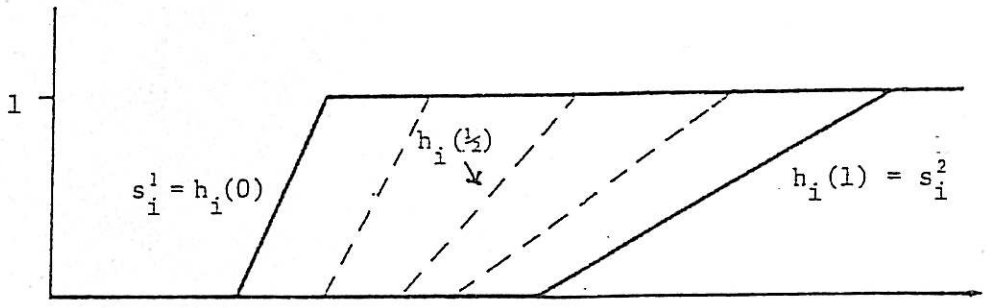


FIGURE 3.11 Definition of the Path Function h.

$$h_i(p)(t) \triangleq \begin{cases} 0 & ,0 \leq t < t_3(p) \\ \left[t - t_3(p)\right]\left[(1-p)\left[\dfrac{1-2\epsilon}{t_2^1 - t_1^1}\right] + p\left[\dfrac{1-2\epsilon}{t_2^2 - t_1^2}\right]\right] & ,t_3(p) \leq t \leq t_4(p) \\ 1 & ,t > t_4(p) \end{cases}$$

where (3.20)

$$t_3(p) = (1-p)\left[t_1^1 - \frac{\epsilon\left[t_2^1 - t_1^1\right]}{(1-2\epsilon)}\right] + p\left[t_1^2 - \frac{\epsilon\left[t_2^2 - t_1^2\right]}{(1-2\epsilon)}\right]$$

$$t_4(p) = (1-p)\left[t_2^1 + \frac{\epsilon\left[t_2^1 - t_1^1\right]}{(1-2\epsilon)}\right] + p\left[t_2^2 + \frac{\epsilon\left[t_2^2 - t_2^2\right]}{(1-2\epsilon)}\right]$$

t_1^1, t_2^1 correspond to the t_1, t_2 of s^1
t_1^2, t_2^2 correspond to the t_1, t_2, of s^2

The slope of $h_i(p)$ between ϵ and $1 - \epsilon$ lies between the slopes of s_1^1 and s_1^2 between ϵ and $1 - \epsilon$. It is easy to verify that $h(p)$ constitutes a valid path connection.

It has been shown that Γ is connected and hence there is a range of functions from Γ that produces metastable behaviour. (The two input case can be easily generalised to a k input arbiter.) That is, there is a range of request functions which result in the arbiter being in a state unable to decide between requests for every finite time allowed for settling. An arbiter cannot be designed to always correctly assert one and only one Ack output within a finite fixed time when asynchronous requests occur.

The inputs in the examples could be transformed by any continuous function (i.e. preserves connectivity) provided that two input functions

still exist that drive the system to either stable state, and metastable behaviour would be unavoidable for the transformed inputs. For example, any analog filter is a continuous transformation and could not change the connectivity of the set of inputs.

3.4.4 Conclusions

The notion of connectivity of the set of possible input functions, that includes two functions that drive the system to two stable states, is sufficient to imply metastable behaviour can occur. The modelling of inputs in terms of their connectivity is more applicable to inputs found in practice, compared with the set of functions having a bounded first derivative and necessarily including those functions with unbounded higher derivatives as presented in [M.4]. Theorem 3.1 extends Marino's work [M.4] to apply in a wider range of situations more closely matching "the real world".

The theory presented above on the unavoidability is directly relevant to physically realisable systems, since it is felt that the assumptions apply to all real systems. The most important assumption is the continuity of the system state with respect to the inputs, time and initial state. The fundamental result is that continuous systems cannot consistently make discrete decisions within a finite time based on non discrete or continuously variable inputs. Thus, a digital system, which intrinsically relies on discrete representation of information, cannot perform perfectly reliably within a finite time when it must process inputs whose timing is crucial in the decision process and yet can take on a continuum of configurations. Note that for metastable behaviour to be unavoidable, it is not sufficient just for the inputs to be asynchronous, but also there must exist two input contingencies which drive the system

to different states and a continuum of input contingencies between them. For example, a single input digital circuit which counts asynchronous occurrences of input edges separated by a minimum time, can be designed without metastable failure, since the circuit state is not dependent on the timing of the input. However, if the restriction of a minimum separation between input edges is lifted, then two edges very close together will not be seen by the circuit and two edges much further apart will count as two. In this situation metastable behaviour is unavoidable due to the continuum of edge inter-arrival times possible between no effect and two counts.

3.5 THE IMPORTANCE OF THE METASTABLE PROBLEM IN SYSTEMS

The errors caused by metastable behaviour are particularly difficult to trace due to their random and intermittent nature. They are possibly the cause of many unexplained computer crashes and other mysterious digital system malfunctions [C.8, C.9]. In the context of systems designed for high reliability, it is particularly important that careful attention be given to reducing this form of failure to an acceptable level [W.3].

It is good practice to design circuits in such a way that the possible locations of synchronisation failure can be readily identified. This can be achieved by confining the synchronisation processes to a minimum number of interfaces, so that the effects of failure can be better understood and evaluated. The ad hoc use of asynchronous inputs throughout a system causes failures to be distributed in a complex way, making design evaluation extremely difficult and error prone. Moreover, inputs may in effect be synchronised more than once in the system,

increasing the probability of failure due to metastable behaviour. Even without metastable failure, a distributed asynchronous input may be interpreted differently by different parts of the system due to clock skew effects, causing possible inconsistencies within the logic. Hence, it is essential to synchronise first before any decisions are made using the input.

An independent set of inputs should be synchronised if possible, otherwise relations assumed to hold between variables prior to synchronisation may be destroyed by the synchronisation process, and inconsistencies may occur in later processing of the inputs. For example, if two asynchronous inputs are complementary then after a marginal or near marginal synchronisation event this complementary relation may not hold in the synchronised version due to slightly different sampling times (for example) or synchroniser characteristics. The solution to this simple example is to synchronise one input only, and derive the other dependent synchronised input using an inverter. Also, it is possible to avoid unnecessary synchronisation by masking appropriated inputs with state variables when the next state transition does not depend on the variables being masked [M.7].

To achieve high reliability, it is good design practice to design circuits in which the worst case of probability of metastable failure can be estimated with confidence, otherwise another degree of unreliability is introduced, namely, the uncertainty of estimation of reliability. It is pointless to design a system that may be extremely reliable when there is no method of reasonably establishing the limits of failure, causing the system's reliability to be questionable. A design is only very reliable if the technique of evaluating the probability of failure is very reliable. Thus, it is essential to establish techniques for the reliable estimation of probability of failure due to metastable behaviour. This is the subject of the next section.

3.6 MODELLING METASTABLE FAILURE PROBABILITY

The analysis of failure probability of flip-flops used as synchronisers appears in several papers [C.16, H.3, P.3, R.1, V.1]. The basic approach is to consider the behaviour of the flip-flop around its unstable equilibrium point, where the flip-flop state is usually in its linear region. When the state leaves this region, the flip-flop is assumed to quickly resolve its output to either stable logic state, and hence the non linear behaviour can be assumed to be of little consequence to the metastable behaviour.

3.6.1 First Order Model

The first order model and analysis of a flip-flop proposed by Veendrick [V.1] (see Figure 3.12) are summarised here: The voltage $v_1(t)$ is defined to be the difference in the complementary output voltages at a time t after triggering of the flip-flop and $v_o \triangleq v_1(0)$. The insignificant decaying exponential term of the solution to the differential equations describing the first order model of the flip-flop is neglected, leaving

$$v_1(t) \cong v_o \, e^{\frac{t_c}{\tau}} \qquad (3.21)$$

The time constant τ is $RC/(A-1)$, where A is the gain of a single amplifier in Figure 3.12 and RC is the time constant of the amplifier. For marginal triggering conditions, v_o is assumed to be uniformly distributed between $-v_d$ and $+v_d$, where v_d is the "boundary" of the linear region of operation of the flip-flop. Beyond $\pm v_d$ the output is assumed to be defined as one of the two logic levels. The assumption that v_o is uniformly distributed can be justified in terms of the clock edge sampling a uniformly rising data input generating v_o [V.1].

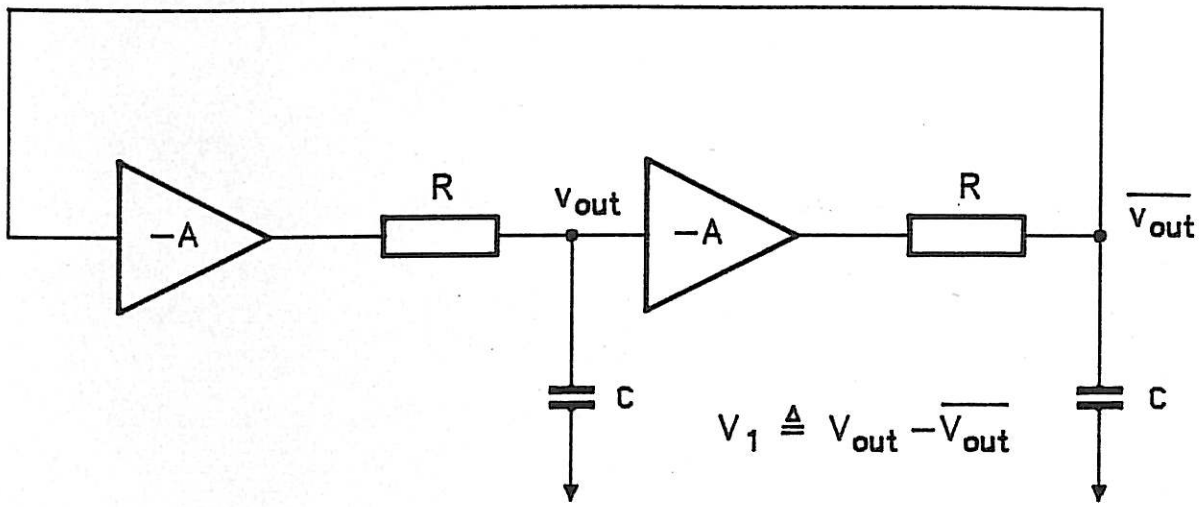


FIGURE 3.12 First Order Model of a Flip-Flop.

The probability of the metastable behaviour lasting longer than t_c is the probability that $|v_1(t_c)| < v_d$. The voltage at t_c can be mapped back to its initial voltage in a monotonic way (as shown in Figure 3.13), since $\exp(t/\tau)$ is monotonic. Thus,

$$\text{Prob}\left[|v_1(t_c)| < v_d\right] = \text{prob}\left[|v_o| < v_d\, e^{-\frac{t_c}{\tau}}\right]$$
$$= e^{-\frac{t_c}{\tau}} \qquad (3.22)$$

Since $v_1 = v_o \exp(t/\tau)$, the uniform distribution of voltages at $t = 0$ is expanded by the factor $\exp(t/\tau)$ and is also uniformly distributed. Superimposed small amplitude unbiased noise on v_1 has negligible effect on the distribution of v_1. This can be seen by considering two adjacent voltage regions. The number of states forced out of one region due to noise will be replaced by an approximately equal number forced in from the adjacent region. This qualitative argument has been verified experimentally and theoretically in [C.16, V.1].

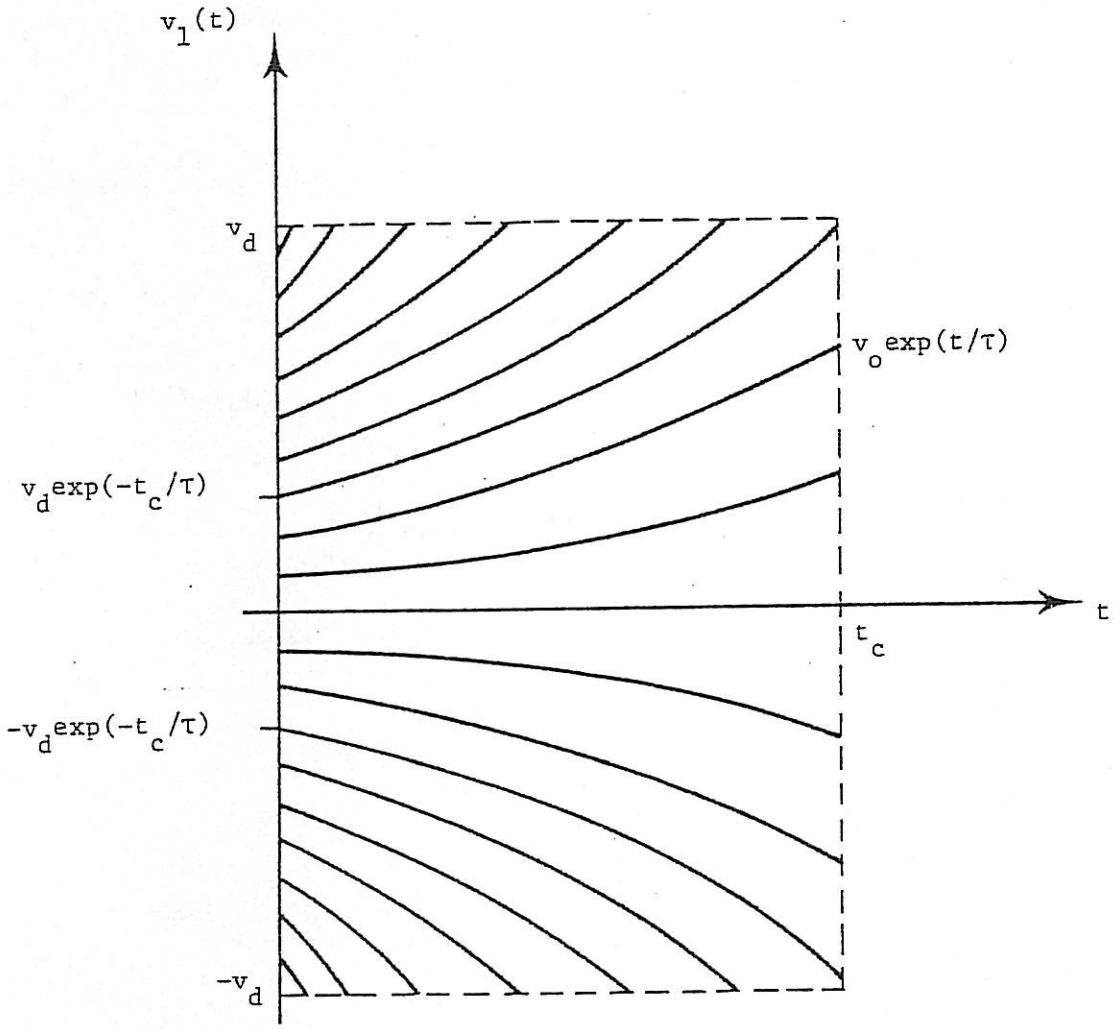


FIGURE 3.13 Exponential Flip-Flop Response.

3.6.2 Exponential Model Based on Experimental Results

Experimental measurements made on flip-flops [C.7, F.2, L.1, V.1] have led to the verification of the exponential model for predicting the failure rate due to metastable behaviour. A model for the probability of failure is presented in [C.7, R.1]. It is described here in terms of a D type flip-flop, but can be generalised to any flip-flop.

Suppose the time between a data edge and a sampling clock edge is t_d, and has a uniform probability density, $p(t_d)$ over an interval $[t_a, t_b]$. Then,

$$p(t_d) = \begin{cases} 0 & t_d < t_a \\ \frac{1}{\delta} & t_a \leq t_d \leq t_b \\ 0 & t_d > t_b \end{cases} \qquad (3.23)$$

where $\delta = t_b - t_a$ and the flip-flop is set by $t_d = t_a$ and reset by $t_d = t_b$ within a normal propagation delay. The times t_a and t_b could be chosen to correspond to the set-up and hold times respectively for the flip-flop. For t_d uniformly distributed over this interval, F(T) is defined as the probability that the flip-flop output has not reached a logically defined and stable value at a time T after triggering. Experiments have shown that for $T > h$ (h defined below) this probability can be represented as

$$F(T) = \left(\frac{T_o}{\delta}\right) e^{-\frac{T}{\tau}} \quad , \quad T > h \qquad (3.24)$$

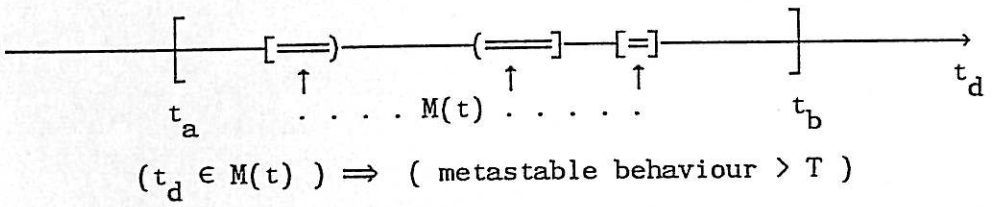
The parameters τ and T_o depend on the particular flip-flop. The parameter h is the minimum value of settling time so that (3.24) fits experimental data. Various values for the parameters of (3.24) are given

in [C.7] (where TAU is equivalent to τ) for available flip-flops. It is observed from [C.7] that there is a large variation in τ and T_o within the same flip-flop type, giving many orders of magnitude difference in the calculated examples of mean time between the synchroniser being unresolved (MTBSU). This dramatic range of performance is particularly disturbing in relation to designing below a particular error rate, and thus designs should be very conservative to guarantee a minimum metastable reliability. In time critical or high reliability applications, individual testing may be worthwhile to achieve high performance. A practical way of achieving individual testing is to incorporate testing circuitry on the same chip as the synchroniser, so that synchronisers can be tested easily. The test circuitry could be made semi-automated so that fast and simple determination of a flip-flop's suitability could be achieved at the fabrication stage of a circuit. A design similar to that proposed in [R.2] could be employed for the on-chip test circuit.

3.6.3 Aperture Model For a Flip-flop and Calculation of a Synchroniser Failure Rate

With the first order model for a D type flip-flop and a uniform distribution of the relative data to clock time, t_d, an exponential relationship is obtained for the probability of the flip-flop not resolving to a logic state within time T. In practice, with a uniform distribution for the exciting signal, good agreement is obtained in the probabilities. This is not to say that from a deterministic viewpoint the set of values of t_d that cause metastable behaviour lasting longer than T, $M(T)$, form a contiguous subset of $[t_a, t_b]$ as is suggested by the analysis of the first order model. It is conceivable that because of internal flip-flop characteristics, the set $M(T)$ is a non contiguous subset of

$[t_a, t_b]$, as shown in the example of Figure 3.14. Nevertheless, provided the probability density function of t_d is constant over $[t_a, t_b]$ one can model $M(T)$ as a contiguous subset of $[t_a, t_b]$ known as an *aperture* as shown in Figure 3.15. In view of the smallness of $[t_a, t_b]$ relative to system delays and clock periods, the uniform distribution is a good approximation for most distributions over $[t_a, t_b]$. The example of an exponential distribution is treated below:



$(t_d \in M(t)) \Rightarrow$ (metastable behaviour > T)

FIGURE 3.14 Example of a Non Contiguous M(t).

The data edge arrival process is assumed to be a Poisson process [G.2, K.12]. That is given any interval of time $[t_1, t_2]$

$$\text{prob(no data edge in } [t_1, t_2] \text{)} = e^{-\lambda(t_2 - t_1)} \qquad (3.25)$$

where λ is the mean rate of edges. A D type flip-flop with a constant clock period of T_c is employed as a synchroniser. Failure of the synchroniser is assumed to occur if the output does not reach a valid logic state a time T after the triggering clock edge. Focusing on one clock period, let p_{nf} be the probability of the flip-flop not failing.

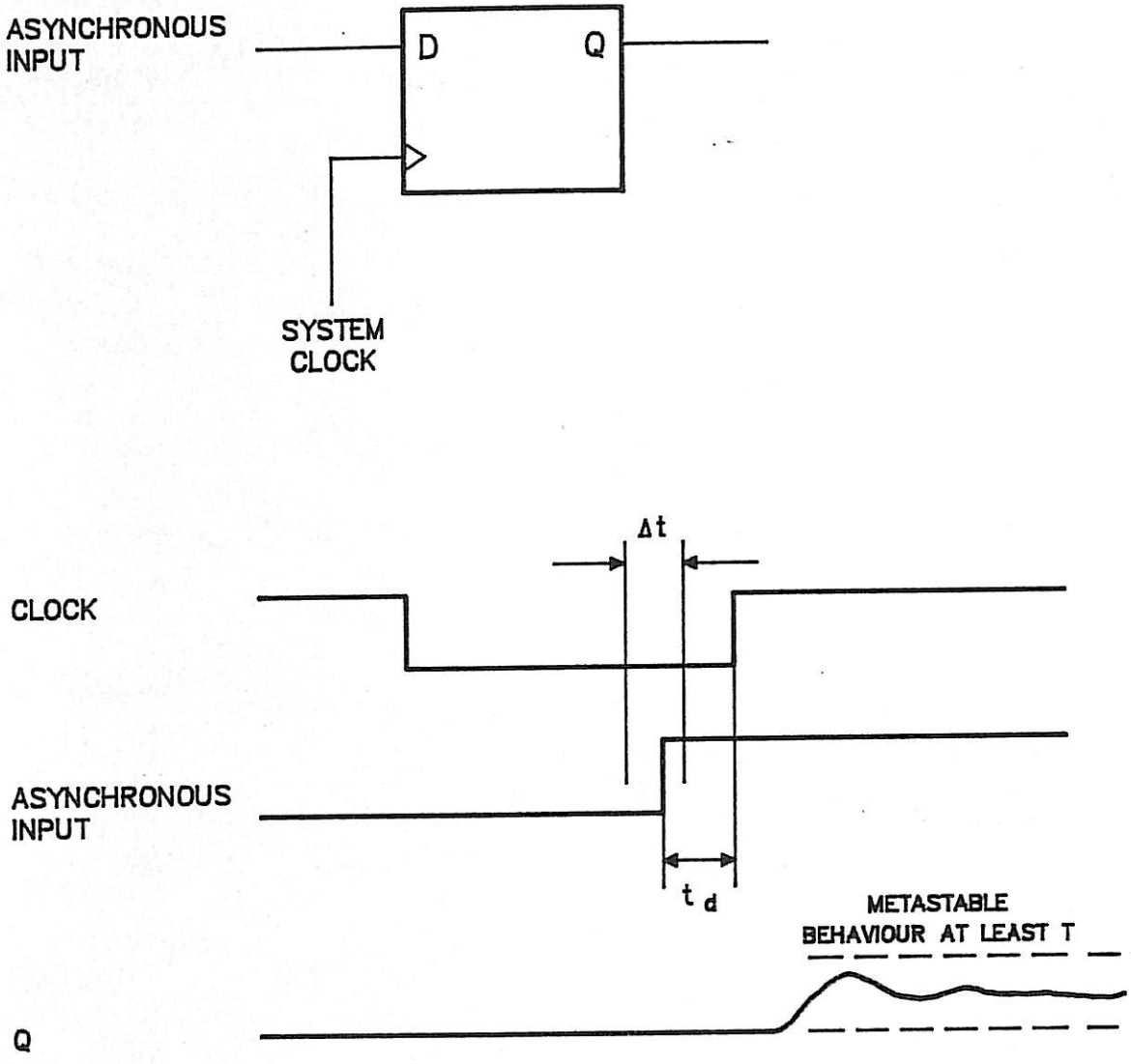


FIGURE 3.15 Aperture Model for a Flip-Flop.

For the empirical model based on (3.24) and assuming $\delta = t_b - t_a < T_c$

$$P_{nf} = \text{prob(no data edge in } [t_a, t_b]) +$$
$$+ \text{prob(data edge in } [t_a, t_b]) \times \text{prob(no failure} \mid \text{ data edge in } [t_a, t_b])$$

$$\cong \quad e^{-\lambda\delta} \quad + \quad (1 - e^{-\lambda\delta})(1 - \frac{T_o}{\delta} e^{-\frac{T}{\tau}})$$

$$= \quad 1 \quad + \quad \frac{T_o}{\delta} e^{-\frac{T}{\tau}} (e^{-\lambda\delta} - 1)$$

$$= \quad 1 - \lambda\,\Delta t \quad + \quad O(\lambda\,\Delta t)^2$$

$$= \quad e^{-\lambda\,\Delta t} \quad + \quad O(\lambda\,\Delta t)^2$$

$$\cong \text{ prob(no data edges within an aperture of width } \Delta t) \qquad (3.26)$$

where $O(\lambda\,\Delta t)^2$ denotes an error of order $(\lambda\,\Delta t)^2$ and

$$\Delta t \triangleq T_o\, e^{-\frac{T}{\tau}} \qquad (3.27)$$

The exponential distribution is assumed to be approximately a uniform distribution over the time interval $[t_a, t_b]$ which is assumed to be small compared with the mean time between data edges $\frac{1}{\lambda}$, in order to apply (3.24) in (3.26) above. The *aperture width*, Δt, is typically of the order of picoseconds for logic families with nanosecond order delays and so $\lambda\,\Delta t \ll 1$ is a reasonable assumption enabling the error terms in (3.26) to be ignored. Thus it has been shown that the probability of failure of the synchroniser with a Poisson data edge process can be modelled by the aperture model since it gives a good approximation to P_{nf}. This result is generalisable to any input process that can be approximated

by a uniform distribution over $[t_a, t_b]$. The failure rate of the synchroniser is now calculated: Assuming p_{nf} is independent for each clock period, it follows that

$$\text{prob(no failure in [0, t])} \cong \left[e^{-\lambda \Delta t}\right]^{\frac{t}{T_c}} \qquad (3.28)$$

where the approximation is due to t not always being an exact multiple of T_c. For $t >> T_c$ this can be neglected. Letting the clock frequency $(= \frac{1}{T_c})$ be f_c, then

$$\text{prob(no failure in [0, t])} = e^{-(\lambda \Delta t f_c)\, t} \qquad (3.29)$$

Equation (3.29) shows that the occurrence of metastable failure is another Poisson process (i.e. inter-arrival times of failures are exponentially distributed) with mean failure rate, MFR

$$\text{MFR} = \lambda \Delta t f_c \qquad (3.30)$$

Note the exponential dependence of Δt on T and τ in (3.27). This explains the enormous variation possible in the MFR even for modest variations in τ or T.

3.7 TECHNIQUES FOR IMPROVING METASTABLE RELIABILITY

This section discusses techniques that may improve metastable reliability of synchronisers and compares the merits of the schemes. The techniques are equally relevant to flip-flops, latches, interlocks, inertial delays and other devices prone to metastable behaviour.

No previous work has appeared that analyses the Schmitt synchroniser presented in Section 3.7.4 nor the possibility of applying redundancy and masking techniques to the problem of metastable behaviour in synchronisers considered in Section 3.7.5.

3.7.1 Fast Devices

Work has been done to design flip-flops with small values of τ using tunnel diodes [E.1, K.3, L.2] to increase the speed of the device. The use of fast logic for synchronisers, when the remainder of the system logic is based on slower logic, represents a solution to the problem of achieving high reliability for synchronisation. However, when systems are built for maximum speed, the system is likely to be realised with the same fast logic technology in all parts of the system and the time scaled down accordingly. Thus, the advantage of the previous high ratio of settling time to the device time constant τ is no longer present and alternative strategies must be sought to achieve high metastable reliability.

The criterion of a device being fast does not always correspond to a low probability of metastable failure. For example, a device may have a fast switching time, yet exhibit extended oscillatory metastable behaviour due to underdamping in the linear region of operation of the device. Test results similar to those obtained in [C.7] should be obtained before confidence is placed in a device's synchronisation performance. Furthermore, Pechoucek [P.3] suggests that improved speed does not necessarily result in reduced metastable durations.

3.7.2 Extended Settling Time

A simple technique to achieve a certain metastable reliability is to allow adequate settling time after each synchronisation event [M.7]. As shown in Section 3.6.3, dramatic (exponential) improvements in metastable reliability can be gained by increasing the ratio of settling time to device time constant, τ.

For example, in a synchronous system this can be achieved, as in Figure 3.16 by cascading N+1 flip-flops to obtain a delay of N clock periods before a sampled input is seen by the rest of the system. During the N clock periods, a marginal input value has the opportunity to resolve to either valid logic value as it is shifted through the synchroniser flip-flops.

An alternative scheme that also allows approximately N clock periods settling time is shown in Figure 3.17. The system clock is divided by (N-1) to achieve an uninterrupted settling time of (N-1) clock periods between FF1 and FF2, and a further clock period between FF2 and FF3. The function of FF3 is to eliminate the delay of the divider circuit d from the final synchronised input available to the system. The performance of the two schemes is analysed by the author in [2] where it is concluded that the divided clock scheme is superior when several clock periods settling time are needed and many asynchronous inputs are synchronised.

3.7.3 Pausable Clock and Metastable Detection Techniques

A scheme suggested in the literature involves the use of metastable detectors and a pausable clock or extendable settling time [F.4, K.3, L.5, P.3, S.6, S.7]. In an asynchronous circuit the settling time of a flip-flop is extended when a metastable state is detected. In a

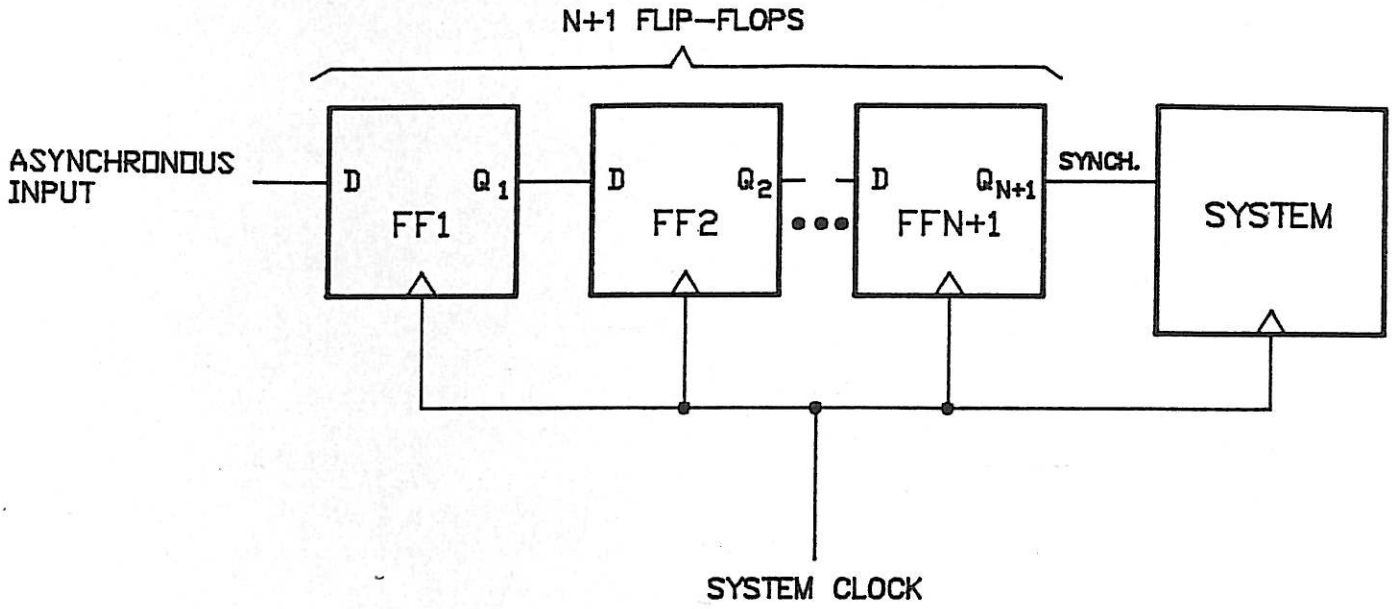


FIGURE 3.16 N+1 Cascaded Flip-Flop Synchroniser.

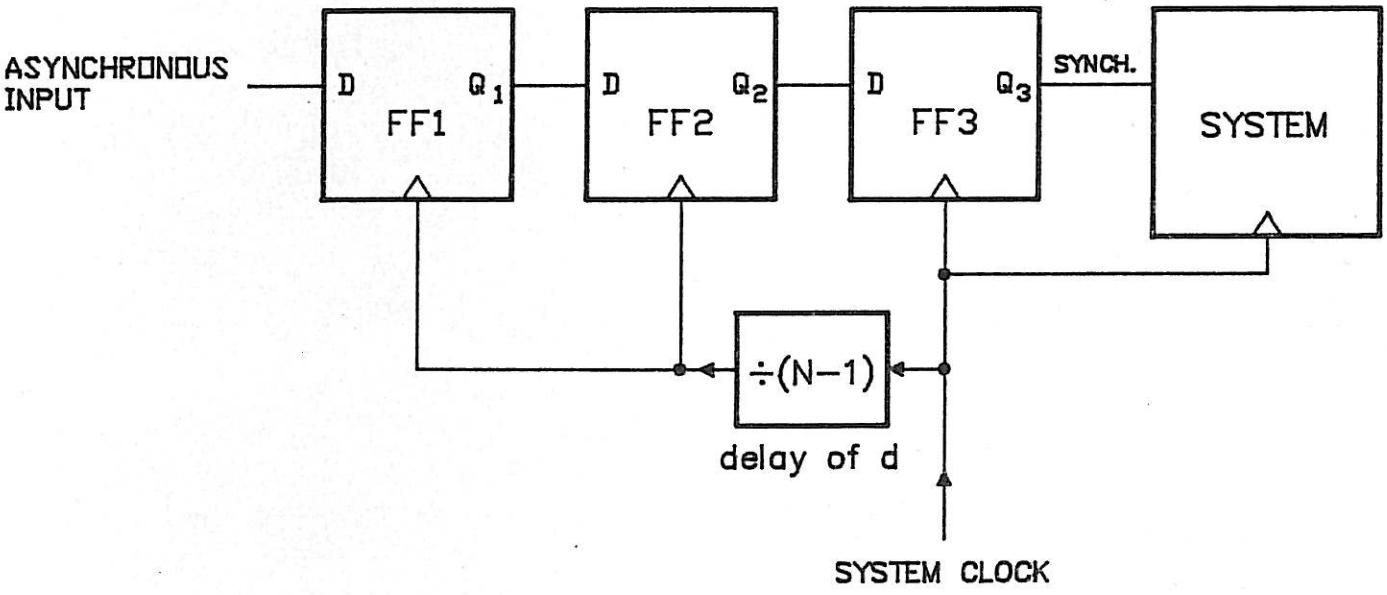


FIGURE 3.17 Divide by N-1 Synchroniser.

synchronous circuit the system clock is pausable as shown in Figure 3.18. Each flip-flop is assumed to have an extra output M which is asserted during a metastable state in the flip-flop delaying the next clock event until the flip-flop has settled. This can be thought of as synchronising the system to the input. It has been suggested [F.4, P.3] that a metastable detector can be implemented using a level detection device. Figure 3.19 shows the author's design of a pausable clock. This circuit pauses and restarts the clock without runt pulses provided the delay in detecting a metastable state after a sampling clock edge, t_m satisfies

$$t_m + t_{set\text{-}up} \leq T_2 \qquad (3.31)$$

where $t_{set\text{-}up}$ is the set-up time required by FFp and T_2, is the positive clock pulse width. At most one positive edge on the pause input is allowed to occur after each synchronisation.

The pausable clock scheme has several disadvantages. Firstly, some systems rely on a constant clock rate for correct operation and cannot utilise this scheme. Also, it is questionable whether reliable metastable detectors can be designed which are not subject to runt pulses on the output when oscillatory metastable behaviour occurs. For example, a simple threshold device may fail under oscillatory metastable behaviour, examples of which can be found in [C.8]. Even for non oscillatory metastable behaviour, the reduced noise margin caused by metastable detectors may be a source of unreliability.

The problem of spurious outputs from the metastable detector is particularly serious because the performance of the pausable system clock is directly dependent on the metastable detector outputs. A failure of the system clock is felt to be more serious than a single synchroniser

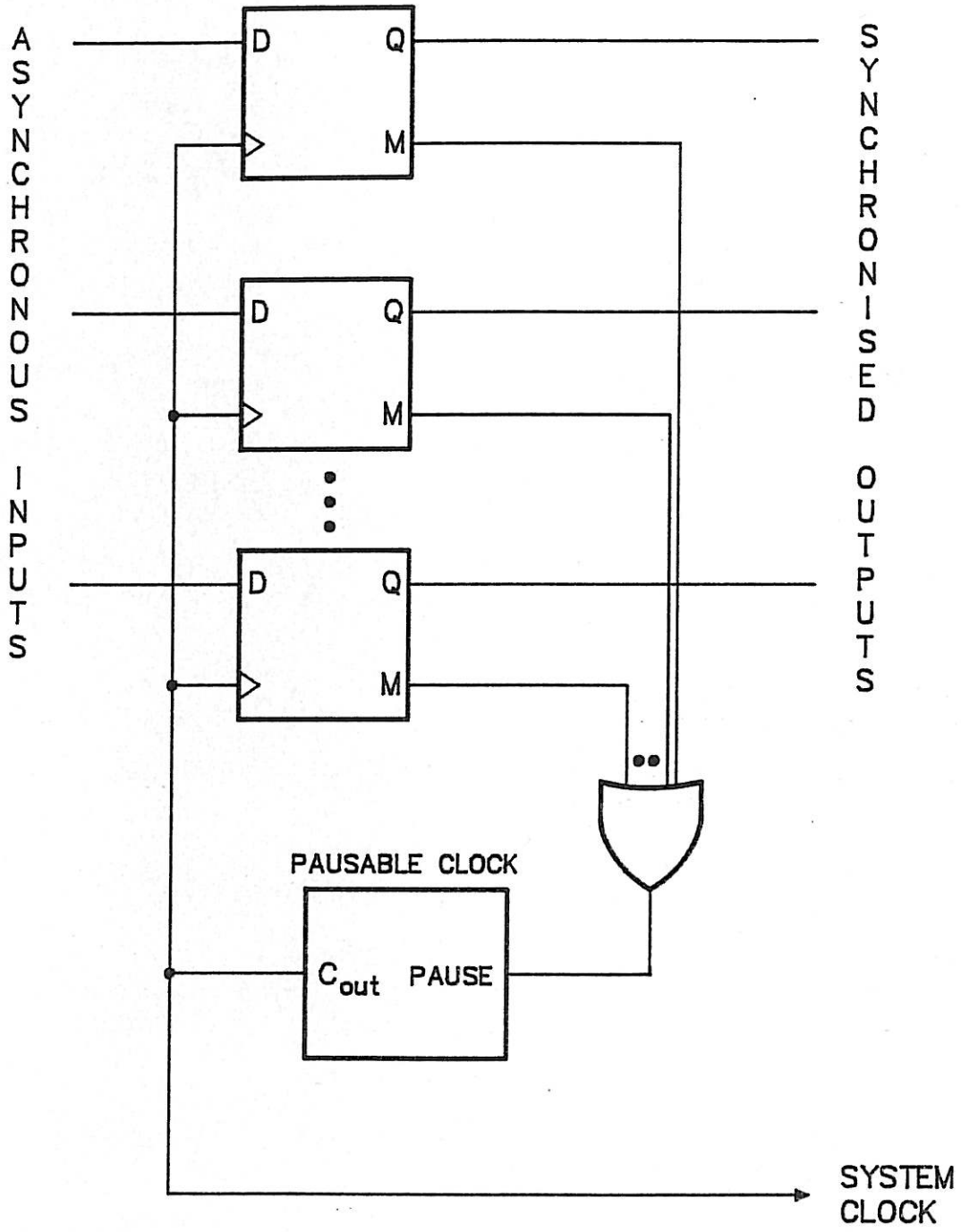


FIGURE 3.18 Pausable Clock Synchroniser.

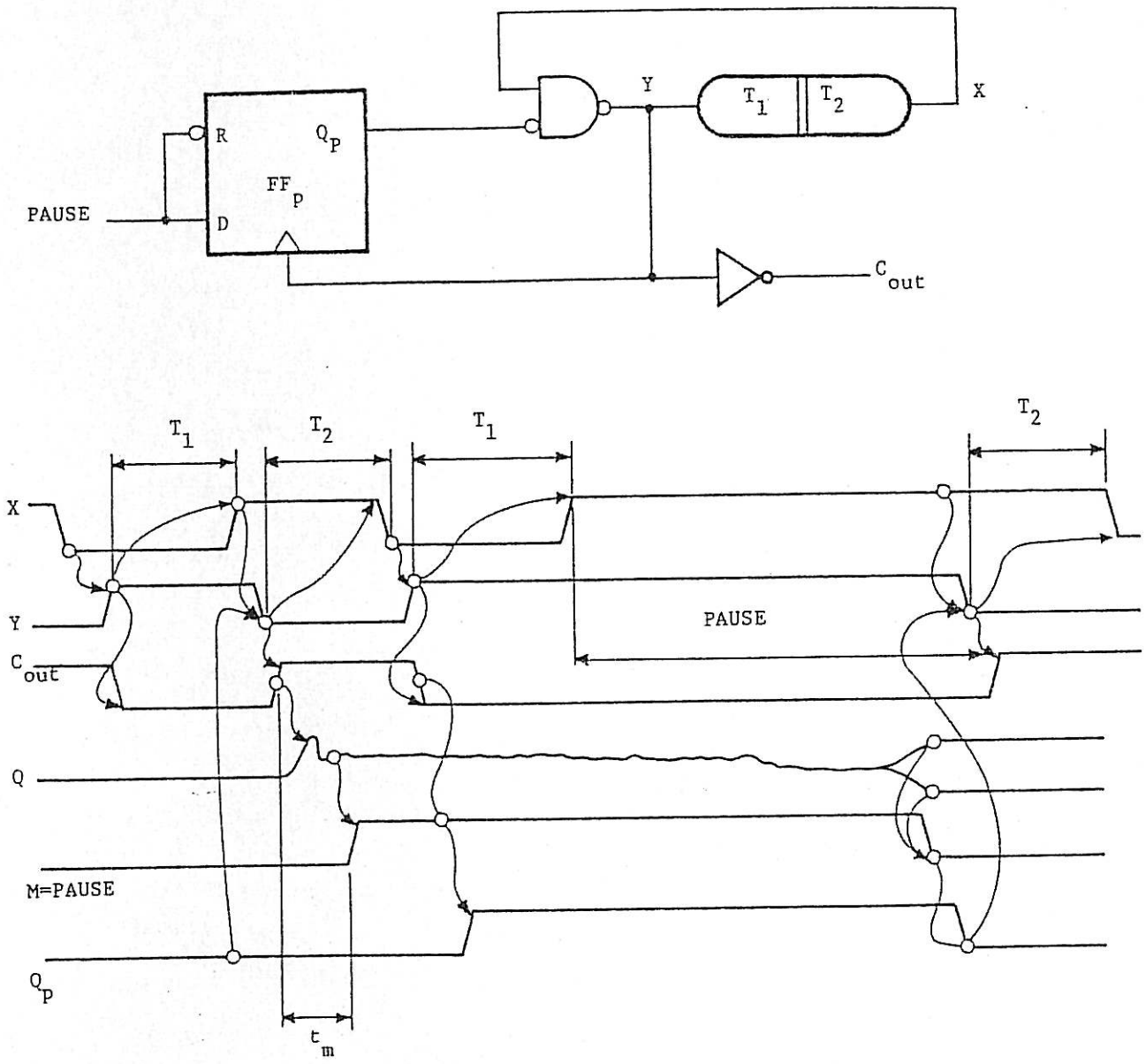


FIGURE 3.19 Pausable Clock Design.

failure because the effects of a system clock failure are more widespread. The time to complete a processing task using a pausable clock scheme is indeterminantly extended, and failure can be defined in terms of a task finishing beyond a time limit. Comparisons with fixed clocks schemes have been made [L.5] that suggest the pausable clock scheme is more reliable under certain conditions which implicitly assume that problems of pausable system clock reliability can be overcome.

In asynchronous circuits where no clock is present, the settling time given to a flip-flop (or other device) is usually determined by delays within the circuit. These delays may be designed to be variable in response to a metastable detector if such a detector could be reliably implemented. These designs are not considered further in this thesis due to the problems concerned with reliable metastable detectors.

3.7.4 Schmitt Trigger Synchroniser

It has been suggested that a Schmitt trigger can be employed to "filter" metastable behaviour and solve the problem of metastable behaviour [C.6]. In this section an analysis due to the author of two synchroniser circuits is presented to compare the probability of metastable failure in each circuit. The circuits are shown in Figure 3.20 and Figure 3.21. Figure 3.20 represents the simpler synchroniser, consisting of two D-type flip-flops. This circuit allows one clock period for the output of FF1 to settle before being sampled by FF2 to form the output of the synchroniser. The Schmitt synchroniser in Figure 3.21 attempts to improve the performance of the simple synchroniser by incorporating a Schmitt trigger on the output of FF1 with the aim of "filtering" a possible metastable state in FF1 before the metastable state can affect FF2. The metastable behaviour of a flip-flop is

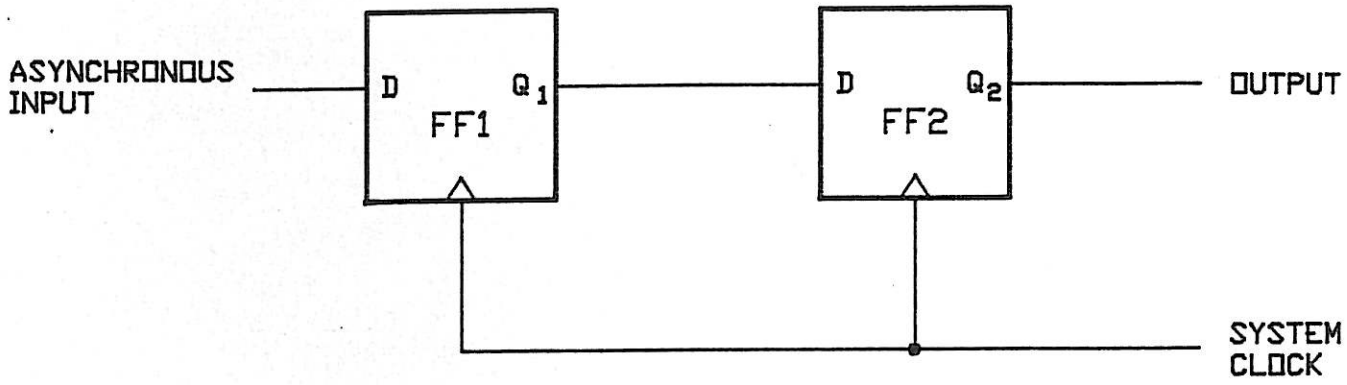


FIGURE 3.20 Simple Synchroniser.

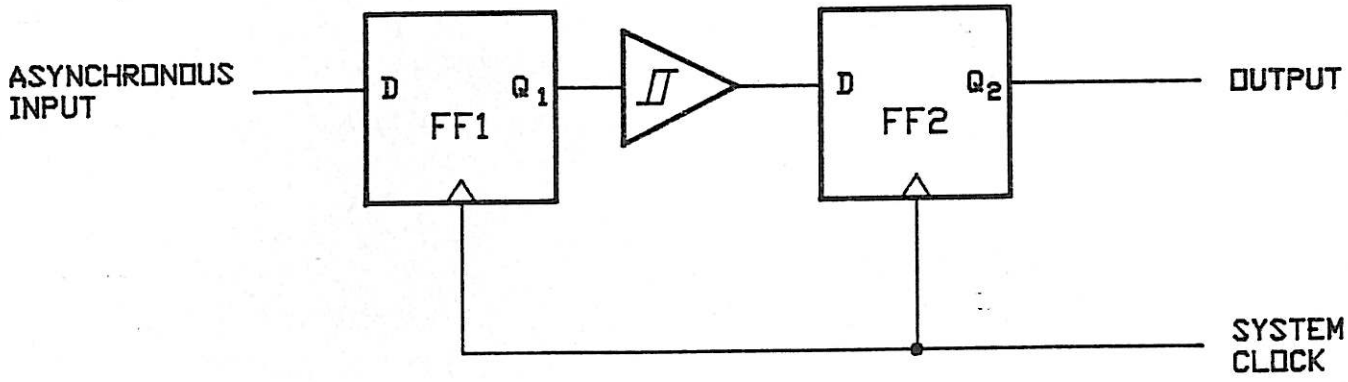


FIGURE 3.21 Schmitt Synchroniser.

modelled using the first order model discussed in Section 3.6.1. Associated with each synchroniser circuit there is a range of values of v_o (the initial voltage sampled on the positive clock edge, $v_o = 0$ corresponds to the unstable equilibrium point between 0 and 1 logic levels) that gives rise to failure of the synchroniser. Failure is defined to occur when FF2 of each synchroniser samples a voltage between $-v_d$ and $+v_d$.

By comparing the ranges of v_o that give rise to failure in the synchronisers, the relative probabilities of failure can be determined when the asynchronous input has the same stochastic properties for each synchroniser.

Firstly, the range of v_o that gives rise to failure is determined for the simple synchroniser. A failure occurs if the output voltage $v(t)$ of FF1 satisfies the following relation

$$-v_d < v(T) < v_d \qquad (3.32)$$

where T is the clock period. Substituting $v(T)$ for v_1 using equation (3.21) gives

$$-v_d\, e^{-\frac{T}{\tau}} < v_o < v_d\, e^{-\frac{T}{\tau}} \qquad (3.33)$$

From (3.33) the size of the range of values of v_o for the simple synchroniser R_1 that give rise to failure is

$$R_1 = 2v_d\, e^{-\frac{T}{\tau}} \qquad (3.34)$$

For the analysis of the Schmitt synchroniser, it is necessary to adopt a model for the behaviour of the Schmitt trigger. Figure 3.22 illustrates the model adopted. The two thresholds of the Schmitt trigger are assumed to be at $+v_d$ and $-v_d$. That is, if the Schmitt trigger has an output of logic 0, then the threshold is v_d whilst if the output of logic 1, then $-v_d$ is the threshold. Once the input passes through a threshold, the output changes accordingly but delayed by t_s and with a rise time of t_r as shown in Figure 3.22. The negative edge behaviour is assumed to be similar. The first order approximation assumed for the flip-flops results in monotonic inputs to the Schmitt trigger, hence, issues relating to marginal triggering of the Schmitt trigger [M.3] are avoided. The effect of non monotonic inputs is discussed at the end of this section.

The Schmitt synchroniser is assumed to fail when the output of FF1, $v(t)$ passes through the Schmitt trigger thresholds v_d and $-v_d$ at a time which causes the Schmitt trigger output to be in transition at $t = T$. This is illustrated by the timing diagram in Figure 3.23 for positive edge transitions.

From Figure 3.23 it can be seen that failure occurs on a positive edge output if

$$v(t) = v_d \quad \text{for} \quad T - T_s < t < T - t_s + t_r \tag{3.35}$$

Substituting for $v(t)$ from (3.21), and rearranging (3.35) gives

$$v_d\, e^{-\frac{T - t_s + t_r}{\tau}} < v_o < v_d\, e^{-\frac{T - t_s}{\tau}} \tag{3.36}$$

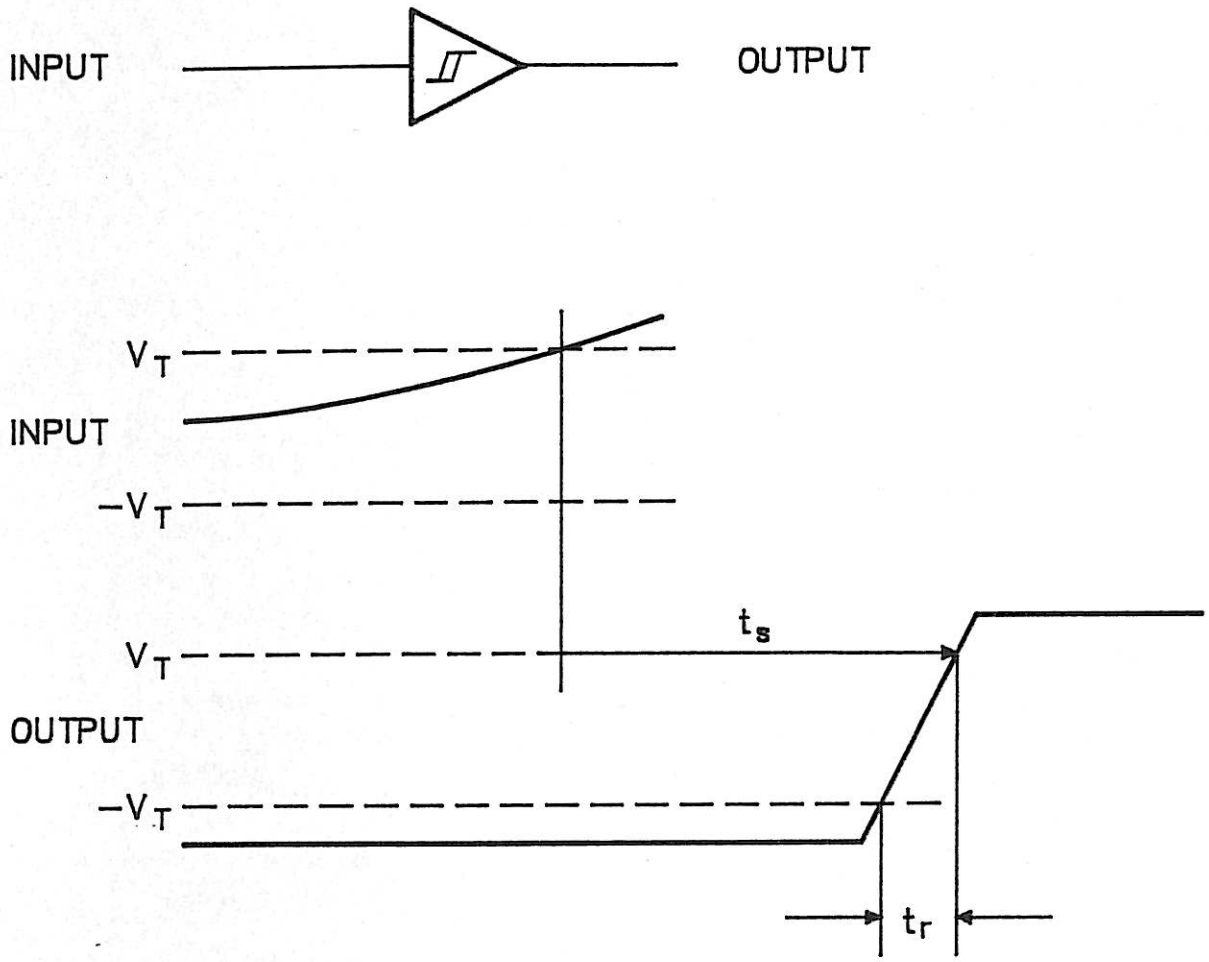


FIGURE 3.22 Schmitt Trigger Model.

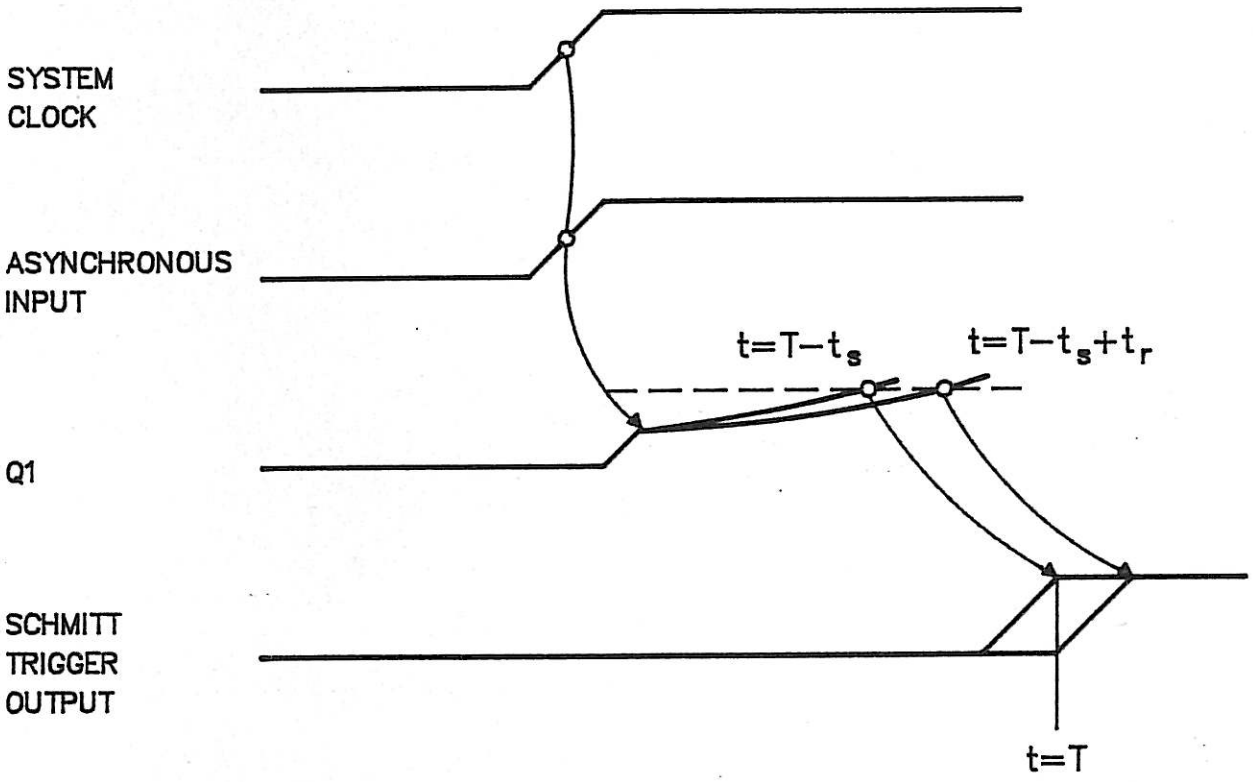


FIGURE 3.23 Failure of Schmitt Synchroniser.

Similarly for the negative edge, the range of values of v_o that give rise to failure is given by

$$-v_d\, e^{-\frac{T - t_s}{\tau}} < v_o < v_d\, e^{-\frac{T - t_s + t_r}{\tau}} \qquad (3.37)$$

From (3.36) and (3.37) the sum of the size of the two intervals of v_o that result in failure of the Schmitt synchroniser R_2 is given by

$$R_2 = 2v_d\, e^{-\frac{T - t_s}{\tau}} \left\{ 1 - e^{-\frac{t_r}{\tau}} \right\} \qquad (3.38)$$

With the assumption that the values of v_o are uniformly distributed between $-v_d$ and $+v_d$ under marginal triggering conditions R_2/R_1 represents the ratio of probability of failure of the Schmitt synchroniser to the simple synchroniser. Note that the probability of marginal triggering in both synchronisers is equal since the asynchronous input is the same stochastic process in both cases.

Comparing the two synchronisers, from (3.34) and (3.38) it follows that

$$\frac{R_2}{R_1} = e^{\frac{t_s}{\tau}} \left\{ 1 - e^{-\frac{t_r}{\tau}} \right\} \qquad (3.39)$$

Assuming the flip-flop FF1, and the Schmitt trigger are in the same logic family then $(t_s/\tau) >> 1$. This follows since the propagation delay of a Schmitt trigger t_s is comparable with a gate delay, whilst τ is typically an order of magnitude smaller than a gate delay. For example, from [C.7] for a 74S74 flip-flop τ is at most 1.7 nsec and the propagation

delay of the Schmitt trigger, 74S11, of the same logic family is typically 8.5 nsec. Even using a very high speed comparator with compatible STTL output the propagation delay is of the same order. Also the rise and fall times of the Schmitt trigger are at least as long as τ (usually much longer) hence $t_r/\tau > 1$. Thus, it is concluded that $R_2/R_1 >> 1$ which implies that the simple synchroniser performs much better than the Schmitt synchroniser. The significant factor is the loss of settling time due to the fact that the propagation delay of the Schmitt trigger exponentially deteriorates the probability of synchronisation failure, whilst the gains due to the possibly short rise time of the Schmitt trigger are at best linear (since $1 - \exp[- t_r/\tau]$ is concave downwards and passes through the origin, halving the rise time results in $1 - \exp[- t_r/\tau]$ larger than half its original value. But for $t_r/\tau << 1$, $1 - \exp[- t_r/\tau] \cong t_r/\tau$).

It has been shown that the strategy employed in the Schmitt synchroniser has had the opposite effect on the probability of failure, compared with the simple synchroniser, that originally intended. A rather simplistic view has been taken of the metastable behaviour of the flip-flop by the first order model employed. It has been shown experimentally, as discussed above, that the first order model gives estimates of probability of failure consistent with probabilities of failure in real flip-flops for the simple synchroniser circuit, however, it is not as clear how applicable the model is to the analysis of more complex synchroniser circuits such as the Schmitt synchroniser. The effects of noise of the flip-flop output and possible non monotonic output behaviour (eg. oscillations as observed in [C.8]) are not taken into account when the first order model is assumed. It is felt that the effect of these departures from the ideal first order behaviour of a flip-flop

would tend to increase the probability of failure of the Schmitt synchroniser, since the output of the Schmitt trigger becomes less reliable under these less ideal conditions. For this reason it is felt that the above conclusions remain the same for practical non ideal flip-flops.

3.7.5 Redundancy and Masking Techniques

It is possible to mask component failures using hardware redundancy techniques and considerable literature exists which considers both permanent and transient faults [K.1, W.1, W.2, W.4]. This section considers the possibility of achieving improvements in the reliability of synchronising an asynchronous signal by exploiting similar masking techniques such as majority voting. As will become evident from the results presented, synchronisation failure due to metastable behaviour cannot be treated as a hardware component failure in the usual sense, and redundancy and masking techniques are ineffective. This original result [1] has not been formally proved previously, nor been suggested elsewhere to the author's knowledge.

3.7.5.1 Masking in Synchronisers

Figure 3.20 illustrates the simplest form of a non redundant synchroniser. In this circuit, metastable behaviour of the input flip-flop, FF1, due to marginal triggering will be inconsequential provided it decays within approximately one clock period.

One possible structure of a redundant synchroniser, which exploits the masking provided by triplicated voting in order to suppress the anomalous behaviour of a synchronising element, is shown in Figure 3.24. The outputs of the three input synchronising elements, IFF1 .. IFF3, are

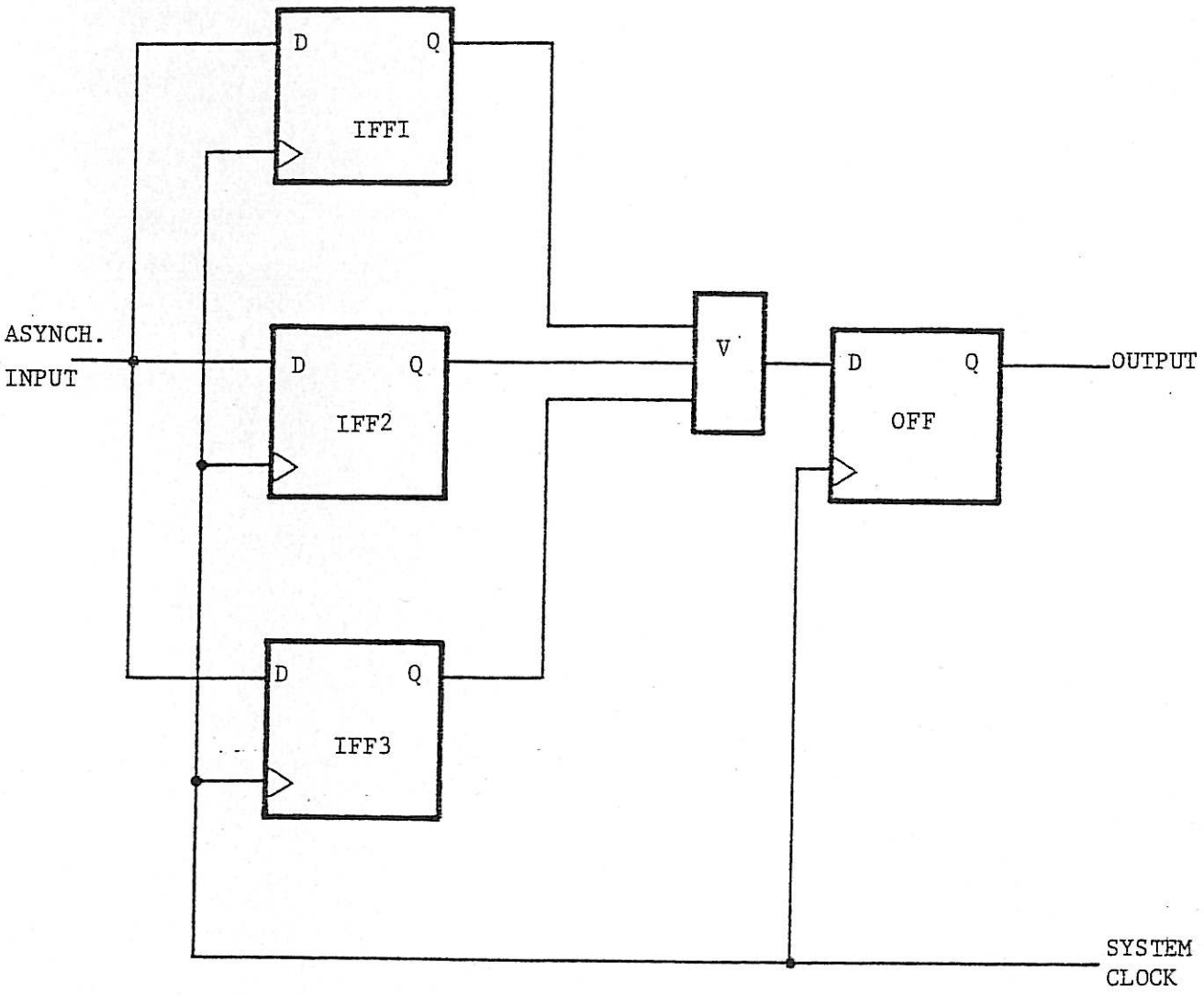


FIGURE 3.24 Triplicated Synchroniser.

voted on after one clock period settling time. If one flip-flop is unresolved, due to metastable operation, whilst the other two resolve to the same logic value, then the voter circuit will mask the metastable state. This occurs because the voter output is insensitive to changes in one input when the other two inputs agree. Should an input data edge occur that results in two flip-flops resolving to different values, and the third enters a metastable state, then the voter output may be undefined, since changes in the third input affect the voter output in this situation. Thus, the redundant synchroniser does not mask all occurrences of one metastable behaviour in three synchronising flip-flops.

A number of possible modifications come to mind which attempt to eliminate the observed deficiencies, for example, some form of phasing of the clock on the input flip-flops and increasing the number of flip-flops and voters. These possibilities can be incorporated in a general system consisting of n synchronising elements with arbitrarily phased synchronous clocks feeding a combinational function as shown in Figure 3.25. Note that the redundant synchroniser shown in Figure 3.24 is a specific example of the generic structure of Figure 3.25, with $n = 3$, $\tau_1 = \tau_2 = \tau_3 = 0$, and the combinational function is a majority voter. It will be shown that for any non trivial combinational function satisfying certain reasonable properties, the general redundant system, shown in Figure 3.25, cannot improve upon the simple synchroniser, shown in Figure 3.20, in terms of the probability of failure of the complete synchroniser due to metastable behaviour of the primary synchronising elements.

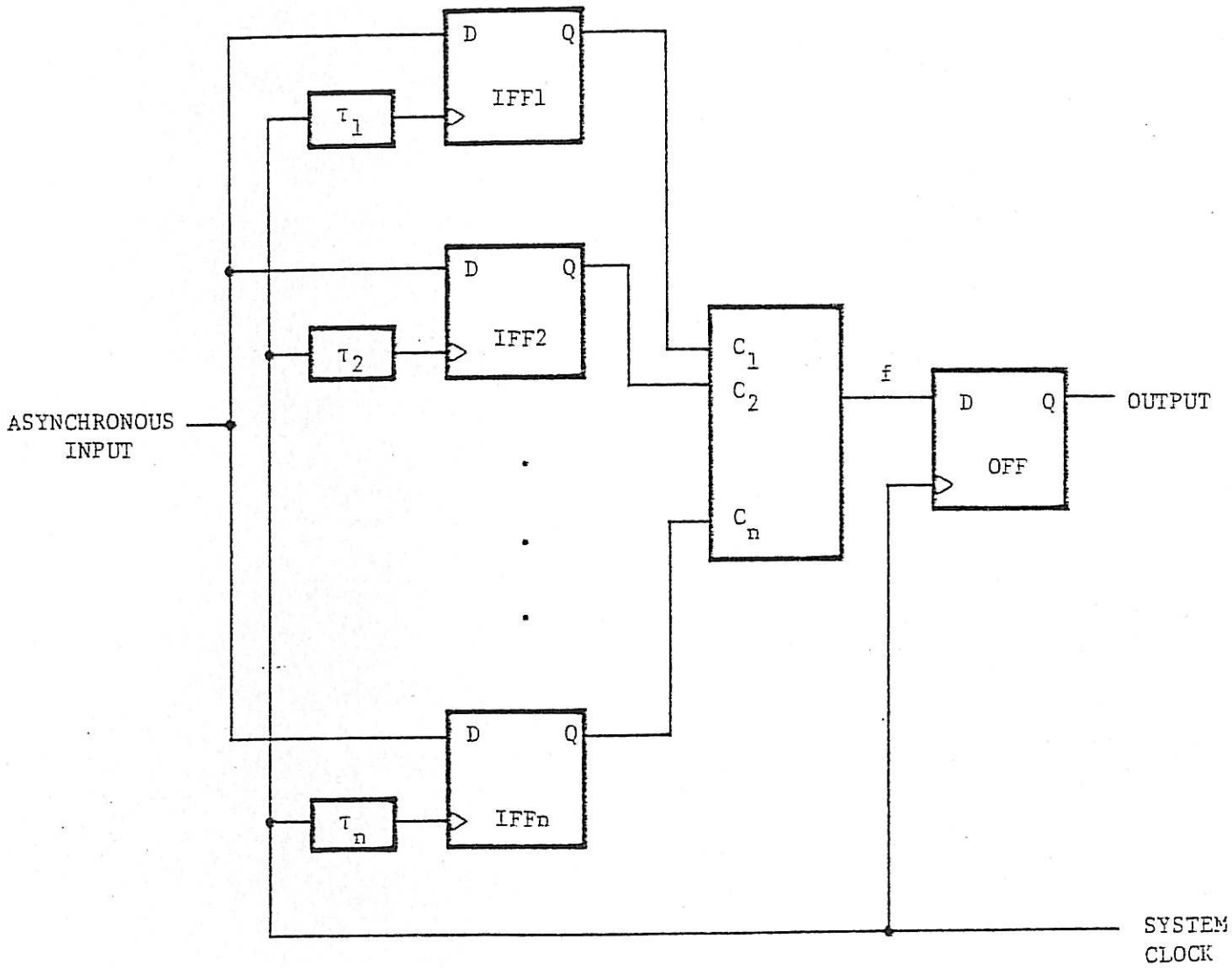


FIGURE 3.25 General Redundant Synchroniser.

3.7.5.2 Modelling of the General Redundant Synchroniser

The combinational function, $f: \{0, 1\}^n \rightarrow \{0, 1\}$, in the system shown in Figure 3.25 will be assumed to satisfy certain idealised but reasonable properties which are discussed in sequel.

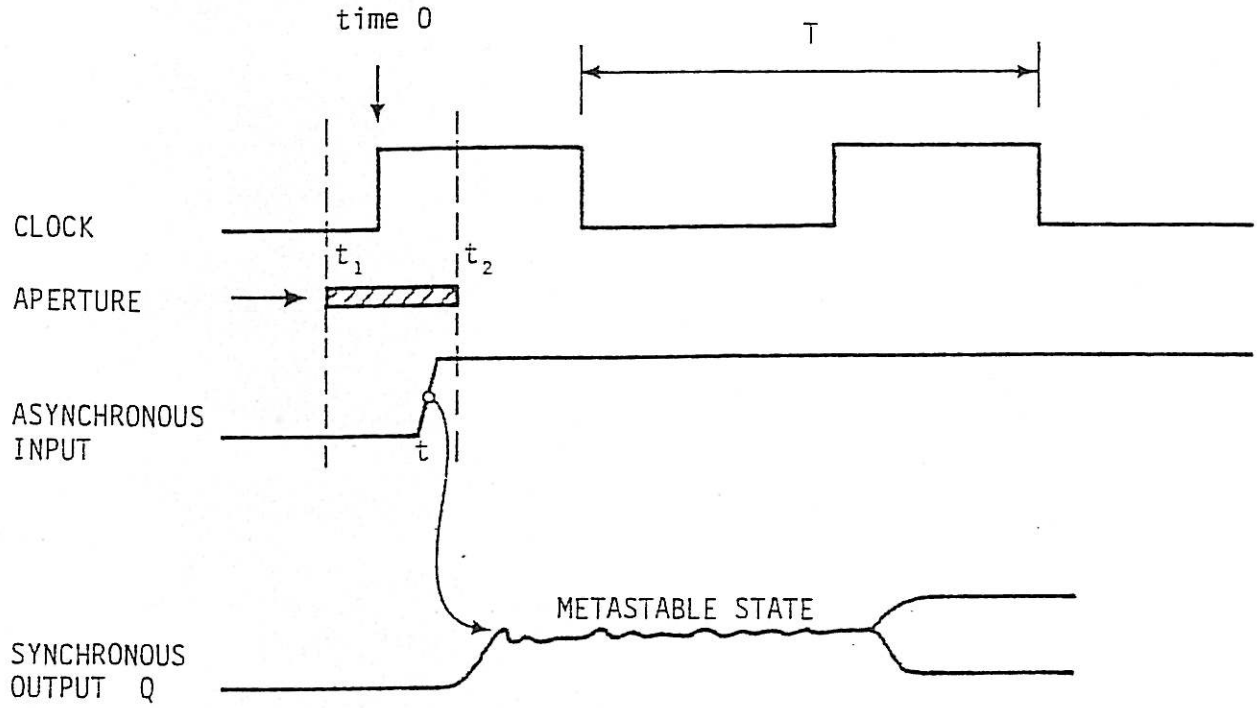
The combinational function is assumed to be non trivial, in the sense that

$$f(0, 0, \ldots, 0) = 0$$
$$f(1, 1, \ldots, 1) = 1 \qquad (3.40)$$

That is, if all synchronisers resolve to the same logic state, 0 or 1, then the combinational function also gives the same value. Note that (3.40) excludes the two trivial constant logic functions which, although completely masking metastable states, do not represent a satisfactory synchronised version of the asynchronous input.

Each of the n synchronising elements shown in Figure 3.25, namely IFF1, .., IFFn is modelled as follows: The output after the one clock period of settling time is one of three values 0, 1 or M. The 0 and 1 correspond to the output resolving to the respective binary logic levels of the system, whilst M corresponds to a synchronising element output not resolving to either logic level, due to a metastable state persisting for at least one clock period. An output M, at the end of one clock period, is assumed to occur as a result of the asynchronous input edge falling within an aperture. This input timing model is consistent with the models adopted in [F.2, H.3, L.1, R.1] and that discussed in Section 3.6.3.

The model for the synchronising element is illustrated in Figure 3.26, which shows the case of a positive asynchronous input edge



If $t_1 < t < t_2$ then metastable operation will persist for at least time ^2T.

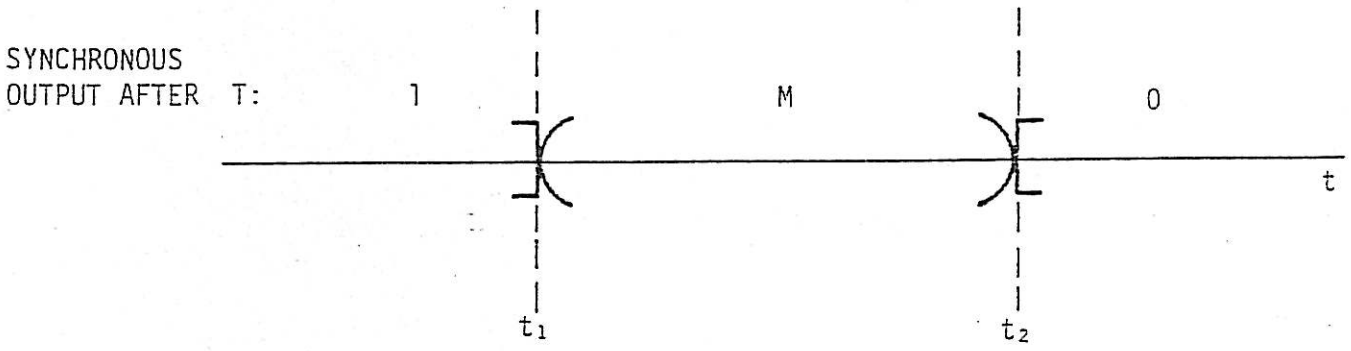


FIGURE 3.26 Aperture Model for Flip-Flop.

and the aperture of a synchronising element. In Figure 3.26, t represents the time after the sampling clock edge at which the asynchronous edge occurs. For $t \leq t_1(i)$ the output of synchronising element i after one clock period is 1; for $t_1(i) < t < t_2(i)$ it is M; and for $t \geq t_2(i)$ it is 0. Note that the aperture position and size is a function of the synchonising element, allowing for variability between the characteristics of the synchronising elements. Furthermore, the aperture position as defined above includes a delay between the clock input of each synchronising element and the system clock. The aperture boundaries are assumed to be sharp in the sense that the synchronising elements behave deterministically as a function of t as defined above.

In order to study the redundant synchroniser circuit shown in Figure 3.25, it is necessary to define the behaviour of the combinational circuit for inputs assuming the value M. Some inputs may be masked or desensitised by other input values. It was assumed implicitly in discussing the voter that its output is unaffected by an M value on a desensitised input. However, an input which assumes a value M, when sensitised by other inputs assuming their appropriate 0 or 1 values, produces an undefined of M output at the voter. This simple concept of single input sensitivity needs to be extended to allow for the presence of more than one M input at a time.

The following generalised definition of input sensitivity is adopted: A set of inputs, which have the value M, is sensitised by particular values of the remaining input variables if there are two substitutions for these inputs that give outputs 0 and 1. It is assumed that if the set of inputs which have value M is sensitised then the output is M. This concept is expressed in the following definition of an extended combinational circuit function, f_M: $\{0, 1, M\}^n \rightarrow \{0, 1, M\}$.

Definition: Let $(C_1, \ldots, C_n)$ be an input vector with components from $\{0, 1, M\}$, k of which have the value M. Let $(D_1, \ldots, D_n)$ be an input vector with M valued components replaced by 0 or 1. That is

$$D_1 = \begin{cases} C_i & \text{if } C_i = 0 \text{ or } C_i = 1 \\ 0 \text{ or } 1 & \text{if } C_i = M \end{cases}$$

then

$$f_M(C_1, \ldots, C_n) \triangleq \begin{cases} B, & \text{if } f(D_1, \ldots, D_n) = B \text{ for all } 2^k \text{ possible substitutions } (D_1, \ldots, D_n), \ B \in \{0, 1\}. \\ M, & \text{if } f(D_1, \ldots, D_n) \text{ is not constant for all } 2^k \text{ possible substitutions } (D_1, \ldots, D_n) \end{cases} \tag{3.41}$$

Example: To illustrate the significance of the definition in (3.41), the special case of the majority voter combinational function is considered in detail. The output as a function of the 27 possible inputs is shown in the Table 3.1. Metastable masking occurs in the rows 2, 4, 10, 18, 24 and 26.

TABLE 3.1

EXTENDED MAJORITY VOTER FUNCTION

ROW	INPUTS			OUTPUT
	x_1	x_2	x_3	$x_1x_2+x_1x_3+x_2x_3$
1	0	0	0	0
2	0	0	M	0
3	0	0	1	0
4	0	M	0	0
5	0	M	M	M
6	0	M	1	M
7	0	1	0	0
8	0	1	M	M
9	0	1	1	1
10	M	0	0	0
11	M	0	M	M
12	M	0	1	M
13	M	M	0	M
14	M	M	M	M
15	M	M	1	M
16	M	1	0	M
17	M	1	M	M
18	M	1	1	1
19	1	0	0	0
20	1	0	M	M
21	1	0	1	1
22	1	M	0	M
23	1	M	M	M
24	1	M	1	1
25	1	1	0	1
26	1	1	M	1
27	1	1	1	1

Consider the redundant synchroniser in Figure 3.24 where the 3 synchronising elements IFF1, IFF2 and IFF3 have non overlapping apertures (say $t_1(1) < t_2(1) < t_1(2) < t_2(2) < t_1(3) < t_2(3)$). In this case, the output of the voter is M after one clock period if and only if IFF2's output is M after one clock period. Thus, the redundant synchroniser has the same aperture characteristic as the single synchronising element IFF2 and no improvement has been achieved in reducing synchronisation failure.

3.7.5.3 Statement and Proof of the Result

Theorem 3.2 Assuming that the n synchronising elements IFF1, ..., IFFn of the redundant synchroniser can be described by the aperture model above, and that (3.40) and (3.41) hold for the combinational circuit combining their outputs, then the range of asynchronous input edge times that cause an undefined combinational circuit output after a clock period is at least as large as the minimum aperture width of the individual synchronising elements IFF1, ..., IFFn. Thus, the redundant synchroniser's probability of failure is no better than the best of its synchronising elements acting alone.

Proof Only the positive asynchronous input edge case is considered since the negative edge proof is similar. In this proof it is shown that there is an interval of asynchronous input edge times that sensitises the combinational circuit to a synchronising element which is exhibiting metastable behaviour. The size of this interval is shown to be at least as large as the minimum aperture width.

Firstly, the synchronising elements are relabelled by the chronological order of right aperture boundaries, to give:

$$t_2(1) \leq t_2(2) \leq \ldots \leq t_2(n) \tag{3.42}$$

Figure 3.27 shows a set of relabelled synchroniser apertures plotted against the asynchronous input edge time. Note that if IFFi's output is 0 then IFFi-1, ..., IFF1 all have 0 outputs.

Consider now the following sequence of input vectors of the combinational circuit that changes one input at a time:

$$\begin{aligned} u_1 &= (1, 1, \ldots, 1) \\ u_k &= (0, 0, \ldots, 0, 1, 1, \ldots, 1) \\ &\qquad \text{where } C_1 = C_2 = \ldots = C_{k-1} = 0 \\ &\qquad \text{and } C_k = C_{k+1} = \ldots = C_n = 1 \\ u_{n+1} &= (0, 0, \ldots, 0) \end{aligned} \tag{3.43}$$

From (3.40) and (3.43) it follows that

$$f(u_1) = 1, \quad f(u_{n+1}) = 0 \tag{3.44}$$

Hence, from (3.43) and (3.44), there exists an integer, i, such that

$$f(u_i) = 1 \text{ and } f(u_{i+1}) = 0 \tag{3.45}$$

That is, with input vector u_1, input C_1 is sensitised. Now define

$$t_1 \triangleq \max \left\{ t_1(1), t_1(2), \ldots, t_1(i) \right\} \tag{3.46}$$

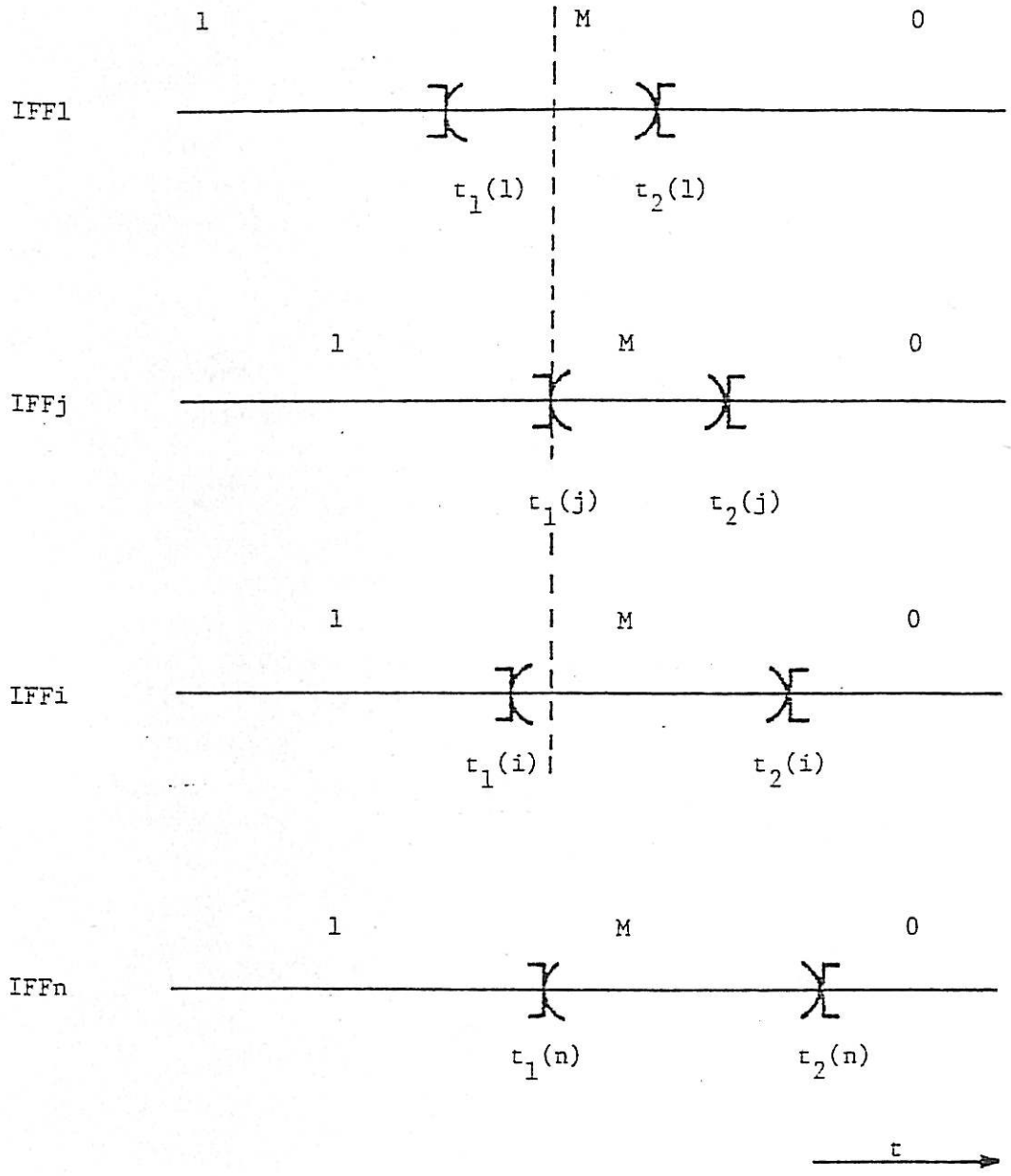


FIGURE 3.27 A Possible Set of Apertures Illustrating the Proof.

It will be shown that for $t_1 < t < t_2(i)$ $f_M = M$. The situation is illustrated in Figure 3.27 where the maximum of (3.46) occurs at $t_1(j)$.

Now, for $t_1 < t < t_2(i)$:

$$C_p = 0 \text{ or } M \qquad \text{for } p = 1, \ldots, i-1 \tag{3.47a}$$

$$C_i = M \tag{3.47b}$$

$$C_q = M \text{ or } 1 \qquad \text{for } q = i+1, \ldots, n \tag{3.47c}$$

Equation (3.47a) follows since $t > t_1 \geq t_1(p)$; (3.47b) since $t_1(i) \leq t_1 < t < t_2(i)$ and (3.47c) since $t < t_2(i) \leq t_2(q)$. These relations can be easily seen in Figure 3.27. Two substitutions $(D_1^1, D_2^1, \ldots, D_n^1)$ and $(D_1^2, D_2^2, \ldots, D_n^2)$ for $(C_1, \ldots, C_n)$ that differ from (3.47) only in M value positions are defined below:

$$\begin{aligned} &D_p^1 = D_p^2 = 0 \qquad , \quad p = 1, \ldots, i = 1 \\ &D_i^1 = 1, \quad D_i^2 = 0 \\ &D_q^1 = D_q^2 = 1 \qquad q = i + 1, \ldots, n \end{aligned} \tag{3.48}$$

It can be seen from (3.43) that these two substitutions are simply u_i and u_{i+1}. Thus (3.45) and (3.41) imply that

$$f_m = M \quad \text{for} \quad t_1 < t < t_2(i) \tag{3.49}$$

Now the width of the interval $(t_1, t_2(i))$ is examined. Recalling that the maximum of (3.46) occurs at $t_1(j)$ (and hence $j \leq i$)

$$\begin{aligned} t_2(i) - t_1 &= t_2(i) - t_1(j) \\ &\geq t_2(j) - t_1(j), \quad \text{from (3)} \\ &\geq \min_{k = 1, \ldots, n} \{t_2(k) - t_1(k)\} \end{aligned} \tag{3.50}$$

Relations (3.49) and (3.50) establish the first part of the theorem.

To examine the probability of failure due to metastable behaviour of the non redundant synchroniser of Figure 3.20, the probability of Q_1 being undefined when the output flip-flop, FF2, samples Q_1 after one clock period, needs to be determined. The probability of failure of the redundant synchroniser of Figure 3.25 and the non redundant synchroniser of Figure 3.20 can be compared by considering the probability that the input of the output flip-flop is undefined at the time of sampling. This is a legitimate comparison of the relative performances, since the output flip-flop can be assumed to exhibit the same behaviour in both cases.

Under reasonable assumptions regarding the stochastic behaviour of the asynchronous input, it is shown in Section 3.6.3 equation (3.30) that for the non redundant synchroniser of Figure 3.20, the rate of FF1 not resolving to a valid logic state after a clock period due to metastable behaviour is proportional to the aperture width of IFF1. It has been shown that the set of time instants giving rise to an undefined combinational output in the redundant synchroniser contains a contiguous interval whose width is at least as large as that of the best synchronising element. Hence, the probability of the input to FF0 of the redundant synchroniser being undefined is bounded below by the probability that would result from the best synchronising element acting alone as in Figure 3.20.

3.7.5.4 Observations

It should be noted that the combinational circuit in the redundant synchroniser of Figure 3.25 was assumed to introduce no delay. The effect of a non zero delay is to reduce the settling time allowed for each of the synchronising elements. This has the effect of dramatically increasing the aperture width of the synchronising elements as can be seen in equation (3.27). This implies a further degradation in performance of the redundant synchroniser.

In conclusion, it has been shown that based on a simple model for the metastable behaviour of a synchronising element, no effective improvement in synchronisation performance can be achieved by employing any number of parallel synchronising elements, regardless of the way their outputs are combined in a combinational logic circuit.

3.8 CONCLUSION

This chapter has presented an overview of developments in the study of metastable behaviour in digital circuits, with more detailed treatment of contributions by the author in areas of analysis and modelling of metastable failure, and development of rigorous techniques to establish the unavoidability of metastable behaviour. Furthermore, the chapter has emphasised the need to design with special attention to the problem of metastable failure in order to achieve high reliability, especially in light of the dramatic range of failure rates possible. Further development of accurate analysis techniques to predict the metastable reliability performance should be pursued to fill a void in the literature.

A number of techniques for reducing the probability of metastable failure have been presented. These include the use of fast, specialised

devices, pausable clocks, redundancy and masking, extended decision time, and filtering combined with extended decision time. Quantitative, as well as, qualitative evaluations of techniques by the author have been presented. It appears that the simple solution of adequate settling time combined with the use of properly tested fast devices is the most reliable method of reducing the failure rate to a negligible level.

CHAPTER 4

ANALYSIS APPROACHES AND MODELLING OF ARBITERS

4.1 INTRODUCTION

It was shown in Chapter 2 that the characteristics of arbiters are key factors in determining the performance of computer systems due to the central role of the arbiter in primitive resource allocation. The aim of this chapter is to introduce the analysis approaches that can be taken to assess the performance of arbiters. Since the suitability of different approaches depends on the modelling assumptions adopted for the arbiter, it is convenient to define and discuss the models in conjunction with the approach of analysis. The actual analyses are deferred to later chapters.

The organisation of this chapter is as follows: Section 4.2 discusses the applicability of queueing theory results and the few previous attempts at a theoretical analysis of arbiters. The approaches taken by the author are introduced in Sections 4.3, 4.4 and 4.5. Sections 4.3 and 4.4 discuss modelling of batched and non batched arbiters respectively, with the batched and non batched fixed priority disciplines used as detailed examples. The analysis approach discussed in Sections 4.3 and 4.4 is based on the technique known as imbedded Markov chains. To the knowledge of the author this approach applied to arbiters is new and, in particular, no previous analyses of *batched* arbiters have appeared. The analysis allows general service time distributions and modelling of circuit delays of the arbiter. In Section 4.5, the commonly used Monte Carlo approach to arbiter analysis is discussed and compared with the theoretical approaches. The chapter ends with conclusions concerning the different analysis approaches, possible improvements, generalisations and future work.

4.2 QUEUEING THEORY TECHNIQUES

A widely accepted notation and classification of queueing systems [G.2, K.12] is:

arrival process /service process /number of servers

and an optional appendage with defaults in brackets is:

/ capacity of system (∞) /number of requesters (∞)
/ service discipline (FCFS).

The arrival and service processes are described by their inter-arrival and service time probability distributions, with the following notation commonly employed:

M	Exponential (Markovian)
D	Constant (Deterministic)
G	General

The M/M/1 system is the simplest queueing problem due to both arrival and service time distributions being Markovian [K.12]. FCFS systems with both arrival and service distributions Markovian are classified as elementary queueing problems [K.12]. An elementary queueing model for a FCFS k requester arbiter is the M/M/1/k/k system which is often called the machine repairman problem. The M/M/1/k/k system can be analysed using a birth-death continuous time Markov process that enables the steady state queue length distribution to be expressed in the form of a truncated Poisson distribution [G.2, K.12]. Performance measures can be derived

from the steady state queue length probabilities, such as the mean number of requests pending, mean waiting time, mean resource utilisation and the effective arrival rate of requests. These performance parameters are a function of the request loading and are independent of the discipline for identical requester excitation characteristics [B.1, G.2, K.12, M.5].

The elementary queueing theory approach has several drawbacks: The discipline is limited to FCFS; arrival and service distributions must be exponential; and only ideal models with no inter-service delay are treated. These points are discussed in sequel below.

Disciplines other than FCFS need to be analysed when the following measures are required: higher than first order moments of waiting times; the mean waiting times of each requester; and proportion of time allocated to each requester. Priority queueing theory results examined in the often quoted work by Jaiswal [J.1] are limited to infinite source models when the number of priority classes is greater than two. These results are described as algebraically complex. The finite source case (applicable to arbiters) is claimed by Jaiswal [J.1, p.152] to lead to "much more complicated results" and these are not presented. (The complexity of finite source models can be attributed to the arrival process of requests being dependent on the number queued. This may explain the scarcity of literature studying non FCFS arbiters from a theoretical viewpoint.) Mühlemann [M.9] examines an M/M/1/k/k system with fixed priority classes, where requests within the same class are resolved by FCFS. The state transition rates are derived for the continuous time Markov process. The complexity and number of simultaneous equations to be solved for the steady state probabilities prevents a closed form solution. For the simple case of a fixed priority arbiter $1 + k2^{k-1}$ states are required (e.g. a 6 requester arbiter has 193 states). This to be compared with 2^k

states (e.g. k = 6 requires 64 states) for the analysis employed by the author in Section 4.4 and which allows general service time distributions and inter-service delays.

The assumption of an exponential re-request time distribution may not be an accurate model for autonomous random requesters due to possible "bursty" arrival periods or other fluctuating behaviour. The exponential distribution more accurately models requesters that have equal likelihood to request at any time (provided they have not already requested). That is, their re-request time distribution is memoryless, meaning the probability of a request is independent of the time already elapsed since last receiving service. The exponential distribution is the only distribution with the memoryless property [G.2, K.12] and it is this property which makes the exponential distribution a suitable basis for analysing queueing models. Many random events can be successfully modelled with the exponential distribution but its limitations cannot go unnoticed.

The fully interlocked protocol between Req and Ack discussed in Chapter 2 ensures that requesters cannot buffer requests while waiting access to the resource. This assumption corresponds to the finite population assumption made in queueing theory [G.2, K.12] and models requesters, such as processors, which cannot proceed until they receive access to the resource, such as a bus or common memory. This blocking action on multiple requests from the one requester is a property of the primitive level of resource allocation offered by arbiters as discussed in Chapter 2 - at a higher level, a processor requesting an I/O transaction, for example, may proceed with other tasks which may themselves generate further I/O requests, until the transaction is completed.

The assumption of an exponentially distributed service time may not be a good model, especially with resources such as buses and memory where the service time is often constant. An exponential distribution weights very short times more than longer times. It is often the case that the minimum service time is significantly above zero. Considerations of this type encourage the adoption of a general service time distribution assumption, as is done by the author. Once a general service time distribution is assumed, analysis by employing a continuous Markov process is not possible due to the non Markovian service process. A Markov renewal process or an imbedded Markov chain [G.2, K.12] is employed by the author. This approach applied to arbiters is new to the author's knowledge.

Kleinrock shows [K.12 Vol II pp 210-212] that the queue length distribution of a M/R/1/k/k system, where R denotes service time distributions with a rational Laplace transform, is independent of service distributions with the same mean. From this result it may be suggested that the performance measures of the M/G/1/k/k queueing problem are relatively insensitive to different service time distributions with the same mean. Small, but not insignificant, sensitivity to the distribution is borne out by results obtained by the author on performance measures other than the queue length, such as MWT(h) and PROP(h) presented in later chapters. Thus, it is more precise to employ the appropriate, possibly non Markovian, service distribution best modelling the application.

Performance measures are, however, sensitive to the magnitude of some non ideal arbitration delays that occur between service times or batches as can be seen from results presented in Chapters 5, 6 and 8. As will be seen in Section 4.3, some of these delays cannot be incorporated at the end of service times because requesters are enabled to re-request

before these delays by their acknowledge signals resetting, while if the delay were modelled as appended to the service time, the requester would not be enabled to request until the end of the delay. These delays consequently cannot be modelled accurately by incorporating them into the service times of the queueing model M/G/1/k/k. The structure of the model employed by the author is described in the next section.

4.3 MARKOV APPROACH TO THE ANALYSIS OF BATCHED ARBITERS

In this section the method of imbedded Markov chains [G.2, K.12] is described as applied to the analysis of batched arbiters. Due to the flexibility of the approach, time intervals resulting from circuit delays occurring at the start and end of the servicing of batches of requests can be modelled. These non ideal durations, in which no requests are serviced, are referred to as *inter-batch delays* and are usually smaller than service times in well designed arbiters. The other advantage of the approach is the capability to treat general service time distributions which may be different for different requesters. The underlying assumptions necessary for the analysis are defined precisely in Section 4.3.3, but the most important is the Markovian re-request time distribution which is necessary for the sequence of batches to be modelled as a Markov chain.

The key step of the method of imbedded Markov chains is the identification of so called *regeneration points*. At the time of a regeneration point, the process is renewed in the sense that all past history becomes irrelevant for predicting future events. In a batched arbiter, the times at which requests are locked out, called batching points (or in terms of the general model, when the batch marker is moved to the end of the queue), form a sequence of regeneration points. The

sequence of discrete times defined by the regeneration points enables a discrete time Markov chain to be formed, where the state is defined just after each regeneration point. Since requests occurring between batching points determine the composition of the next batch, the state of the arbiter is dependent only on the state at the previous batching point (provided the discipline is amenable to a Markovian state representation - this is discussed more fully below). This result is proved formerly in Section 5.3 for a fixed priority batched arbiter, where the state is defined to be the set of pending requests at a batching point.

Once regeneration points are defined, state transition probabilities are derived for the Markov chain. Provided the Markov chain is irreducible and acyclic [G.1, G.2, K.12] unique steady state probabilities exist and are independent of the initial state of the arbiter. From the steady state probabilities, performance parameters can be derived for the arbiter.

These results assume the batched discipline is Markovian. For a fixed priority batched arbiter the discipline is static and hence its state is not influenced by past events at all (and is not included in the state definition). Other batched disciplines, such as dynamic priority batched disciplines, are dependent on the previous state history of the arbiter. For example, LRU is dependent on the order of most recent services of each requester, and so can be dependent on events occurring an unbounded time in the past. This does not present a problem when the discipline state is defined appropriately, for example by the current priority mapping. With this state definition the discipline can be seen to be Markovian, as demonstrated by the iterative relation on the dynamic priorities presented in Chapter 2. All the other disciplines presented in Chapter 2 are Markovian when an appropriate discipline state definition is

adopted. The total arbiter state can then be represented as a two dimensional vector of the discipline state and request state just after a batching point. However, it is conceivable that a discipline could be defined in such a way that either a Markovian state representation cannot be found or all Markovian state representations are of infinite dimension. For example, a dynamic priority discipline can be defined that requires an infinite dimensional state representation as follows: Priority is given in order of each requester's individual sum of previous waiting times. In order to determine these priorities, tallies of waiting times (real numbers) should be kept for each requester which theoretically requires an infinite number of states.

The analysis approach of finite imbedded Markov chains must obviously be restricted to disciplines with a finite Markovian state representation. In this thesis details of the analysis technique are restricted to the fixed priority batched arbiter where no discipline state representation is necessary, however the generalisation to any Markovian discipline is not difficult but adds to the complexity, especially in the number of states and transitions, and is left for future work. Moreover, the analysis of the fixed priority batched discipline is more fruitful than would be the case for symmetric disciplines, due to the bias in request treatment which is observed in the MWT(h) and PROP(h) results.

4.3.1 Examples of Asynchronous Fixed Priority Batched Arbiters

One example each of a centralised and a distributed asynchronous fixed priority batched arbiter is presented in this section with the aim of developing a general model for asynchronous fixed priority batched arbiters. The general model is presented in the section following. Other batched disciplines can be modelled in a similiar manner. Clocked versions are considered in Chapter 7.

Distributed Example

A simple distributive daisy-chained arbiter is described in this example. Similar designs have appeared elsewhere [C.5].

In the distributed arbiter circuit shown in Figure 4.1. each requester has a circuit module associated with it. The modules are connected via a daisy chain and a common line. The order of the modules in the daisy chain determines the priority of the requesters.

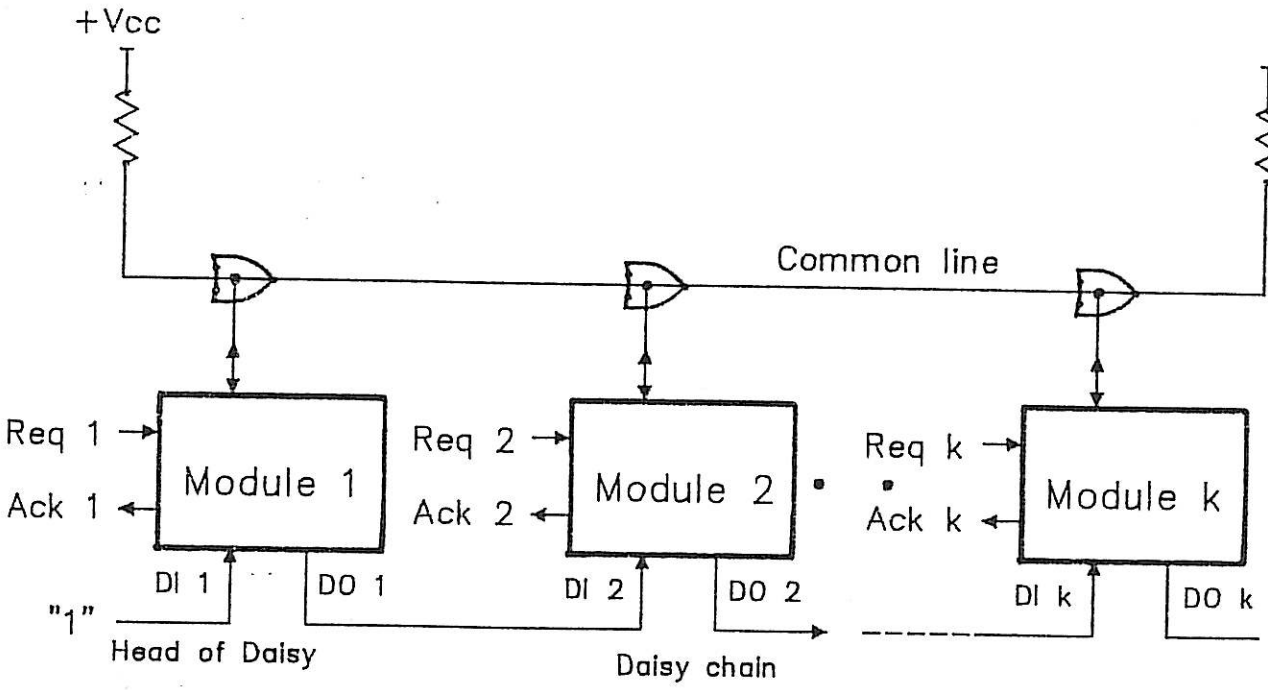


FIGURE 4.1 Structure of the Decentralised Daisy-Chained Batched Arbiter.

The start of the daisy chain is connected to logic '1', and the daisy chain is "threaded" through each module. When a module receives a 1 on the daisy chain from higher modules in the chain, it passes the 1 further down the chain if the module has no request latched from its requester, otherwise it "keeps" the 1 until the latched requests have been serviced. This is achieved by use of the common line which realises a wired-or

function of all the latched requests of the k modules. The common line is asserted when at least one request is latched. When the common line is asserted, no new requests can be latched in the modules, and when all latched requests have been serviced, the common line resets and all pending requests are latched. This has the effect of batching requests and preventing a high priority requester hogging the system by continually locking out the other requesters.

Figure 4.2 shows the details of one module in the arbiter. Figure 4.3 illustrates the internal arbiter timing at the end of a non zero batch and identifies the inter-batch delays that occur. The following abbreviations for delays appear in Figure 4.3. FFreset is the flip-flop reset time; gd is a gate delay; clp is the common line propagation delay between modules ℓ and h; τ_η rev is the delay on the reverse edge through τ_η, $\eta = 1,2,3,4$; dcpd is the daisy chain propagation delay from module 1 to h.

The batch in Figure 4.3 is terminated by requester ℓ resetting its request line, causing the common line to reset (high). Low-high-low pulses can occur on a wired-or line connected to open collector drivers even when a driver is constantly active (low) [G.5]. For the case of the arbiter circuit, the width of these spurious pulses is less than twice the maximum common line propagation delay, clp. (An alternate solution to the problem is to use current drivers in place of open collector drivers to eliminate spurious pulses. This scheme is employed in the Fastbus Cable Segment [C.10]). The edge delay τ_4 acts as a filter, which prevents common line low-high-low pulses of duration less than 2 clp from entering arbiter modules. Hence, a delay of τ_4 and reverse edge delays, τ_3 rev and τ_2 rev, occur before the daisy chain is reset at which time only

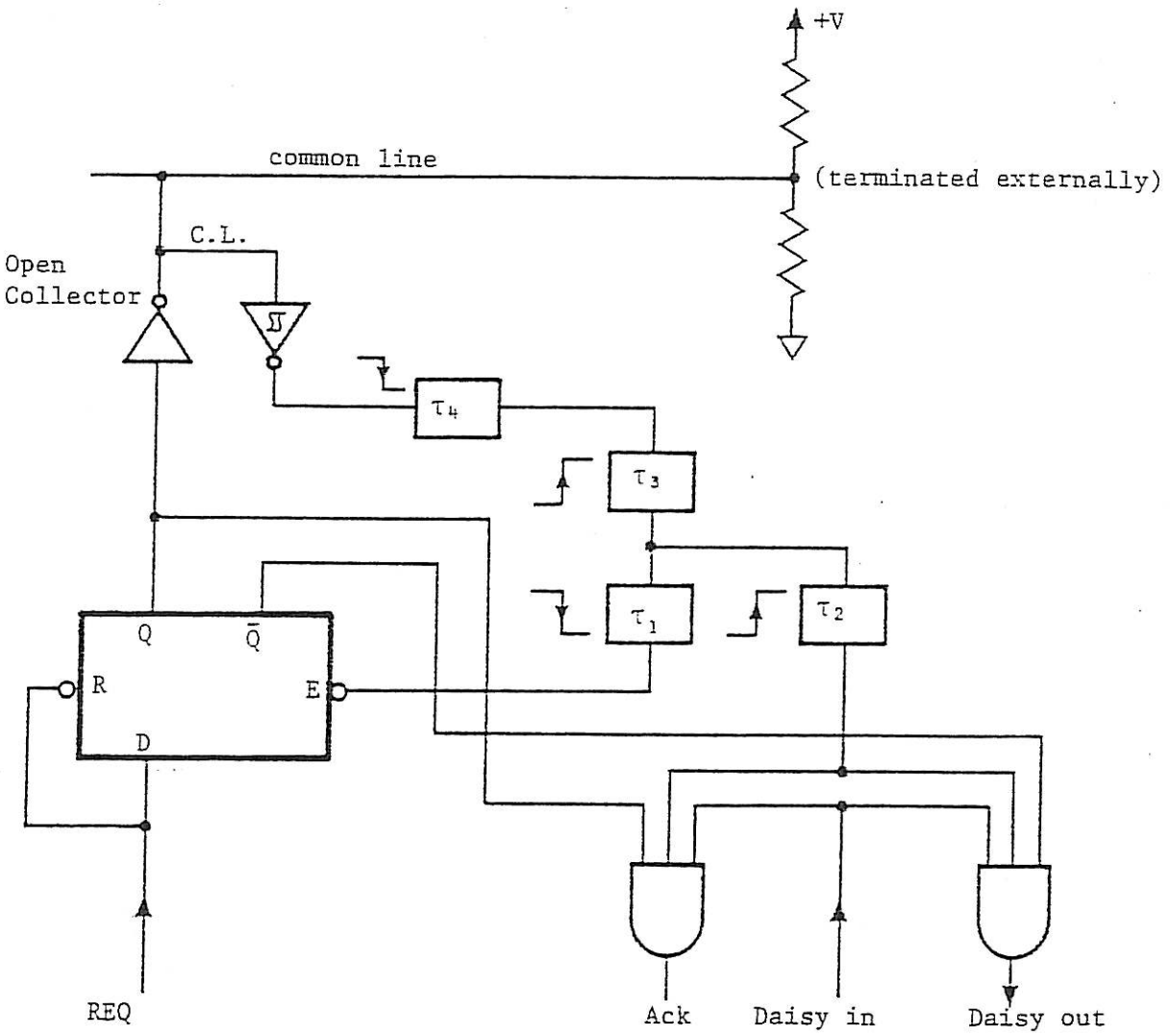


FIGURE 4.2 The Daisy Chained Batched Arbiter Module.

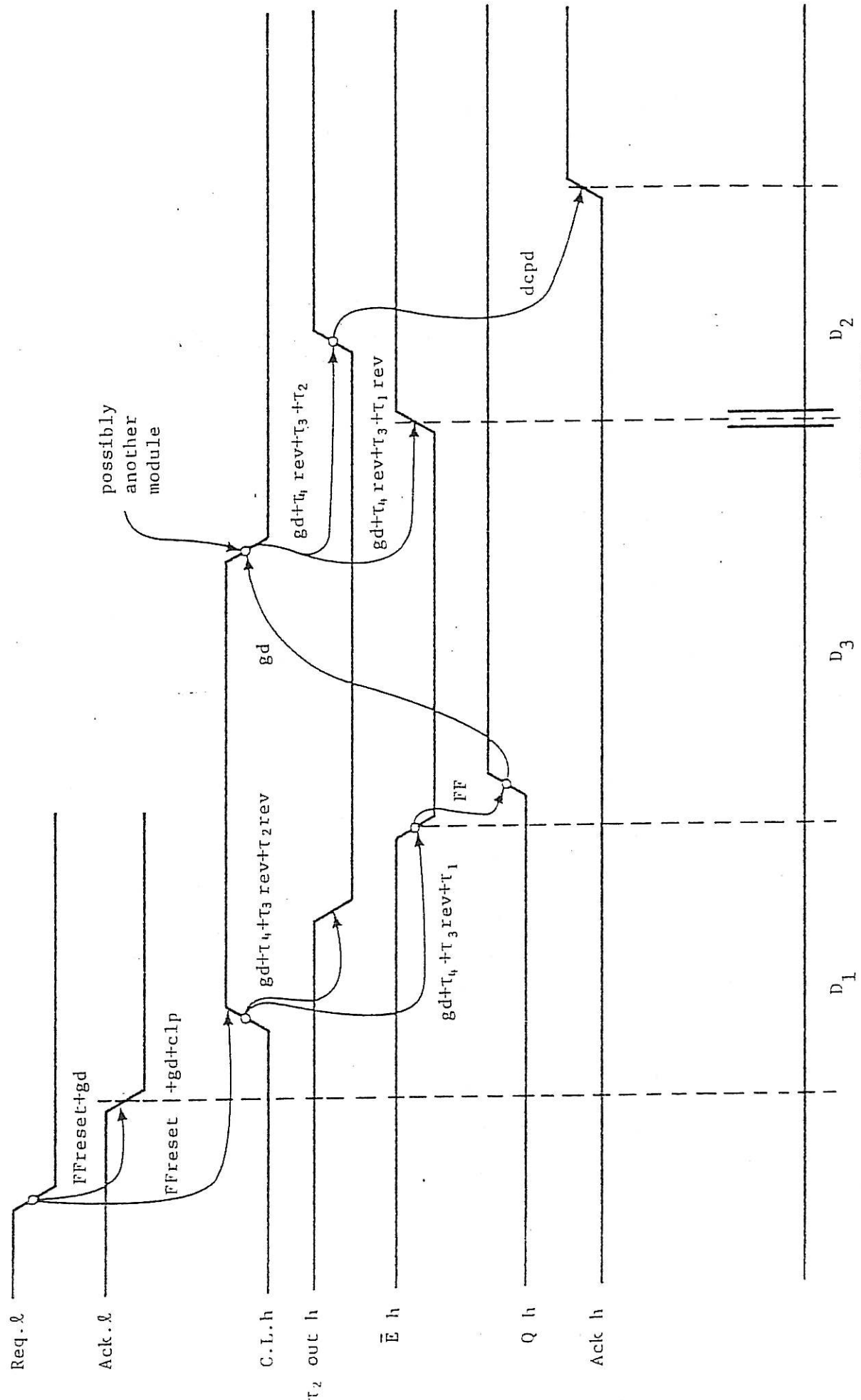


FIGURE 4.3 Timing Diagram for End of Non Zero Batch.

module 1 has "daisy in" asserted. The latch is enabled a time $\tau_1 - \tau_2$ rev later, in order to allow the daisy chain to reset before any flip-flops are set. The time duration from the last requester in a batch dropping its request to the enabling of each latch is denoted D_1. Note that if no request inputs are high of the end of D_1 a zero batch would follow immediately.

The next time duration, D_3, occurs only when requests are pending at the end of D_1. The requests are latched by their respective module flip-flops when $\bar{E}$ goes low, causing the common line to be asserted (low). The removal of the enable to the flip-flops occurs as a result of the common line going low but is delayed by τ_3 to guarantee a minimum enable pulse width. This completes D_3, and any further requests occurring after D_3 are not latched until all requests that have already been latched are serviced in the next batch.

The first acknowledge does not occur until after τ_2 delay plus the delay for the daisy chain to propagate to the highest priority module with a flip-flop set. The delay τ_2 is present to allow metastable settling time for flip-flops whose request input happens to be asserted very close to when the enable is removed, causing marginal triggering of the flip-flop. Notice that any requests occurring during or after D_2 are not latched until after all latched requests are serviced.

Consider now the case of a request dropping during a batch. When a requester finishes with the resource and drops its request whilst other requests are latched, the daisy chain propagates a "1" further down the chain until another module blocks it and the respective requester takes the resource. The common line is not affected (apart from spurious pulses mentioned above) since there are still latched requests.

The transition from a zero batch, the idle state of the arbiter, to a non zero batch is similar to the above timing diagram except D_1 is absent, whilst the non zero to zero batch transition is similar except with D_2 and D_3 absent.

The fixed priority batched arbiter model described in the next section approximates this decentralised daisy chain arbiter by assuming D_1, D_2 and D_3 are constant. The time durations D_1, D_2, and D_3 shown in Figure 4.3 may vary slightly between batch transitions and modules due to different common line and daisy chain propagation delays, but the effect of incorporating differences is felt to cause unnecessary complexity for little or no improvement in modelling.

Centralised Example

A centralised asynchronous fixed priority batched arbiter is shown in Figure 4.4. The priority logic implements the fixed priority discipline of the arbiter (a priority encoder-decoder circuit may be designed similar to that used in the non batched fixed priority arbiter of Figure 2.9), where an Ack output can only be asserted when the enable is asserted.

The functions of the edge delays τ_1, τ_2 and τ_3 are similar to those in the previous example. τ_1 delays enabling of batches until the priority logic is fully disabled; τ_2 determines the metastable settling time and τ_3 enables a minimum enable pulse width to be guaranteed at the end of a batch.

The timing of the arbiter for a non zero to non zero batch transition is similar to the previous example except that $\tau_4 = 0$ and dcpd is replaced by the priority logic delay. Similar inter-batch durations D_1, D_2, and D_3 can be defined. Due to the centralised nature of the circuit, D_1, D_2, and D_3 vary little between batches.

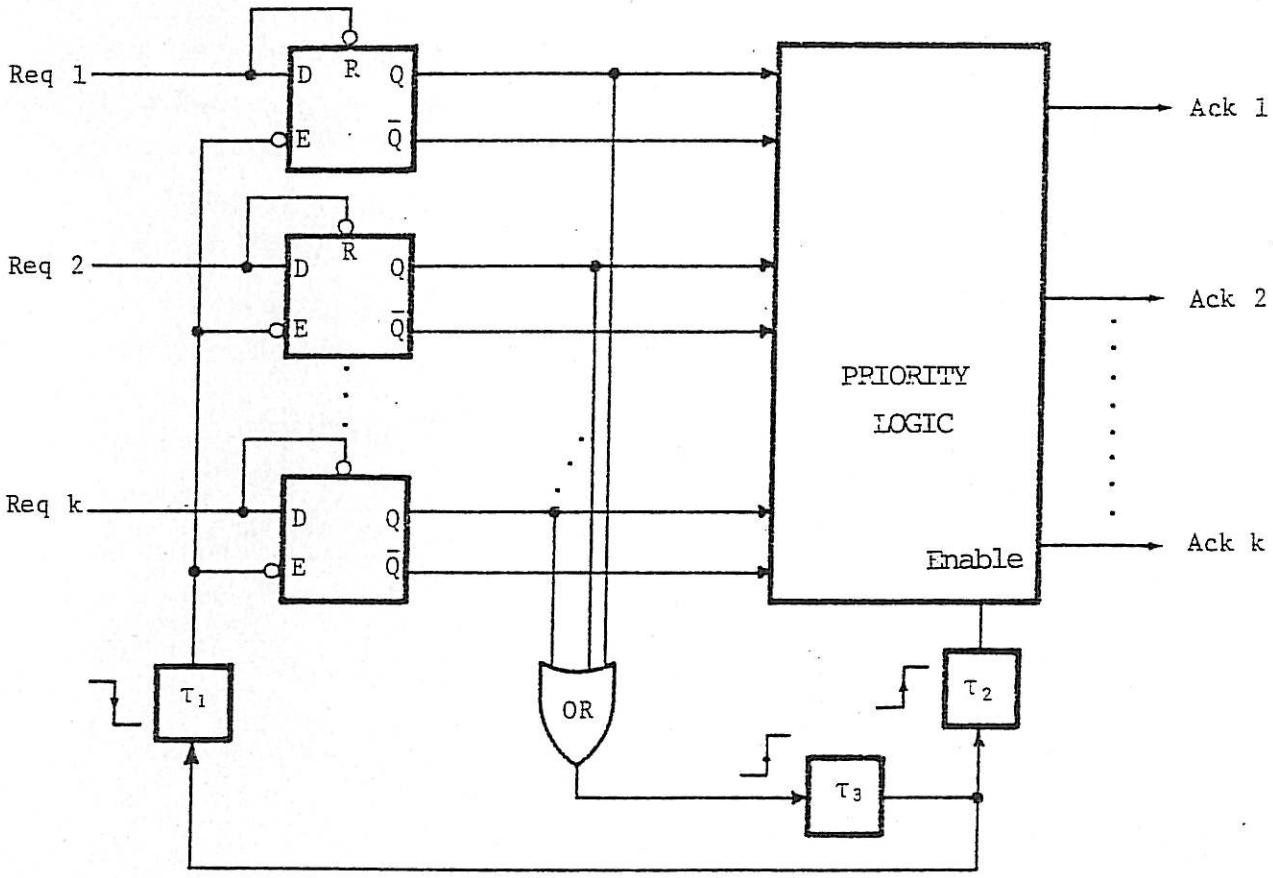


FIGURE 4.4 Centralised Batched Fixed Priority Arbiter.

4.3.2 Fixed Priority Batched Arbiter Model

The behaviour of arbiters with fixed priority and batching can be described by the following model. Suppose, initially the arbiter is idling (no requests pending). The first request initiates a time interval of duration D_3. This and other requests lodged during D_3 are batched together, and later requests are not considered until all the batched requests have been serviced in a logical time period referred to as a *batch*. As shown in Figure 4.5, a non zero batch consists of a time interval D_2 followed by priority ordered servicing of the batched requests followed by another time interval D_1 and then, if at least one request is lodged during the batch, a further time interval D_3 follows.

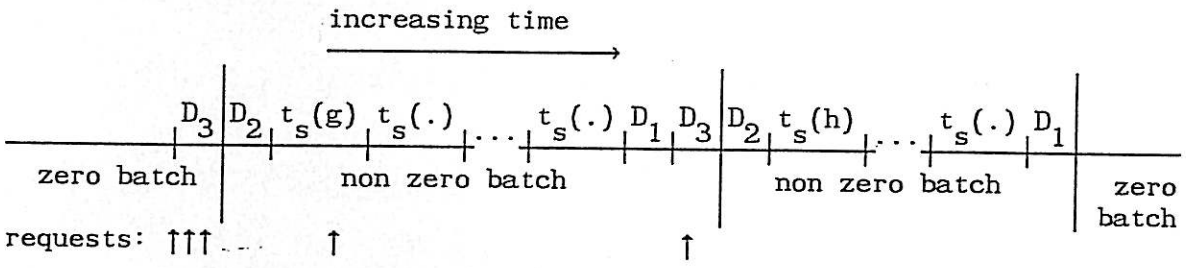


FIGURE 4.5 Batch Transitions of the fixed Priority Batched Arbiter Model.

Example

The arbiter presented in this example is designed for three requesters labelled 1, 2, and 3. The highest priority is given to 1 then the next priority to 2 and 3 is the lowest. The request pattern determines the behaviour of the model. Figure 4.6 illustrates a sequence of events.

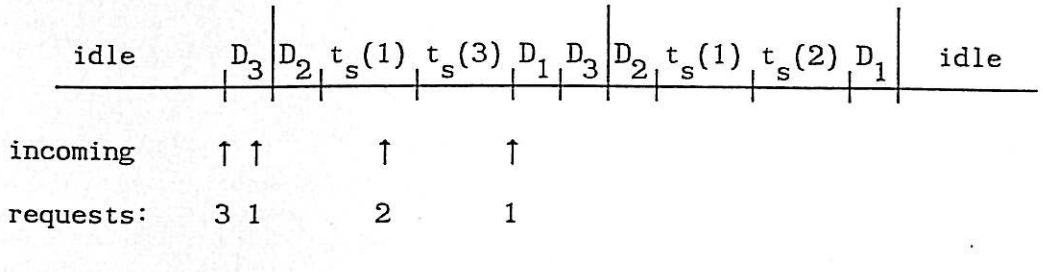


FIGURE 4.6 Example Model Behaviour.

Definitions of time durations

D_1 occurs after all batched requests have been serviced.

D_2 occurs after requests are batched. Requests lodged during D_2 wait until after all currently batched requests have been serviced before being batched themselves.

D_3 occurs at the end of all batches preceeding a non zero batch. D_3 is generated only if at least one request occurs before the end of D_1 in a non zero batch, otherwise a zero batch follows D_1. In a zero batch D_3 occurs as soon as a request occurs.

$t_s(h)$ corresponds to the time duration in which requester h is serviced and holds the resource (Ack h high).

The time duration $t_s(h)$ correponds to Ack h being high. The order $t_s(h)$ occurs within a batch is determined by h, with the smallest index h occurring first. Other batched disciplines could be modelled with the same model structure and the appropriate order of servicing within a batch.

For every batched request, a corresponding $t_s(h)$ appears in the batch. The service times, $t_s(h)$, occur consecutively after D_2 in order of priority.

Remarks

(i) The case of $D_1 = D_2 = D_3 = 0$ is not excluded.

(ii) There are 2^k possible different batches, where k is the number of requesters.

(iii) The composition of time duration in a batch is a function of not only the requesters serviced during the batch, but also whether a request occurs before D_1 ends, because the existence of D_3 depends on such a request. (This was ignored in counting batches in remark (ii) above).

(iv) A *singleton batch* refers to a batch in which only one request is serviced. Suppose $D_1 = 0$, then the last requester serviced in a batch cannot form a singleton batch immediately following, without an intevening zero batch. (It is assumed that requester h requests a non zero time after its last $t_s(h)$).

(v) If $D_1 = D_3 = 0$, the lowest priority requester serviced in a batch cannot be serviced in the following batch.

(vi) A *full batch* refers to the batch in which all requesters are serviced. If $D_1 = D_3 = 0$, remarks (iv) and (v) imply the full batch never occurs.

4.3.3 Re-Request Time and Service Time Modelling

In both the batched and non batched analyses presented in Chapters 5 and 6 respectively, the following assumptions are made concerning the requester execution of the models. The first is a result of the signal convention:

(i) A requester is a finite request source of size 1. (That is, at most one request can be lodged by a requester and must be held until acknowledged).

(ii) Re-request times are independently and exponentially distributed:

$$\text{prob(no request in } [0,t] = e^{-\lambda_h t} \tag{4.1}$$

where $\frac{1}{\lambda_h}$ is the mean time to request or alternatively, λ_h is the mean request rate from requester h when Req. h = 0.

(iii) Service times are independently distributed with a distribution function f_h for requester h

$$\text{prob}(t_a \leq t_s(h) \leq t_b) = \int_{t_a}^{t_b} f_h(t)dt \tag{4.2}$$

where the mean of $t_s(h)$ is denoted $\frac{1}{\mu_h}$.

4.4 MARKOV APPROACH TO ANALYSIS OF NON BATCHED ARBITERS

The approach of imbedded Markov chains can also be applied to non batched arbiters. The difference with respect to the batched case is in the location of the regeneration points which mark the discrete state transition times for the Markov chain. In an ideal non batched arbiter with no inter-service delays, the regeneration points occur at the start of each service time or idle period. At these times in an M/G queueing system, all memory due to the non Markovian service distribution is "flushed" and the process is renewed in the sense that the memory starts again from the regeneration point. In a non ideal arbiter with non zero inter-service times the regeneration points can be defined at the start of a service phase where the arbiter locks out consideration of further requests in order to arbitrate. The regeneration point at the start of an idle period occurs at the latest time such that if a request were to occur at that time no extra time would be introduced in the state transition to the next service. These points are considered in more detail in the example presented in the next section.

4.4.1 Example of an Asynchronous Non Batched Fixed Priority Arbiter

The example presented in this section is the asynchronous daisy chained distributed fixed priority arbiter described in Chapter 2. The module interconnection and circuit details are shown in Figure 4.7. Since the arbiter design is similar to that of the distributed design in Section 4.3.1, a description is given only in terms of the differences between the designs.

An extra distributed wired-or line, $\overline{\text{BUSY}}$, is employed to reset all module flip-flops, except the one currently being serviced (and $\overline{\text{REQ PEND}}$ corrresponds closely to the common line of Figure 4.1). The arbiter then acts like the batched version, except batches are limited to a size of at most one service due to the resetting action of $\overline{\text{BUSY}}$. A filter corresponding to τ_4 on the $\overline{\text{REQ PEND}}$ wired-or line, is not needed for the $\overline{\text{BUSY}}$ line since at most one open collector driver ever is active, preventing spurious pulses [G.5] when a driver effectively releases current into the line when another driver holds it low. The timing of the arbiter is shown in Figure 4.8 , which shows transitions from an idle state to a service state and to another service state. Reverse edge delays are neglected in the timing for simplicity. The model of the timing shown at the bottom of Figure 4.8 is almost identical to the batch model presented in Section 4.3.2, except that batches are always of length at most one here.

Since a time duration D_2 always occurs at the start of a service, it can be modelled as within the service time by adding an offset to every service time distribution. (This assumption must be remembered when calculating utilisation performance measures if a non zero D_2 is to be taken into account. A non zero D_2 is not taken into account when comparing non batched disciplines in this thesis.) Another simplification

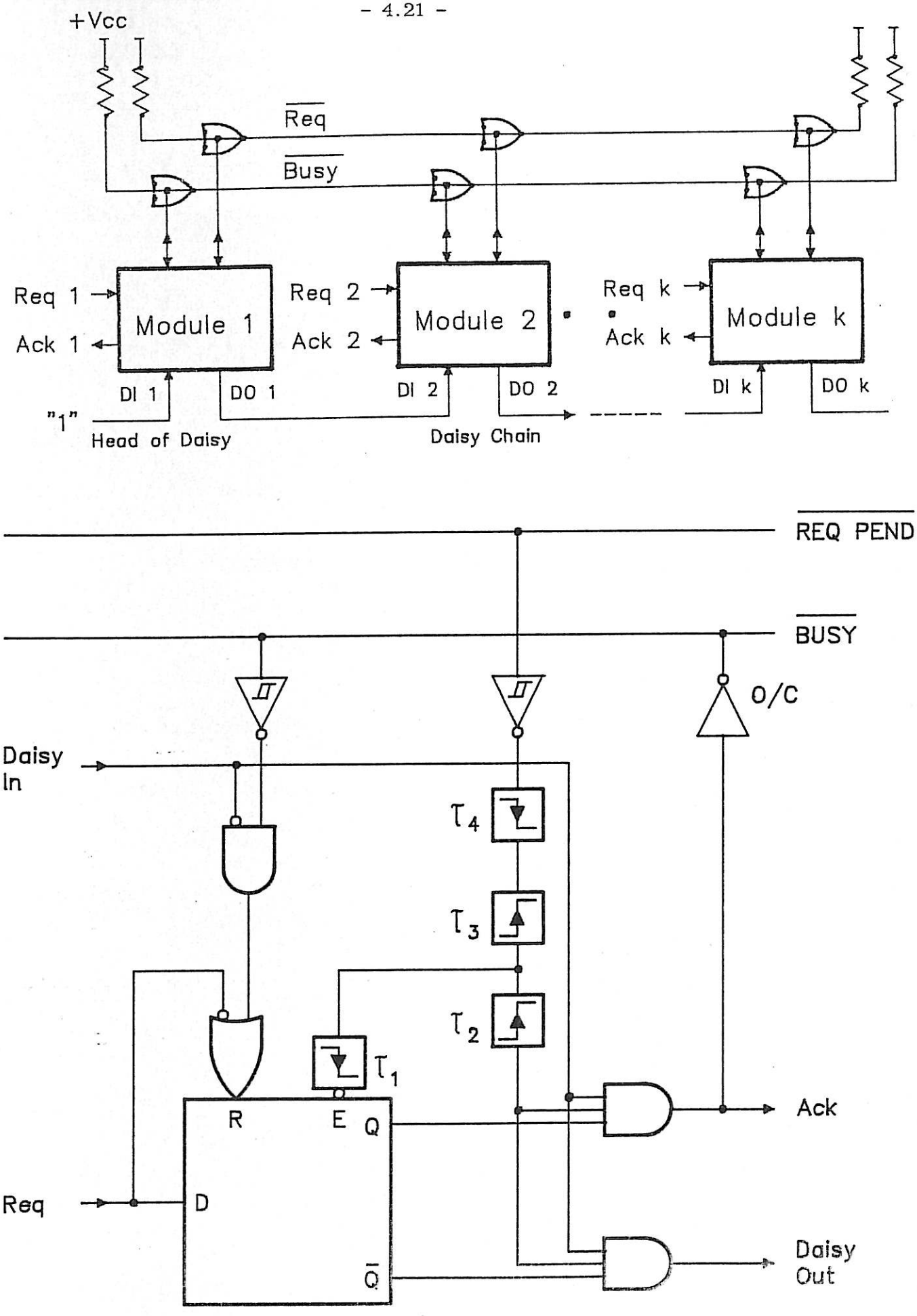


FIGURE 4.7 Module Interconnections and Circuit Details of the Distributed Daisy-Chained Fixed Priority Arbiter.

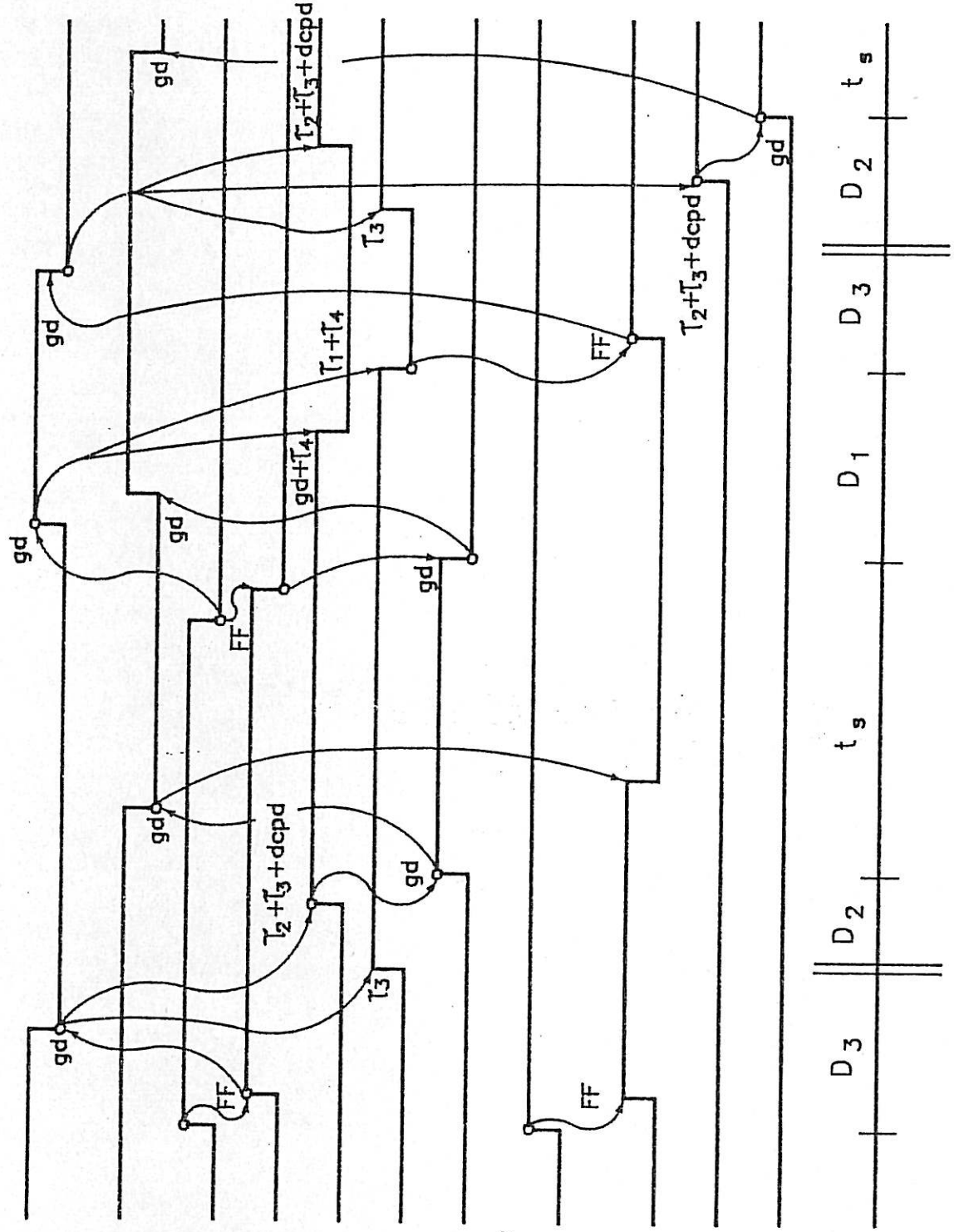


FIGURE 4.8 Timing of the Fixed Priority Daisy-Chained Arbiter.

to the model is the lumping together of D_1 and D_3 into one duration. This is an approximation, since when an idling period follows a service, the time duration D_3 is absent. However, the model behaviour will be identical when no request occurs within a time D_3 into the idling period as shown in Figures 4.9 and 4.10.

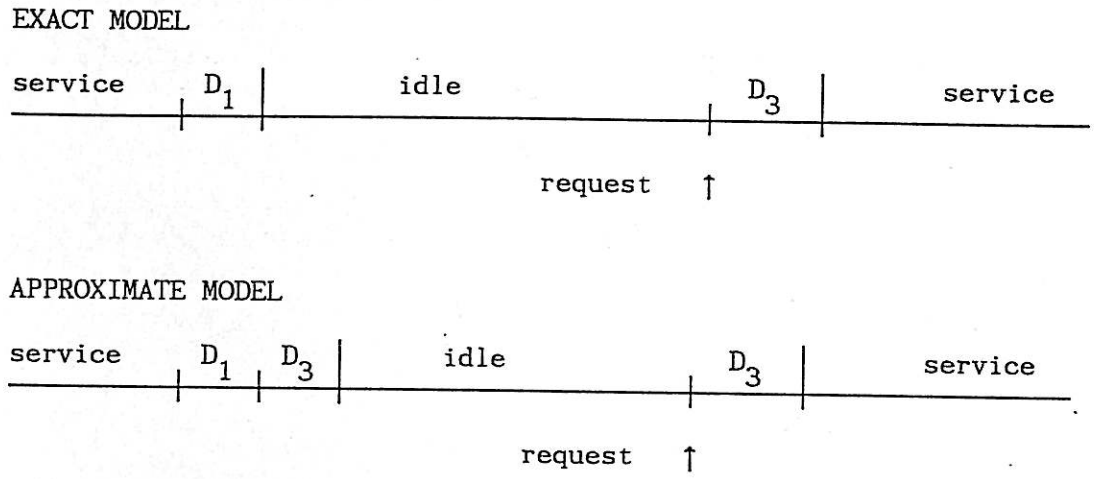


FIGURE 4.9 No Difference in Service Timing.

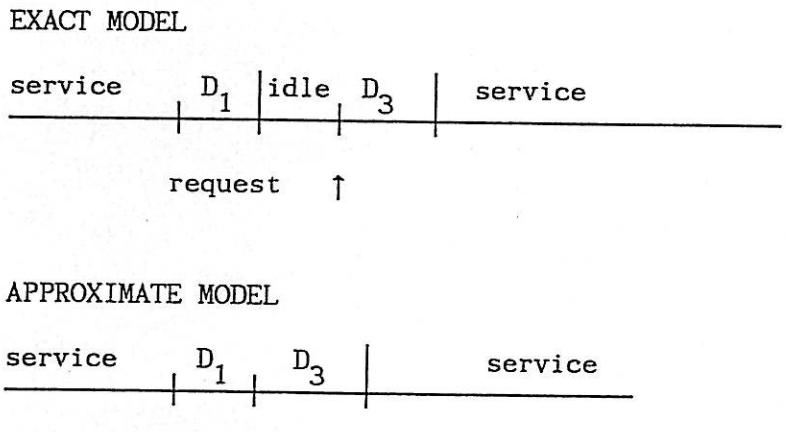


FIGURE 4.10 Disparity in Service Timing.

The models will behave approximately the same because:

(i) under heavy request loading, very few idling periods will occur;

(ii) under light request loading, idle periods will usually be longer than D_3;

(iii) at intermediate request loading, the difference will be slight because D_3 is usually much smaller than the mean time to next request.

When the above simplifications are acceptable, the general model described in the next section can be employed.

4.4.2 Non Batched Asynchronous Arbiter Model

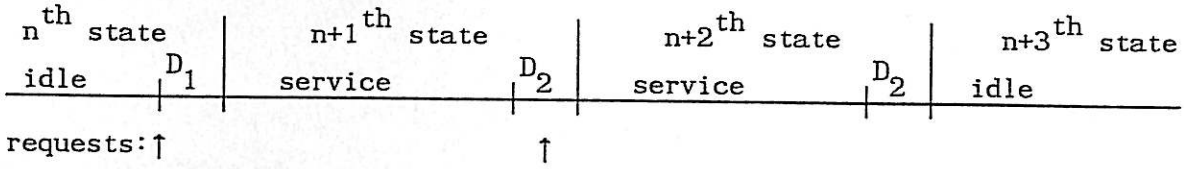


FIGURE 4.11 Simple Non Ideal Arbiter Model.

The model of a non ideal non batched arbiter incorporating non zero inter-service times is illustrated in Figure 4.11. A time duration D_1 occurs immediately a request is lodged during an idle period. Further requests lodged during D_1 are considered by the arbitration discipline for service in the next state. Requests lodged after D_1 are considered at the end of the following D_2 for service in the next state. At most one request is serviced in a state. It then follows that a request will not always be serviced in the state following the request. Also, an idling state need not follow a state in which no request occurred. (This is not the case in the batched arbiter model of Section 4.3.2).

Note that the non batched discipline of the model can be completely general and the model only describes the non ideal non batched nature of the arbiter.

Remarks

(i) If $D_2 = 0$, a requester cannot be serviced in consecutive states.

(ii) If $D_1 = 0$, the requester first to end an idle period is serviced in the following state.

(iii) Modelling of a time period at the start of a service can be incorporated into the service time as described in the previous section.

(iv) The re-request times and service times are modelled as described in Section 4.3.3. The service time distribution may be altered with an offset in line with remark (iii) above.

4.5 MONTE-CARLO ANALYSIS APPROACH

Most literature concerning the analysis of arbiters other than FCFS employ a Monte-Carlo analysis approach [B.1, H.2, K.13, S.1] due to the apparent difficulty and limitations of a theoretical approach. The Monte-Carlo approach involves generating random re-request and service times to excite a model of the arbiter which simulates the behaviour in a practical application. The mass of information generated by the excitation is condensed into appropriate performance estimates. Naturally, a large number of samples is necessary to obtain a good estimate of a performance measure of the arbiter. The simulation process is a well known and powerful tool, which can be used to obtain results and insights when utilised properly.

The advantage of a simulation over a more formal analysis approach is that complex models can be employed that are aimed at more closely

modelling the real world. For example, one is no longer restricted to an exponential re-request time distribution , and phenomena such as bursty arrivals or a uniform distribution of arrival times could be employed. Models can be quickly developed and coded on a computer to generate large amounts of data. Complex performance measures can also be employed.

The advantages of a Monte-Carlo analysis can be easily abused by simulating a problem prematurely without much thought. Large amounts of computer time can be expended generating results of little significance due to a possible misunderstanding of the fundamental behaviour of the system. It is often too easy to crunch first and sort out the mess later, when a more formal analysis may reveal simple and often enlightening characteristics that were overlooked in the dash for numerical results.

A simulation is essentially an experimental device that generates one data point of a performance measure every run. In order that results be meaningful, statistical considerations should be taken into account, such as the number of samples needed to accurately generate a point. Often, an expensively large number of iterations is necessary to obtain useful data, and even more to reveal relationships. Limiting results, as parameters tend to infinity for example, may not even be possible. A simulation cannot prove a result beyond any doubt, but can disprove a conjecture analogous to a counter example in mathematics. For these reasons simulation may best be employed only after other more formal approaches have been explored. It is frequently the case that once a theoretical foundation is established, simulation can be based on that foundation as is the case for the study of metastable failure rate of arbiters presented in the following chapters.

Despite all, Monte-Carlo techniques are employed in this thesis, along with the theoretical approaches, for the following reasons:

(i) They allow verification of precise theoretical results, to give confidence in theoretical findings and derivations.

(ii) Performance measures and disciplines too complex to be handled theoretically can be generated.

(iii) Conjectures can be tested, for example whether a discipline has a larger STDW than the batched version.

The results generated by the author are expensive in computation time (up to 100 000 iterations per point) compared with the corresponding theoretical results.

4.6 CONCLUSIONS

The various approaches that can be taken in the analysis of arbiters have been introduced in this chapter. The limitations and advantages of the methods have been discussed. The lack of suitability of standard queueing theory results applied to the problem of analysing arbiters with non ideal inter-batch/service delays and general service time distributions has been highlighted, and the author's alternative approach involving the use of imbedded Markov chains has been introduced. The models employed in the analysis have been motivated by practical examples of arbitration circuits. Assumptions involving requester excitation assumptions have been discussed and defined precisely for later use in Chapters 5 and 6. Although exponential service time distributions may enable a simplified analysis, it has been argued this may be too restrictive in many practical applications, and consequently general service time distributions are allowed in the author's analysis.

CHAPTER 5

ANALYSIS OF ASYNCHRONOUS BATCHED FIXED PRIORITY ARBITERS

5.1 INTRODUCTION

The model introduced and defined in Chapter 4 for an asynchronous batched fixed priority arbiter is analysed in this chapter. The aim of the chapter is to not only analyse this class of arbiters, but also to present the analysis technique in a manner that enables generalisation and application to other problems. The analysis of the fixed priority non batched arbiter of Chapter 6 employs similar techniques to those employed in this chapter. To the author's knowledge the results and techniques are new in the context of arbiters. They are also presented in [4].

In Section 5.2, the state of the arbiter is defined in terms of the set of requests pending at a batching point. With this state definition the model is shown in Section 5.3 to be Markovian under the requester excitation assumptions (see Section 4.3.3). The state transition probabilites are derived in Section 5.4 for the discrete time Markov chain representing the arbiter. Two special cases of the general service distribution are considered : constant and exponential. These represent the two extremes of deterministic and completely random behaviour (intermediate distributions known as Erlang distributions bridge the gap [K.12]) and can be used to explore sensitivity of the model to different service time distributions. Limiting properties, both in time (steady state) and request loading, of the state transition probabilites are considered in Section 5.5, where it is shown that unique steady state

probabilities exist for the states independent of the initial state. The request loading limits highlight some interesting characteristics of batched arbiters. Performance measures are precisely defined in Section 5.6 in terms of the steady state limiting probabilities of the arbiter state. Request loading limits are also examined for the performance measures. To examine intermediate request loadings between zero and heavy saturation, a computer study is performed to calculate steady state performance measures. Numerical results relevant to points discussed in the chapter are presented in Section 6.6, where limiting request loading results derived in Section 6.5 are observed in the results. Conclusions concerning batched arbiters are presented in Section 6.7.

5.2 STATE DEFINITION

The arbiter behaviour is characterised by a sequence of batches in time. An idling period corresponds to a zero batch. Each batch has an associated *state* determined by the set of requesters serviced in the batch (or the set of pending requests at a batching point). The state of the n^{th} batch, $S(n)$, is defined as an integer in the range $0 \,..\, 2^k-1$, where k is the total number of requesters. $S(n)$ is a function of the n^{th} batch as follows.

Define v_n: $\{1, \ldots, k\} \rightarrow \{0,1\}$ by

$$v(h) \triangleq \begin{cases} 1 \;, & \text{if requester } h \text{ is serviced in the } n^{th} \text{ batch, denoted } h \in S(n) \\ 0 \;, & h \notin S(n) \end{cases}$$

Then

$$S(n) \triangleq \sum_{h=1}^{k} v(h)2^{h-1} \qquad (5.1)$$

For example, the zero batch has state zero and the full batch has state 2^k-1. There is a one to one correspondence between the set of requesters serviced in a batch and the state of the batch. Consequently, set operations are used on state variables as though they are sets such as the notation $h \in S(n)$, which means requester h is serviced in the state $S(n)$.

5.3 MARKOV PROPERTY

The sequence of states of the arbiter model with requesters obeying assumptions (i), (ii) and (iii) of Section 4.3.3 forms a finite state Markov chain. That is

$$\text{prob}\left[S(n) = 1 \;\middle|\; S(n-1) = j_1,\ S(n-2) = j_2,\ \ldots,\ S(0) = j_n\right] \qquad (5.2)$$

is independent of $j_2, \ldots, j_n$.

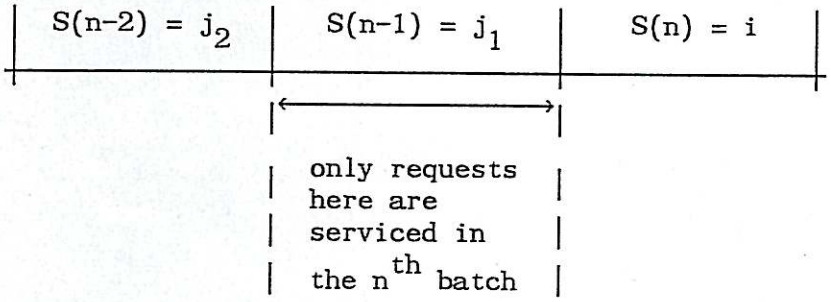


FIGURE 5.1 Markov Property of Arbiter Model.

Figure 5.1 illustrates that the state of the n^{th} batch is determined only by requests made during the $(n-1)^{th}$ batch. The probability that a particular requester requests during the $(n-1)^{th}$ batch is only a function of the $(n-1)^{th}$ batch's configuration and the request rates. This is true because the probability of a request is independent of the

requester's history when it has no request already pending. Any request pending at the start of the $(n-1)^{th}$ batch is reflected by the servicing of that request during the $(n-1)^{th}$ batch. Thus, all the information that determines (5.2) is contained in the states i and j_1. Hence, it suffices to write $\text{prob}[S(n) = i \mid S(n-1) = j_1]$ as a shorthand for (5.2).

5.4 DERIVATION OF STATE TRANSITION PROBABILITIES

In order to derive state transition probabilities, some preliminary atomic expressions are introduced with which final results can be expressed. Firstly, the probability that requester h does not request during the service time of requester ℓ is derived under the condition that requester h has no request pending already.

$$\text{prob}\left[\begin{array}{l|l}\text{requester h does not} & \text{Req h = 0 at the} \\ \text{request during } t_s(\ell) & \text{start of } t_s(\ell)\end{array}\right]$$

$$= \int_0^\infty f_\ell(t) e^{-\lambda_h t} dt \qquad (5.3)$$

Consider the following conditional event: Requester h not requesting during the service of requester ℓ_1, followed by the servicing of requester ℓ_2 and finally a fixed time D. The probability of this conditional event is given by:

$$\int_0^\infty \int_0^\infty f_{\ell_1}(t_1) f_{\ell_2}(t_2) e^{-\lambda_h(t_1 + t_2 + D)} dt_1 dt_2$$

$$= e^{-\lambda_h D} \int_0^\infty f_{\ell_1}(t_1) e^{-\lambda_h t_1} dt_1 \int_0^\infty f_{\ell_2}(t_2) e^{-\lambda_h t_2} dt_2 \qquad (5.4)$$

The joint density function of the service times of requesters ℓ_1 and ℓ_2 is the product of their individual service time density functions since service times are assumed to be independent. The decomposition of (5.4) generalises to any number of time intervals. Consider now the probability of requester h not having a request pending at the end of D_1 in a batch with state $i \neq 0$. This probability is denoted by $Q(i, h)$. Two cases arise:

(i) requester h is serviced in the batch with state i ($h \in i$);

(ii) requester h is not serviced ($h \notin i$) and hence Req h = 0 at the start of the batch.

Figure 5.2 illustrates the definition of $Q(i, h)$ when $h \in i$.

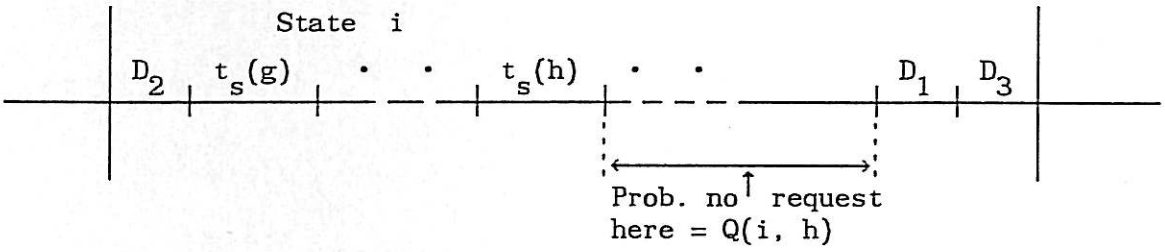


FIGURE 5.2 Definition of $Q(i, h)$, $h \in i$.

It follows that

$$Q(i,h) = \begin{cases} e^{-\lambda_h D_1} \prod_{\substack{\ell \in i \\ \ell > h}} \int_0^\infty f_\ell(t) e^{-\lambda_h t}\, dt, & \text{if } h \in i \\ e^{-\lambda_h (D_1 + D_2)} \prod_{\ell \in i} \int_0^\infty f_\ell(t) e^{-\lambda_h t}\, dt, & \text{if } h \notin i \end{cases} \qquad (5.5)$$

where

$$\int_0^\infty f_\ell(t)e^{-\lambda_h t}dt = \begin{cases} \dfrac{\mu_\ell}{\mu_\ell+\lambda_h} & \text{, exponentially distributed service time} \\ e^{-\lambda_h/\mu_\ell} & \text{, constant service time} \end{cases}$$

The state transition probabilities are now derived for four cases of S(n-1) and S(n) being zero or non zero :

Case 1: S(n-1) = S(n) = 0

$$\text{prob}[S(n) = 0 \mid S(n-1) = 0] = 0 \qquad (5.6)$$

Equation (5.6) follows from the fact that a non zero batch always precedes and follows a zero batch. The zero batch is defined to be the idle time between two non zero batches and two consecutive zero batches cannot exist. Refer to Figure 5.3.

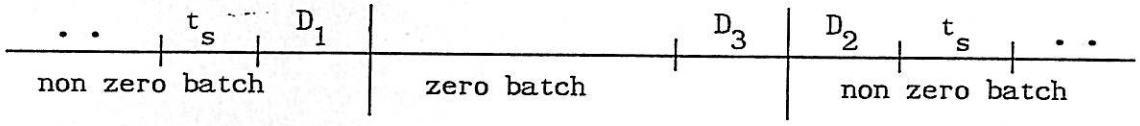


FIGURE 5.3 Zero Batch between Non Zero Batches.

Case 2: S(n-1) = j ≠ 0, S(n) = 0

$$\text{prob}[S(n) = 0 \mid S(n-1) = j] = \prod_{h=1}^{k} Q(j,h) \qquad (5.7)$$

Equation (5.7) gives the probability of the intersection of the k independent events that each requester does not request during the $(n-1)^{th}$ batch and hence a zero batch follows.

Note that the time duration D_3 is not included in the $(n-1)^{th}$ batch because the n^{th} batch is the zero batch.

Case 3: $S(n-1) = 0, \quad S(n) = i \neq 0.$

$\text{prob}[S(n) = i \mid S(n-1) = 0]$ is required.

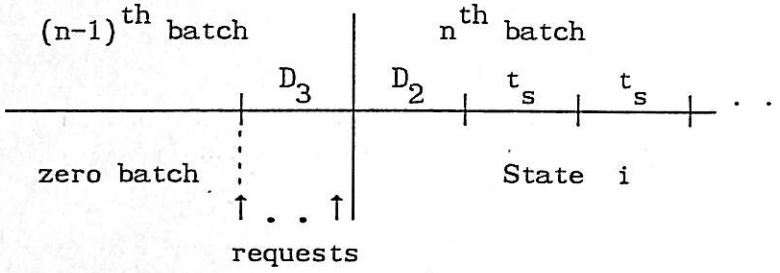


FIGURE 5.4 Zero Batch to Non Zero Batch Transition.

Considering Figure 5.4, for $S(n) = i$ to follow $S(n-1) = 0$, the first requester to end the zero batch by requesting must be in $S(n)$. All other requesters in $S(n)$ must request within the following D_3 time period. But so that no more requesters request, the intersection must be taken with the event that all requesters not in $S(n)$ do not request within D_3. Thus,

$$\text{prob}[S(n) = i \mid S(n-1) = 0]$$

$$= \sum_{h \in i} \left\{ \text{prob}\left[\begin{array}{l} h \text{ is the first requester} \\ \text{to request during} \\ S(n-1) = 0 \end{array} \;\middle|\; S(n-1) = 0 \right] \right.$$

$$\times \text{prob}\left[\begin{array}{l} \text{remaining requesters} \in i \\ \text{request during } D_3 \end{array} \;\middle|\; S(n-1) = 0 \right]$$

$$\left. \times \text{prob}\left[\begin{array}{l} \text{requesters} \notin i \text{ do not} \\ \text{request during } D_3 \end{array} \;\middle|\; S(n-1) = 0 \right] \right\} \qquad (5.8)$$

Since the events of requesters requesting first are mutually exclusive, the summation applies in (5.8). The terms in (5.8) are shown in Appendix B to give :

$$\text{prob}[S(n) = i \mid S(n-1) = 0] = \frac{\sum_{h \in i} \left\{ \lambda_h \cdot \prod_{\substack{g \in i \\ g \neq h}} \left[1 - e^{-\lambda_h D_3} \right] \right\} \cdot \prod_{f \notin i} e^{-\lambda_f D_3}}{\sum_{\ell=1}^{k} \lambda_\ell} \tag{5.9}$$

Case 4: $S(n-1) = j \neq 0$, $S(n) = i \neq 0$

$$\begin{aligned} &\text{prob}[S(n) = i \mid S(n-1) = j] \\ &= \text{prob}\left[\begin{array}{l} \text{requesters } \in i \text{ request during} \\ \text{the } (n-1)\text{th batch and at} \\ \text{least one of which before } D_3 \end{array} \;\middle|\; S(n-1) = j \right] \\ &\times \prod_{h \notin i} \text{prob}\left[\begin{array}{l} \text{requester h does not request} \\ \text{during the } (n-1)\text{th batch} \end{array} \;\middle|\; S(n-1) = j \right] \end{aligned} \tag{5.10}$$

The reason for the inclusion of the condition "at least one of which before D_3" is explained with the aid of Figure 5.5.

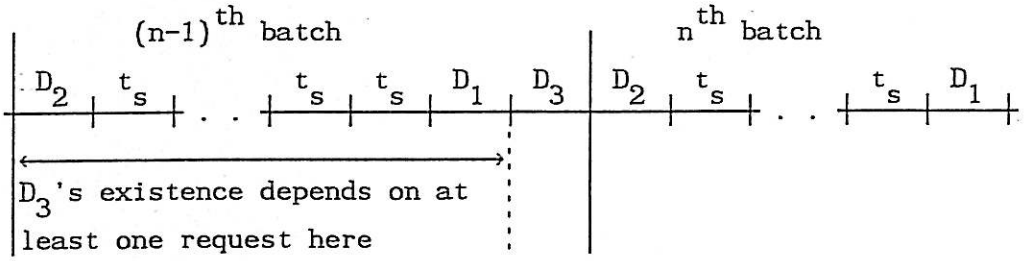


FIGURE 5.5 Non Zero to Non Zero Batch Transition.

If no request occurred up to the end of D_1, then D_3 would not be present in the $(n-1)^{th}$ batch and a zero batch would follow.

The terms in (5.10) are derived in Appendix C to give :

$$\text{prob}[S(n) = i \mid S(n-1) = j]$$

$$= \left\{ \prod_{h\in i} \left[1-Q(j,h)e^{-\lambda_h D_3}\right] - \prod_{h\in i} \left[Q(j,h)(1-e^{-\lambda_h D_3})\right]\right\} \prod_{h\notin i} Q(j,h)^{-\lambda_h D_3} \tag{5.11}$$

Equations (5.6), (5.7), (5.9) and (5.11) define the Markov transition matrix for the batch states of the system. Notice that in all cases $\text{prob}[S(n) = i \mid S(n-1) = j]$ is independent of n and hence the Markov chain is homogeneous. Define

$$p_{ij} \triangleq \text{prob}[S(n) = i \mid S(n-1) = j] \qquad i,j = 0, \ldots, 2^k-1 \tag{5.12}$$

and the ***probability transition matrix***, P, is defined by:

$$P \triangleq [p_{ij}] \tag{5.13}$$

5.5 LIMITING PROPERTIES OF THE PROBABILITY TRANSITION MATRIX

The probability transition matrix, P, is a non negative matrix (i.e. *all* elements non negative) with unity column sums. The probability vector of the n^{th} batch, $p(n)$, represents with its i^{th} component, $p_i(n)$, the probability of the n^{th} batch being in state i. The probability vector $p(n)$ is given by :

$$p_i(n) = \sum_{j=0}^{m-1} \text{prob}[S(n) = i \mid S(n-1) = j] \cdot p_j(n-1)$$

$$= \sum_{j=0}^{m-1} p_{ij} \cdot p_j(n-1) \tag{5.14}$$

where $m \triangleq 2^k$. In matrix notation

$$p(n) = P \cdot p(n-1) = P^n \cdot p(0) \tag{5.15}$$

where $p(0)$ is the initial probability vector. Furthermore, the *limiting probability vector* p is defined as:

$$p \triangleq \lim_{n\to\infty} p(n) \qquad \text{(when the limit exists)}$$

$$= \lim_{n\to\infty} P^n p(0) = P \lim_{n\to\infty} P^{n-1} p(0) = Pp \tag{5.16}$$

Note that (5.16) shows that p is an eigenvector of P corresponding to the eigenvalue of 1.

5.5.1 Principle Limiting Results as $n\to\infty$

Theorem 5.1: The probability transition matrix of the fixed priority batch arbiter model, P, has a unique positive limiting probability vector independent of the initial probability vector, provided $\mu_h, \lambda_h > 0$ for $h = 1, \ldots, k$ and $D_1 + D_3 > 0$.

Proof: Firstly it is shown that P is irreducible and primitive [G.1] by showing that all elements of P^2 are positive (written $P^2 > 0$). This follows from the theorem:

> P is irreducible and primitive if and only if some power of P is positive [G.1, p.80].

To show $P^2 \triangleq \left[p_{ij}^{(2)}\right]$ is positive, consider :

$$p_{ij}^{(2)} = \sum_{u=1}^{m-1} p_{iu}p_{uj}$$

$$\geq p_{iw}p_{wj}, \qquad \text{for some } w,\ 0 \leq w \leq m-1 \qquad (5.17)$$

since $p_{iu}p_{uj} \geq 0$ for all i, u, j. Therefore if, for every i and j, a w can be chosen such that $p_{iw}p_{wj} > 0$ then it follows that $P^2 > 0$. The state w corresponds to an intermediate state between i and j as shown in Figure 5.6.

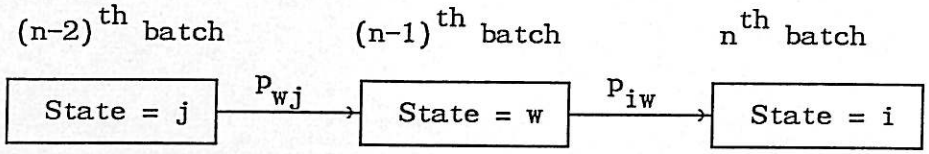


FIGURE 5.6 State Transition Illustrating the Proof of Theorem 5.1.

In Appendix D it is shown that such an intermediate state w can be chosen when $D_1 + D_3 > 0$.

When $D_1 = D_3 = 0$, the full batch can never occur because all transitions to it have zero probability (including the full batch to full

batch transition). This implies that P is reducible in the case of $D_1 = D_3 = 0$, since P has an invariant coordinate subspace of dimension $2^k-1 < 2^k$ = dimension of P, consisting of all but the full batch state.

In order to study the zero inter-batch time case, the matrix, P_1, is defined as P with the last row and column removed (i.e. without full batch state):

$$P_1 \triangleq [p_{ij}] \qquad i,j = 0, \ldots, m-2 \tag{5.18}$$

P_1 has the same properties as P in Theorem 5.1. Note that P_1 is a probability transition matrix (i.e. column sums are unity) only when $D_1 = D_3 = 0$. The following theorem is proved in Appendix D.

Theorem 5.2: The probability transition matrix P_1 as defined in (5.18) has a unique positive limiting probability vector independent of the initial probability vector provided $\mu_h, > 0$ for $h = 1, .., k$ and $D_1 = D_3 = 0$.

5.5.2 Light Request Loading Limit of the Probability Transition Matrix

In this section the limiting behaviour of P as the request rates tend to zero is examined. The manner in which the request rates tend to zero is defined as follows :

The average request rate, λ, is defined:

$$\lambda \triangleq \frac{\sum_{f=1}^{k} \lambda_f}{k} \tag{5.19}$$

and relative request rates, r_h, are defined:

$$r_h \triangleq \frac{\lambda_h}{k\,\lambda} \qquad h = 1, 2, \ldots, k \tag{5.20}$$

The request rates are restricted in the following discussion so that all r_h are constant as λ varies. From (5.5) it follows that

$$\lim_{\lambda \to 0} Q(i, h) = 1 \tag{5.21}$$

From (5.7)

$$\lim_{\lambda \to 0} p_{0j} = 1, \qquad \text{for} \quad j = 1, 2, \ldots, m-1 \tag{5.22}$$

That is, a zero batch follows every non zero batch. From (5.11) and (5.21) it follows that:

$$\lim_{\lambda \to 0} p_{ij} = 0 \qquad \text{for} \quad i,j = 1, 2, \ldots, m-1 \tag{5.23}$$

From (5.9) and (5.21) it follows that:

$$\lim_{\lambda \to 0} p_{i0} = \begin{cases} 0 \ , & |i| > 1 \text{ i.e. contains more than one requester} \\ r_g \ , & i = \{g\} \text{ i.e. singleton state of requester } g \end{cases} \tag{5.24}$$

Define:

$$P_0 \triangleq \lim_{\lambda \to 0} P \tag{5.25}$$

It can be seen from (5.23) and (5.24) that

$$P_0 = P_0^3 \tag{5.26}$$

and it follows that the Markov chain is cyclic with period 2. Note also that P_o is independent of D_1, D_2, D_3 and μ_h.

The limiting probabilities of the 2-cycle are dependent on the initial state with the limiting values reached after the first iteration. In physical terms the arbiter alternates between idling and servicing one requester. Obviously as $\lambda \to 0$, the length of the zero batch $\to \infty$. For small request rates compared with delays and service times, the arbiter behaves like the limiting case.

5.5.3 Heavy Request Loading Limits of the Probability Transition Matrix

The request rates are allowed to approach infinity keeping r_h of (5.20) constant for each requester. As request rates become large, requesters lodge a new requests soon after releasing the resource. The limiting behaviour depends on whether $D_1 + D_3$ is zero and two cases are considered below:.

Case 1: $D_1 + D_3 > 0$.

From (5.6) and (5.7) it follows that

$$\lim_{\lambda \to \infty} p_{0j} = 0 \qquad \text{for } j = 0, 1, \ldots, m-1 \tag{5.27}$$

If $D_3 > 0$, then from (5.9)

$$\lim_{\lambda \to \infty} p_{i0} = \begin{cases} 0 & , \text{ if } i \neq m-1 \\ 1 & , \text{ if } i = m-1 \end{cases} \tag{5.28}$$

and if $D_3 = 0$, then from (5.9)

$$\lim_{\lambda\to\infty} p_{i0} = \begin{cases} r_f & , \text{ if } i = \{f\} \\ 0 & , \text{ if } |i| \neq 1 \end{cases} \tag{5.29}$$

From (5.11) for $j \neq 0$ and $i \neq 0$

$$\lim_{\lambda\to\infty} p_{ij} = \begin{cases} 0 & , |i| \neq k \text{ i.e. not full state} \\ 1 & , |i| = k \end{cases} \tag{5.30}$$

Summarising these results: for $D_3 > 0$

$$P_\infty(X,+) \triangleq \lim_{\lambda\to\infty} P = \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & & & 0 \\ \vdots & & & \vdots \\ 0 & & & 0 \\ 1 & & \dots & 1 \end{bmatrix} \tag{5.31}$$

The notation $P_\infty(X,+)$ needs some explanation. "X" means the argument is ≥ 0, and "+" means it is > 0. The first argument refers to D_1, whilst the second refers to D_3. For $D_3 = 0$:

$$P_\infty(+,0) \triangleq \lim_{\lambda\to\infty} P \tag{5.32}$$

In both cases the square of the probability transition matrix is the same.

$$[P_\infty(X,+)]^2 = [P_\infty(+,0)]^2 = P_\infty(X,+) \tag{5.33}$$

The limiting probability transition matrix has been reached in both cases in (5.33). Since all the columns of the limit, $P_\infty(X,+)$, are the same, the limiting probability vector is independent of the initial probability and is given by $[0 \quad 0 \quad \ldots \quad 0 \quad 1]^T$ i.e., the full state has probability one.

Case 2: $D_1 + D_3 = 0$, the full batch can never occur and the limiting behaviour is more interesting. From equation (5.5)

$$\lim_{\lambda\to\infty} Q(i,h) = \begin{cases} 1 & , \text{ if } h = \max\{\ell: \ell \in i\} \\ 0 & , \text{ otherwise} \end{cases} \tag{5.34}$$

and from (5.7) and (5.34)

$$\lim_{\lambda\to\infty} p_{0j} = 0 \qquad j = 0, 1, \ldots, m-1 \tag{5.35}$$

and from (5.10)

$$\lim_{\lambda\to\infty} p_{i0} = \begin{cases} r_f & , \text{ if } i = \{f\} \\ 0 & , \text{ if } |i| \neq 1 \end{cases} \tag{5.36}$$

For $i \neq 0$ and $j \neq 0$ (5.11) and (5.34) imply

$$\lim_{\lambda\to\infty} p_{ij} = \begin{cases} 1 & , \text{ if } i \text{ contains all requesters except the lowest priority requester in state } j. \\ 0 & , \text{ otherwise} \end{cases} \tag{5.37}$$

The powers of the probability transition matrix, $P_\infty(0,0)$, formed from (5.35), (5.36) and (5.37) cycle with a period of two after the third

power. The limiting probability vectors in the two cycle have all zero elements except for two states corresponding to the full batch without requester k and the full batch without requester k-1. The probabilities of these states exchange after each state transition and depend on the initial state. The physical interpretation of this result is that the arbiter allows no time at the end of a batch for the two lowest priority requesters k and k-1 in which to request before the batching point immediately following their servicing. This interesting situation can be seen later in the numerical results for heavy request loading and small values of $D_1 + D_3$.

5.6 PERFORMANCE PARAMETERS

It has been shown that the Markov chain model for the arbiter has a unique limiting probability vector for finite positive request rates. After sufficient time, the probability of a particular batch occurring will be constant regardless of the initial state of the arbiter. From the batch probabilities in the steady state, many useful performance parameters of the arbiter can be determined which are functions of the request rates, service times, the total number of requesters and inter-batch times.

5.6.1 Utilisation

Suppose the system has reached steady state with probability vector

$$p = [p_0, p_1, p_2, \ldots, p_{m-1}]^T \qquad (5.38)$$

Suppose n successive batches are sampled and let n_j $(j = 0, \ldots, m-1)$ be the number of states = j, with

$$n = \sum_{j=0}^{m-1} n_j \tag{5.39}$$

Let t_j be the mean length of time of a batch with state j, defined as follows:

$$t_j \triangleq \begin{cases} D_1 + D_2 + D_3 + \sum\limits_{h \in j} \dfrac{1}{\mu_h} & , j = 1, 2, \ldots, m-1 \\ \dfrac{1}{\sum\limits_{h=1}^{k} \lambda_h} & , j = 0 \end{cases} \tag{5.40}$$

The proportion of time devoted to the state j is given by

$$T_j^n = \frac{n_j t_j}{\sum\limits_{i=0}^{m-1} n_i t_i} \tag{5.41}$$

taking the limit as $n \to \infty$ of (5.41) gives

$$T_j \triangleq \lim_{n\to\infty} T_j^n = \frac{t_j \lim\limits_{n\to\infty} \frac{n_j}{n}}{\sum\limits_{i=0}^{m-1} t_i \lim\limits_{n\to\infty} \frac{n_i}{n}} \tag{5.42}$$

Equation (5.42) suggests defining the following utilisation parameter:

$$U_j \triangleq \frac{t_j p_j}{\sum_{i=0}^{m-1} t_i p_i} \tag{5.43}$$

Similarly the proportion of time spent servicing requester h, PROP(h), is defined by:

$$PROP(h) \triangleq \frac{\sum_{\substack{i \text{ such that} \\ h \in i}} \frac{p_i}{\mu_h}}{\sum_{i=0}^{m-1} p_i t_i} \tag{5.44}$$

The idling time for the system is defined as the proportion of time not spent servicing requests:

$$IDLE \triangleq 1 - \sum_{h=1}^{k} PROP(h) \tag{5.45}$$

5.6.2 Mean Waiting Time for Each Requester

The waiting time for requester h is the time from when requester h requests to when requester h receives the resource. In Appendix E an expression is derived for W(i, j, h), the mean waiting time for requester h given that S(n) = i follows S(n-1) = j, where requester h is serviced during state i. The unconditional mean waiting time for requester h, MWT(h), can then be expressed as:

$$MWT(h) = \lim_{n\to\infty} \sum_{j=0}^{m-1} \sum_{\substack{i \text{ such that} \\ h \in i}} W(i,j,h)\ prob[S(n-1) = j,\ S(n) = i \mid h \in S(n)]$$

(5.46)

$$= \sum_{j=0}^{m-1} \sum_{\substack{i \text{ such that} \\ h \in i}} \frac{W(i,j,h) \cdot p_{ij} \cdot p_j}{\sum_{\substack{u \text{ such that} \\ h \in u}} p_u}$$

5.6.3 Metastable Reliability Performance

The aperture model for metastable behaviour of synchronising elements, such as flip-flops and batches, discussed in Section 3.6.3 and Section 3.7.5 is employed in the metastable reliability analysis of this section. The analysis is applied to the distributed daisy chained batched arbiter described in detail in Section 4.3.1, however, the technique can be applied more generally to other implementations of batched arbiters.

The basic synchronising element of the arbiter module shown in Figure 5.7 is the latch.

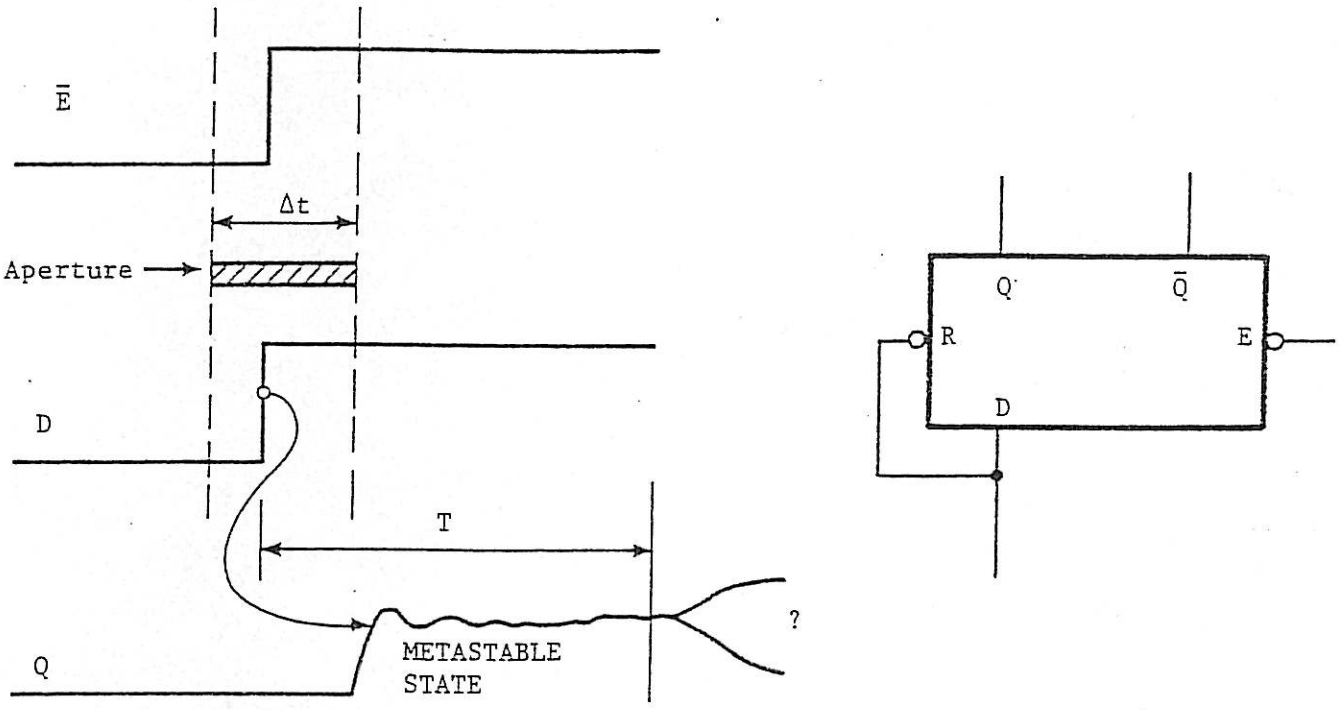


FIGURE 5.7 Aperture Model for a Latch.

The latch is transparent from D to Q when $\bar{E}$ is low, and the value of D is stored when $\bar{E}$ goes high (disabled). When the latch is in the process of being disabled an *aperture event* occurs. Referring to Figure 5.7, should the data input of the latch change during the aperture (of width Δt) then the outputs Q, $\bar{Q}$ will be indeterminate for a time duration exceeding T. The aperture event is the *presentation* of the aperture to the data and not the actual occurrence of a data change within the aperture. The exact position of the aperture with respect to the $\bar{E}$ rising edge is not critical in the analysis, with the width Δt determining the probability of an indeterminate output lasting at least a time T. The aperture width is a function of device dependent parameters and T as defined in equation (3.27).

For the metastable reliability analysis, *failure* of the arbiter is assumed to correspond to an Ack output being sensitised to a metastable latch output. (The same concept of sensitisation was employed in studying the combinational circuit of the redundant synchroniser in Section 3.7.5). Obviously, this concept of failure may not always correspond to external perceptions of failure of the arbiter circuit, such as simultaneous Ack signals asserted or undefined Ack signals. It is conceivable that an Ack output sensitised to a metastable latch may be interpreted as a valid and harmless 0 logic state by some devices most of the time, but this certainly cannot be guaranteed nor should be expected.

The definition of failure adopted here is a necessary condition for metastable behaviour to cause undesirable output disturbances, and is useful in worst case design and reliability estimation. The "worst case" situation is depicted in the timing diagram of Figure 5.8, which shows a request with priority higher than all pending requests occurring within its latch's aperture at a batching point - the only time aperture events can occur.

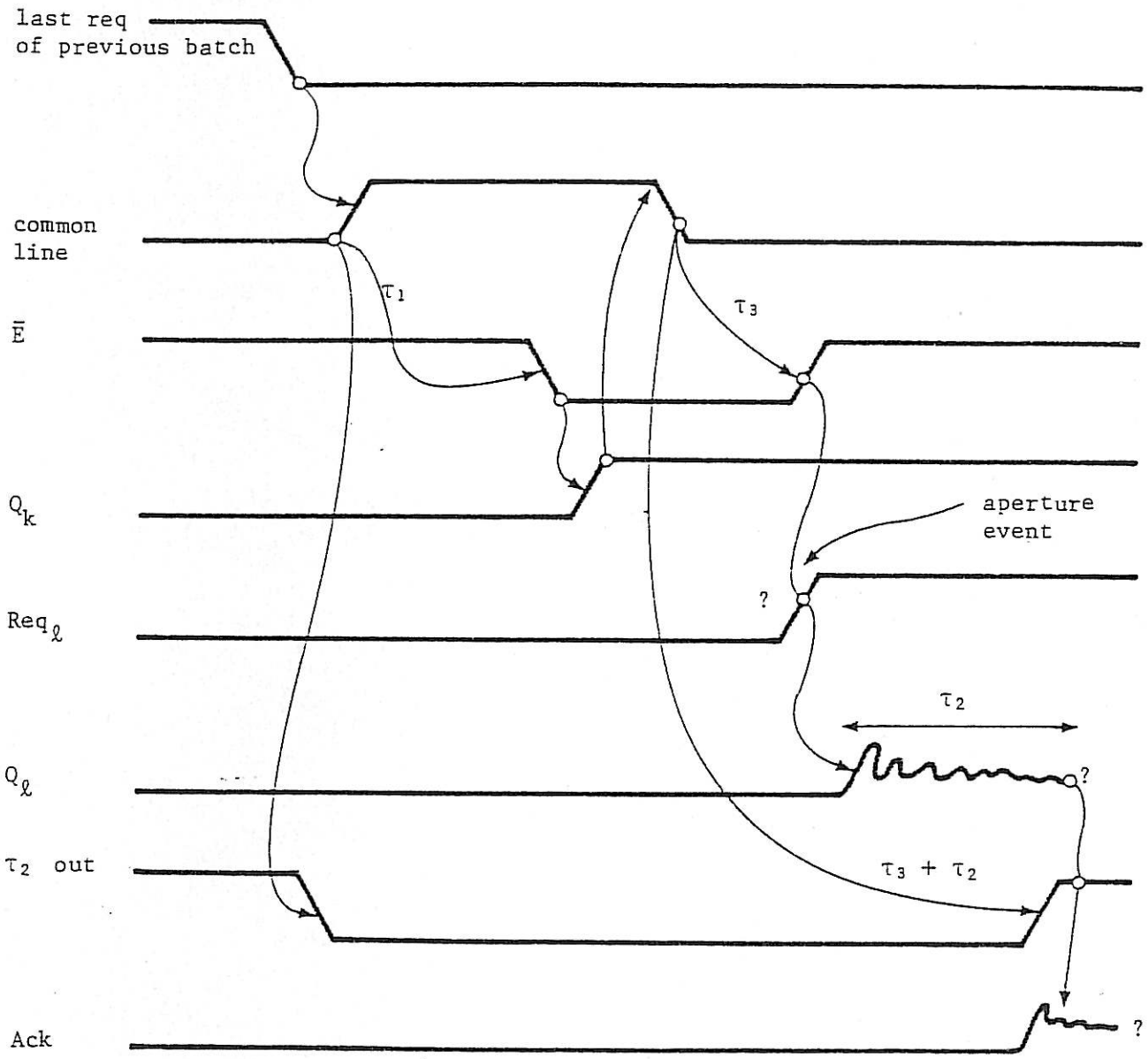


FIGURE 5.8 Metastable Failure of the Batched Daisy Chained Arbiter.

Because this requester's latch has a sensitised path to both Ack and Daisy Out signals, failure of the arbiter can result. The key point is that the request must be of sufficient priority so that its occurrence (at the batching point) has immediate impact on the decision as to which request to acknowledge next. A low priority request, below pending request priorities cannot cause the arbiter to change its next decision. The arbiter is sensitised after the allocated metastable settling time, approximately τ_2, only to latches of priority higher than all pending requests. A low priority metastable latch will be sensitised only *after* all higher priority batched requests have completed service, but this necessitates an extremely long metastable duration which is of insignificant probability due to the exponential dependence on settling time. Thus, the aperture events associated with modules of priority lower than existing requests are justifiably ignored in the analysis.

The fixed priority batched and non batched arbiter models described in Chapter 4 are ideal in the sense that they implicitly assume that metastable failure never occurs. The metastable failure rate is derived from the ideal models by calculating the mean rate at which aperture events occur and the probability requests occur within these apertures. It is assumed in the analysis that all latches have identical synchronising properties and allowable settling times, approximately τ_2, and hence all aperture widths, Δt, are identical. In practice, variation may occur between synchronising elements and this could be conceivably accounted for in the analysis if the required parameters were known *a priori*. This is not usually the case and a worst case estimation may be obtained by using the worst synchroniser parameters in all latches.

The positioning of the aperture with respect to the batching point deserves some comment. From the functional working of the batched arbiter, the batching point is the exact idealised point where all batches sample their requests. If a request is asserted just prior to the batching point, it is batched, but just after the batching point the request is not registered until after servicing of the batched requests. From the aperture model as described in relation to the redundant synchroniser in Section 3.7.5, an idealised synchronising element would have an aperture "point" (i.e. zero aperture width) where the batching point would occur.

To model the more realistic metastable susceptible batch, the aperture interval should contain the batching point. Since aperture widths are very small in practice compared with circuit delays (usually ~ picoseconds for ~ nanosecond gate delays), the exact position of the batching point within the aperture is inconsequential as will be seen below. For convenience in the analysis, the aperture is placed just after the batching point as depicted in Figure 5.9.

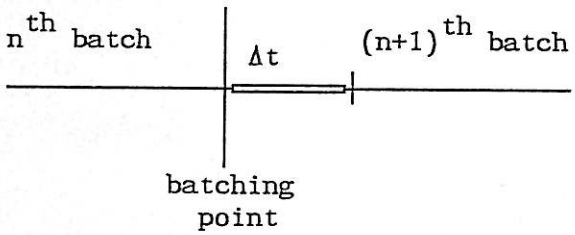


FIGURE 5.9 Assumed Position of Aperture in the Analysis.

One may argue that the batching point should be in the centre of the aperture, depicted in Figure 5.10, as suggested by the argument that half the metastable states resolve to each stable valid logic state "eventually" and the idealised batch is the limit as $\tau \to 0$ of the non ideal latch.

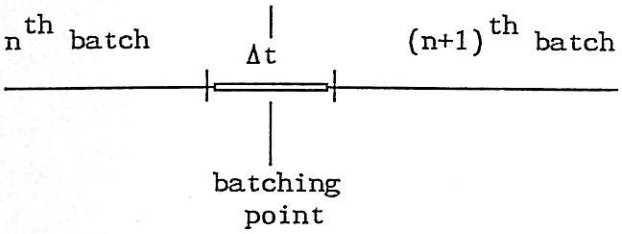


FIGURE 5.10 Central Aperture Position.

The following is a comparison of the two models: Suppose requester h is last serviced a time t_h before the batching point. The probability of requester h lodging a request within the aperture of the first model, Figure 5.9, is denoted p_1 and the corresponding probability for the central aperture position model model, Figure 5.10, is denoted p_2.

$$p_1 = e^{-\lambda_h t_h}(1 - e^{-\lambda_h \Delta t}) \qquad (5.47a)$$

$$p_2 = e^{-\lambda_h (t_h - \frac{\Delta t}{2})}(1 - e^{-\lambda_h \Delta t}) \qquad (5.47b)$$

(Equation (5.47b) assumes $t_h \geq \frac{\Delta t}{2}$ which can be guaranteed by ensuring $D_3 \geq \frac{\Delta t}{2}$ with very little error due to the smallness of Δt).

Since the mean time between requests, $\frac{1}{\lambda_h}$, is much larger than half the aperture width in practice then

$$\frac{p_2}{p_1} = e^{\frac{\lambda_h \Delta t}{2}} \cong 1 \qquad (5.48)$$

That is the probability of metastable failure under both models is almost identical. (The ratio of probabilities is good comparison only for $p_1, p_2 \ll 1$ which is the case here).

The metastable failure rate is now derived, based on the observation that aperture events of significance occur only at the start of non zero batches and in modules with priority higher than all pending requests. Firstly, apert(i) is defined, where i is the state of the batch following a batching point.

$$\text{apert}(i) \triangleq \begin{cases} \sum_{\substack{h \notin i \\ h < hp(i)}} \lambda_h & , i \neq 0 \\ 0 & , i = 0 \end{cases} \tag{5.49}$$

where hp(i) is the highest priority requester in state i. The value apert(i) represents the sum of request rates of those requesters which can cause failure (with metastable duration $> \tau_2$) at the beginning of state i. The probability of no failure at the beginning of state i is given by

$$p_{nf}(i) = e^{-\text{apert}(i)\Delta t} \tag{5.50}$$

where

$$\Delta t = T_0 \, e^{-\frac{\tau_2}{\tau}} \tag{5.51}$$

Using the same notation as in Section 5.6.1, the number of batches with state i per unit time, R_i^n is given by

$$R_i^n = \frac{n_i}{\sum_{j=0}^{m-1} n_j t_j} \tag{5.52}$$

Taking the limit as the number of samples increases

$$R_i \triangleq \lim_{n\to\infty} R_i^n = \frac{\lim_{n\to\infty} \frac{n_i}{n}}{\sum_{j=0}^{m-1} t_j \lim_{n\to\infty} \frac{n_j}{n}} \tag{5.53}$$

From time 0 to t the probability of no failures, p_{nf}, assuming independence of failures, is given by

$$p_{nf} = \prod_{i=1}^{m-1} \left[p_{nf}\ (i)\right]^{R_i t}$$

$$= e^{-\left[\sum_{i=1}^{m-1} \text{apert}(i)\Delta t\ R_i\right] t} \tag{5.54}$$

The independence assumption is equivalent to assuming that when a latch in a module is disabled the metastable generation mechanism is memoryless, that is, independent of the previous latching history. From experimental evidence it is found that this may not be precisely accurate [R.2].

Equations (5.54) and (5.53) suggest defining a normalised metastable failure rate performance parameter for the circuit, NMFR, given by

$$NMFR \triangleq \frac{\sum_{i=1}^{m-1} apert(i)\, p_i}{\sum_{j=0}^{m-1} t_j p_j} \qquad (5.55)$$

The equivalent Poisson failure rate of the arbiter in steady state can be thought of as $\Delta t.NMFR$. The NMFR is normalised with respect to the aperture width Δt to give the failure rate a measure of independence of circuit parameters and to represent the intrinsic synchronisation performance of the arbiter circuit.

The occurrence of aperture events is linked to the arbitration discipline as well as the implementation of the discipline. One can identify possible aperture events by locating points where request timing has an immediate impact on the allocation of the resource. In the above example it was seen that high priority requests occurring at batching points can marginally trigger latches within the circuit that have direct bearing on the decision as to which request to service next. Aperture event locations are identified in other disciplines in Chapter 8, and shown also to be dependent on the circuit implementation in Chapter 7 where clocked batched arbiters are discussed.

5.6.4 Light Request Loading Limits of Performance Parameters

Firstly, the steady state probability vector in the limit as the request rates approach zero is determined. The limit is taken with the ratios of request rates constant as defined in (5.20).

$$\lim_{\lambda \to 0} p = \lim_{\lambda \to 0} \lim_{n \to \infty} p(n) = \lim_{\lambda \to 0} \lim_{N \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$$

$$= \lim_{N\to\infty} \lim_{\lambda\to 0} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$$

$$= \left[\frac{1}{2}, \frac{r_1}{2}, \frac{r_2}{2}, 0, \frac{r_3}{2}, 0, \ldots, \frac{r_k}{2}, 0, 0, \ldots, 0\right]^T \qquad (5.56)$$

The steps in (5.56) are justified in Appendix F, by showing $\lim_{n\to\infty} p(n) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$, and $\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$ converges uniformly as $\lambda\to 0$. Equation (5.56) states that only the zero batch and singleton batches can occur in the light loading limit. The probability of a singleton batch is half the request ratio of its requester, with the zero batch probability being $\frac{1}{2}$. Physically, the arbiter alternates between idling and servicing one requester. Note, however that the mean duration of a zero batch approaches infinity as the request rates approach zero, and the proportion of time spent in batch j, U_j, is given by

$$\lim_{\lambda\to 0} U_j = \begin{cases} 0 & , \quad j \neq 0 \\ 1 & , \quad j = 0 \end{cases} \qquad (5.57)$$

and the proportion time given to requester h, PROP(h) is given by

$$\lim_{\lambda\to 0} \text{PROP}(h) = 0 \qquad (5.58)$$

It follows from Appendix E and (5.56) that the limiting mean waiting time for requester h, is, as expected

$$\lim_{\lambda\to 0} \text{MWT}(h) = D_3 + D_2 \quad , \quad h=1, \ldots, k \qquad (5.59)$$

Also, the limiting normalised metastable failure rate, NMFR, from

(5.55) and (5.56) is

$$\lim_{\lambda \to 0} \text{NMFR} = 0 \tag{5.60}$$

5.6.5 Heavy Request Loading Limit of Performance Parameters

Two cases arise: (1) $D_1 + D_3 > 0$ (2) $D_1 + D_3 = 0$. The limit as request rates approach infinity with fixed ratios as defined in (5.20) is an idealisation of a heavily loaded request situation.

Case 1 $D_1 + D_3 > 0$

The limiting probability vector, $\lim_{\lambda \to \infty} p$, is given by (justification given in Appendix F)

$$\lim_{\lambda \to \infty} p = [0, \quad 0, \quad \ldots \quad ,0, \quad 1]^T \tag{5.61}$$

where the limit is taken as in Section 5.5.3. From (5.61) and (5.43):

$$\lim_{\lambda \to \infty} U_j = \begin{cases} 1 & , \quad j = \text{full batch} \\ 0 & , \quad \text{otherwise} \end{cases} \tag{5.62}$$

and from (5.61) and (5.44):

$$\lim_{\lambda \to \infty} \text{PROP}(\ell) = \frac{\frac{1}{\mu_\ell}}{D_1 + D_2 + D_3 + \sum_{g=1}^{k} \frac{1}{\mu_g}} \tag{5.63}$$

Equation (5.63) is expected if every requester is serviced in turn and receives per batch the time it takes to be serviced.

The limiting MWT(ℓ) is obtained from Appendix E and (5.61)

$$\lim_{\lambda\to\infty} \text{MWT}(\ell) = \lim_{\lambda\to\infty} W(m-1, m-1, \ell)$$

$$= D_1 + D_2 + D_3 + \sum_{\substack{g=1 \\ g\neq\ell}}^{k} \frac{1}{\mu_g} \qquad (5.64)$$

That is, every requester requests straight after being serviced and waits for every other requester to be serviced and the inter batch delays D_1, D_2 and D_3. This is not the maximum mean waiting time for low priority requesters, since under lighter loading conditions these requesters may not request in time for the next batch and hence have to wait for an additional batch of requesters to be serviced before being serviced at the end of the batch after.

The limiting metastable failure rate is given by

$$\lim_{\lambda\to\infty} \text{NMFR} = \frac{\Delta t \text{ apert}(m-1)}{t_{m-1}} = 0 \qquad (5.65)$$

Case 2 $D_1 + D_3 = 0$

$$\lim_{\lambda\to\infty} p = \lim_{\lambda\to\infty} \lim_{n\to\infty} p(n) = \lim_{\lambda\to\infty} \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$$

$$= \lim_{N\to\infty} \lim_{\lambda\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \qquad (5.66)$$

$$= [\, 0,\ 0,\ \ldots,\ \underbrace{\tfrac{1}{2}}_{k-1},\ \underbrace{\tfrac{1}{2}}_{k},\ 0]^T$$

Where $\overline{k-1}$ is the state with all requesters serviced except k-1. Uniform convergence of $\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n)$ as $\lambda \to \infty$ is shown in Appendix F. Equation (5.66) can be interpreted physically by observing that the arbiter alternates between the batch servicing all but the lowest priority requester and the batch servicing all but the second lowest priority requester. This occurs because the last requester service in a batch is unable to request in time to be included in the next batch.

It follows from (5.66), and (5.43) that

$$\lim_{\lambda\to\infty} \mathrm{PROP}(h) = \begin{cases} \dfrac{2}{\mu_h T_1} & , \quad h \neq k \quad \text{and} \quad j \neq k-1 \\ \\ \dfrac{1}{\mu_h T_1} & , \quad h = k \quad \text{or} \quad h = k-1 \end{cases} \tag{5.67}$$

where $T_1 = 2\left\{D_2 + \sum_{g=1}^{k-2} \frac{1}{\mu_g}\right\} + \frac{1}{\mu_{k-1}} + \frac{1}{\mu_k}$

Note the unfairness in (5.67) for the two lowest priority requesters, who receive half the servicing other requesters receive.

It follows from (5.66), and Appendix E that

$$\lim_{\lambda\to\infty} \mathrm{MWT}(h) = \begin{cases} \dfrac{T_1}{2} - \dfrac{1}{\mu_h} & , \quad h \neq k \quad \text{and} \quad h \neq k-1 \\ \\ T_1 - \dfrac{1}{\mu_h} & , \quad h = k \quad \text{or} \quad h = k-1 \end{cases} \tag{5.68}$$

where T_1 is defined in (5.67).

The limiting metastable failure rate is given by

$$\lim_{\lambda \to \infty} \text{NMFR} = 0 \qquad , k > 2 \tag{5.69}$$

Note that the metastable failure rate is zero at both loading extremes which suggests a maximum exists at moderate request loading.

5.7 COMPUTER STUDY AND NUMERICAL RESULTS

Much of the interesting and relevant behaviour of batching arbiters occurs between the limiting extremes of light and heavy request loading. This behaviour may best be examined using numerical computer techniques since the theoretical expressions are not easily interpreted for intermediate request rates.

5.7.1 Assumptions and Parameter Selection

The following simplyfying assumptions are made in the computer study:

$$\mu_h = \mu \quad , \quad h = 1, 2, \ldots, k \tag{5.70}$$

$$\lambda_h = \lambda \quad , \quad h = 1, 2, \ldots, k \tag{5.71}$$

All requesters are assumed to have identical request and service statistical characteristics, with two cases considered for service times: constant and exponential distributions.

The inter-batch time durations D_1, D_2 and D_3 are selected to correspond approximately to an arbiter design such as the decentralised batching arbiter presented in Section 4.3.1. A typical TTL design for this arbiter might be : common line delay = 25 nsec; τ_1 = 20 nsec; τ_2 = 50 nsec; τ_3 = 50 nsec and τ_4 = 50 nsec. From the timing diagram in Figure 4.3 and typical TTL delay times, approximate values for D_1, D_2 and D_3 are

$$D_1 = 110 \text{ nsec} \qquad D_2 = 110 \text{ nsec} \qquad D_3 = 90 \text{ nsec} \tag{5.72}$$

Within the framework of the batching arbiter model the relative values of D_1, D_2 and D_3 to the mean service time $\frac{1}{\mu}$ are all that is needed and not absolute values. Consequently, the *unit of time* adopted for all results will be the mean service time $\frac{1}{\mu}$.

Typical service times vary with applications. For example, an arbiter for resolving multiple interrupts may result in typical service times not shorter than 150 μsec. The values of D_1, D_2 and D_3 could then be approximated as zero in this application. Another example is that of a bus arbiter. Typically, a bus transaction may be as short as 500 nsec, giving $D_1 = D_2 = D_3 \cong 0.2$. To show the effect of large inter-batch times in relation to mean service times, the case of $D_1 = D_2 = D_3 = 1$ is also considered in the numerical results.

The range over which the mean request rate varies is chosen to cover most conceivable applications. In a 5 requester arbiter with $D_1 = D_2 = D_3 = 0$, for example, each requester can be coordinated to request in an orderly fashion (deterministically) to tie up the resource continuously with 0.25 requests per mean service time. In the non deterministic situation of the arbiter model (random requests), request rates above 0.25 tend to saturate the arbiter. It would be expected that a system desiger would not allow request saturation to be too heavy due to the resulting inefficiencies incurred due to requesters waiting. A range from 0 to 1.5 requests for mean service time is considered adequate in the numerical results.

5.7.2 Proportion of Time Allocated to Each Requester

Graphs 5.1, 5.2, 5.3, 5.4 and 5.5 show the proportion of total time allocated to servicing each requester, PROP(h), of a 5 requester arbiter. Higher priority requesters receive more time than lower priority requesters in all graphs. However, for request rates less than 0.25 requests/service time, all requesters receive approximately equal proportions of time. Hence, the batched arbiter is fair, as far as time allocation is concerned, when request rates are less than 0.25 requests/service time.

A comparison is made between results from exponentially distributed and constant service times in Graphs 5.1 and 5.2. It can be seen that there is little difference in the results for the two service time distributions. The exponentially distributed service time results tend to saturate at slightly higher request rates. This can be generally observed for all performance measures even with different values of D_1, D_2 and D_3, and can be explained rather crudely by noting that greater variability introduced into the request service cycle results in a more random service ordering. This randomness tends to favour lower priority requesters and requires higher mean request rates to produce a given level of saturation in the arbiter. One could take the extreme case of deterministic re-requests times and service times as an example with an ideal batched arbiter $(D_1 = D_2 = D_3 = 0)$. It is clear that for $\lambda > 0.25$ the arbiter is completely saturated with

$$PROP(h) = \begin{cases} 0.25 & , \quad 1 \leq h \leq 3 \\ 0.125 & , \quad h = 4 \text{ or } 5 \end{cases} \qquad (5\ 73)$$

As randomness is introduced into request or service behaviour, the request rate necessary to produce a given level of saturation increases.

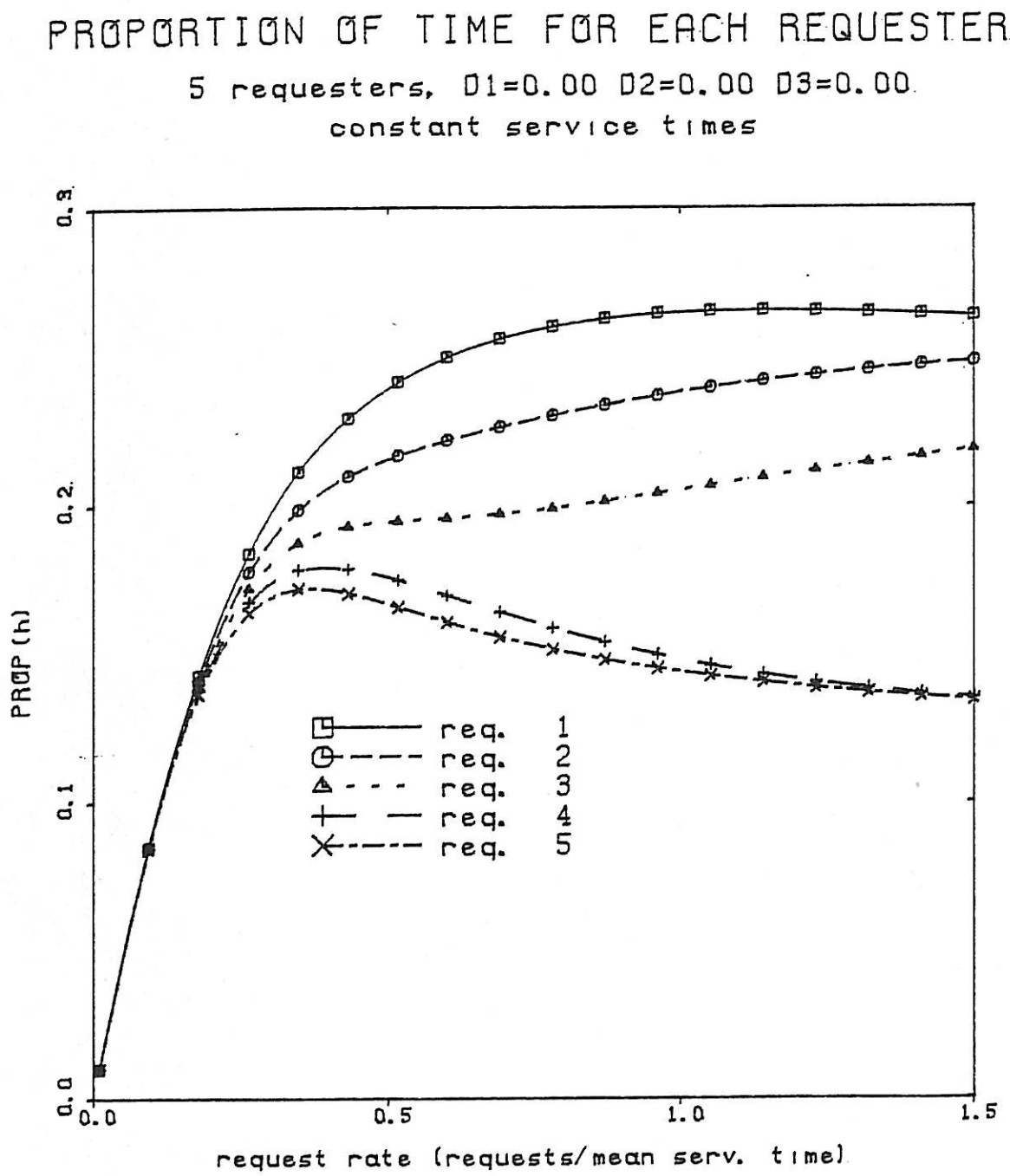
In Graphs 5.1 and 5.2 where $D_1 = D_2 = D_3 = 0$, the two lowest priority requesters 4 and 5, receive approximately half the servicing that the others receive for heavy requests loadings, as predicted in the heavily loaded limits discussed in Section 5.6. Note that this effect becomes less pronounced as $D_1 + D_3$ increases as shown in Graphs 5.3 and 5.5. In fact in Graph 5.5, for large values of $D_1 + D_3$ and request rates the arbiter distributes the resource equally amongst the requesters, due to sufficient time being available at the end of batches for re-requesting.

In all cases, the batched arbiter only displays significant time allocation unfairness for request rates above 0.25 requests/service time which corresponds to full utilisation when deterministic request rates and service times are applied.

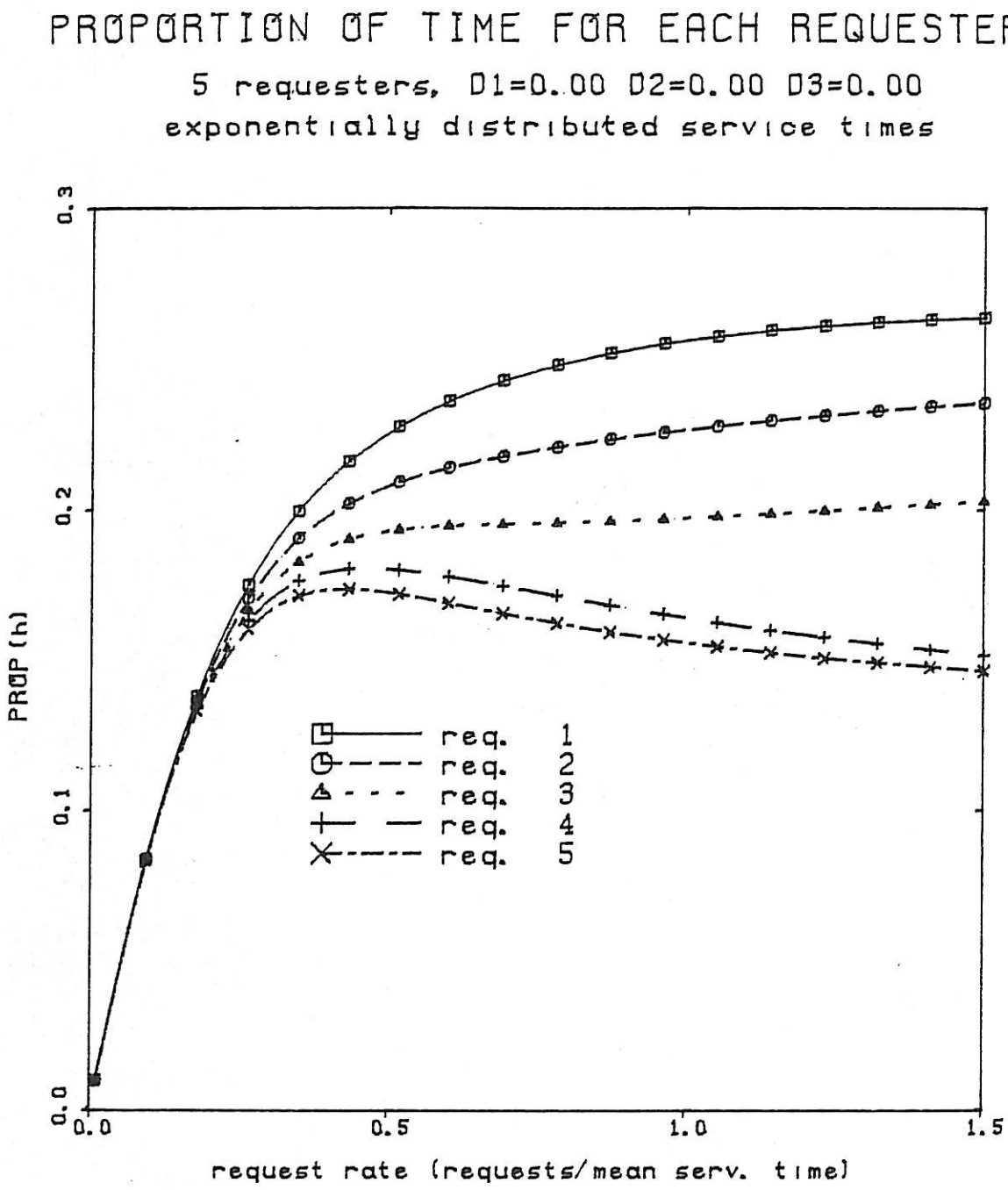
Graphs 5.3 and 5.4 show a comparison of results from the Markov analysis and an independent Monte-Carlo simulation[1] with 20,000 requests per point. The two sets of results agree within statistical variation of the Monte-Carlo simulation, and provide a degree of confidence in both the analysis techniques. This technique of validating analytical methods has been employed throughout this thesis and limited examples only are presented.

---

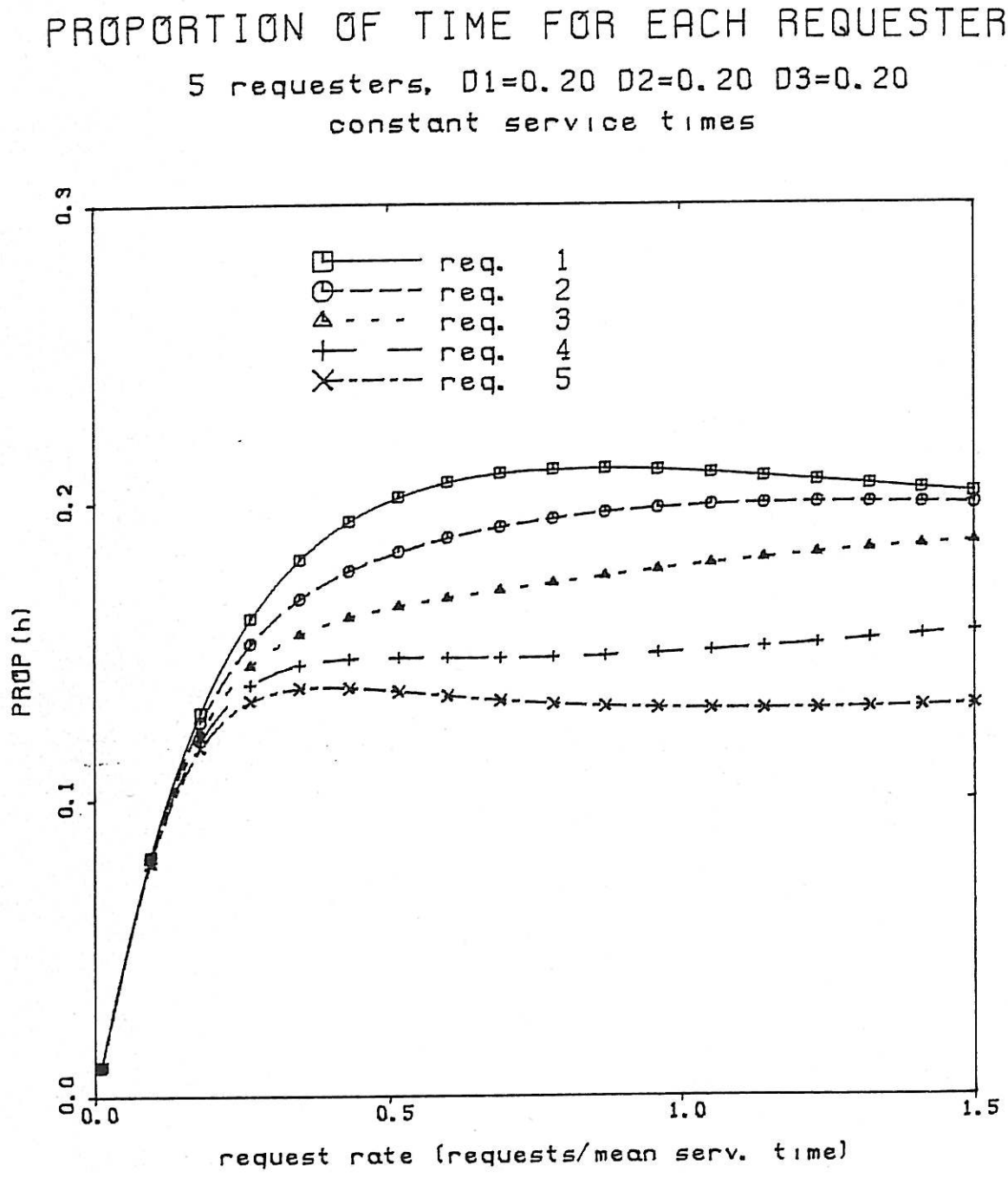
[1] A brief description of the Monte-Carlo simulation is given in Chapter 8.


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00 D3=0.00
constant service times
PROP (h)
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

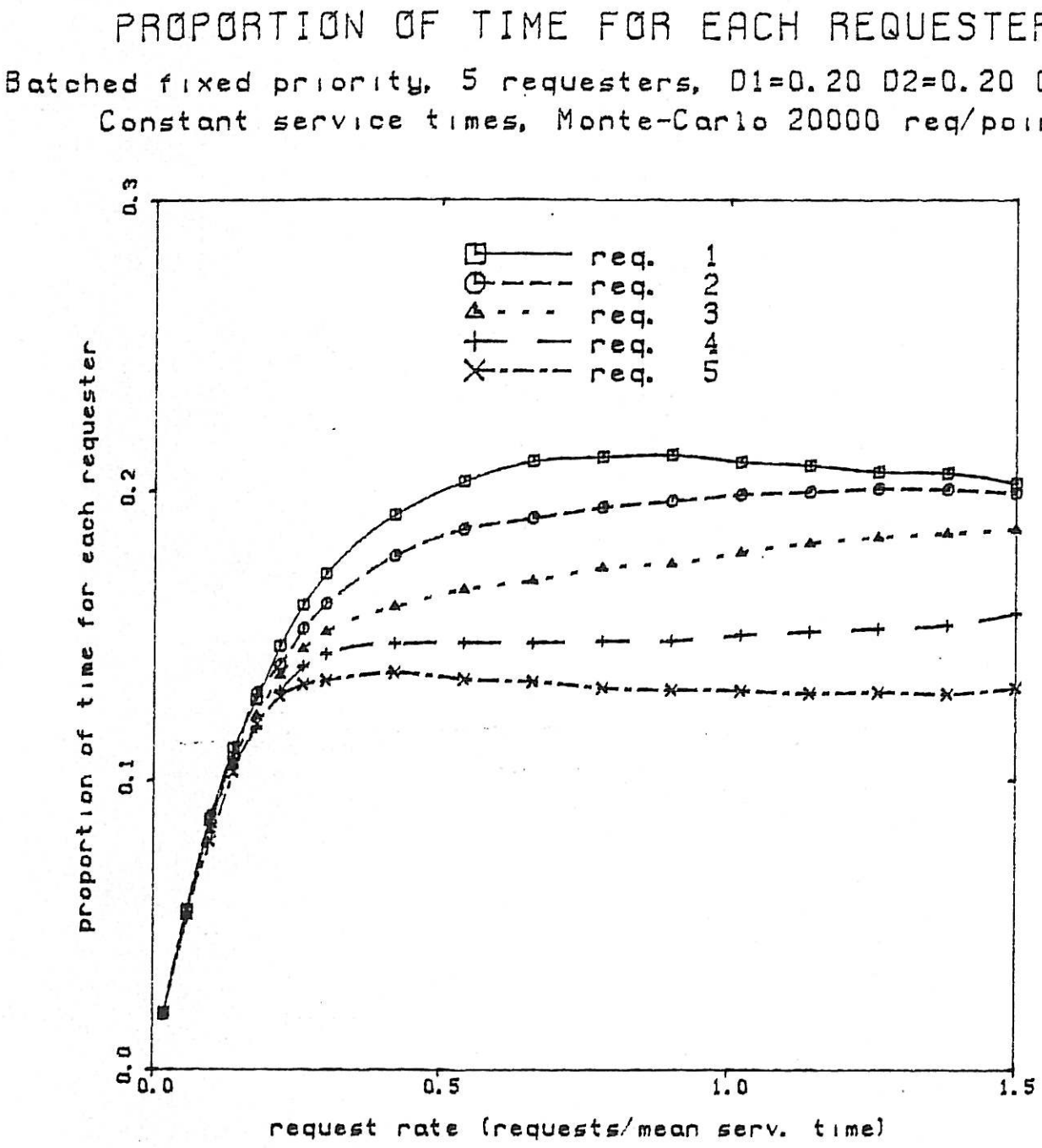
GRAPH 5.1


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00 D3=0.00
exponentially distributed service times
PROP (h)
0.0
0.1
0.2
0.3
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

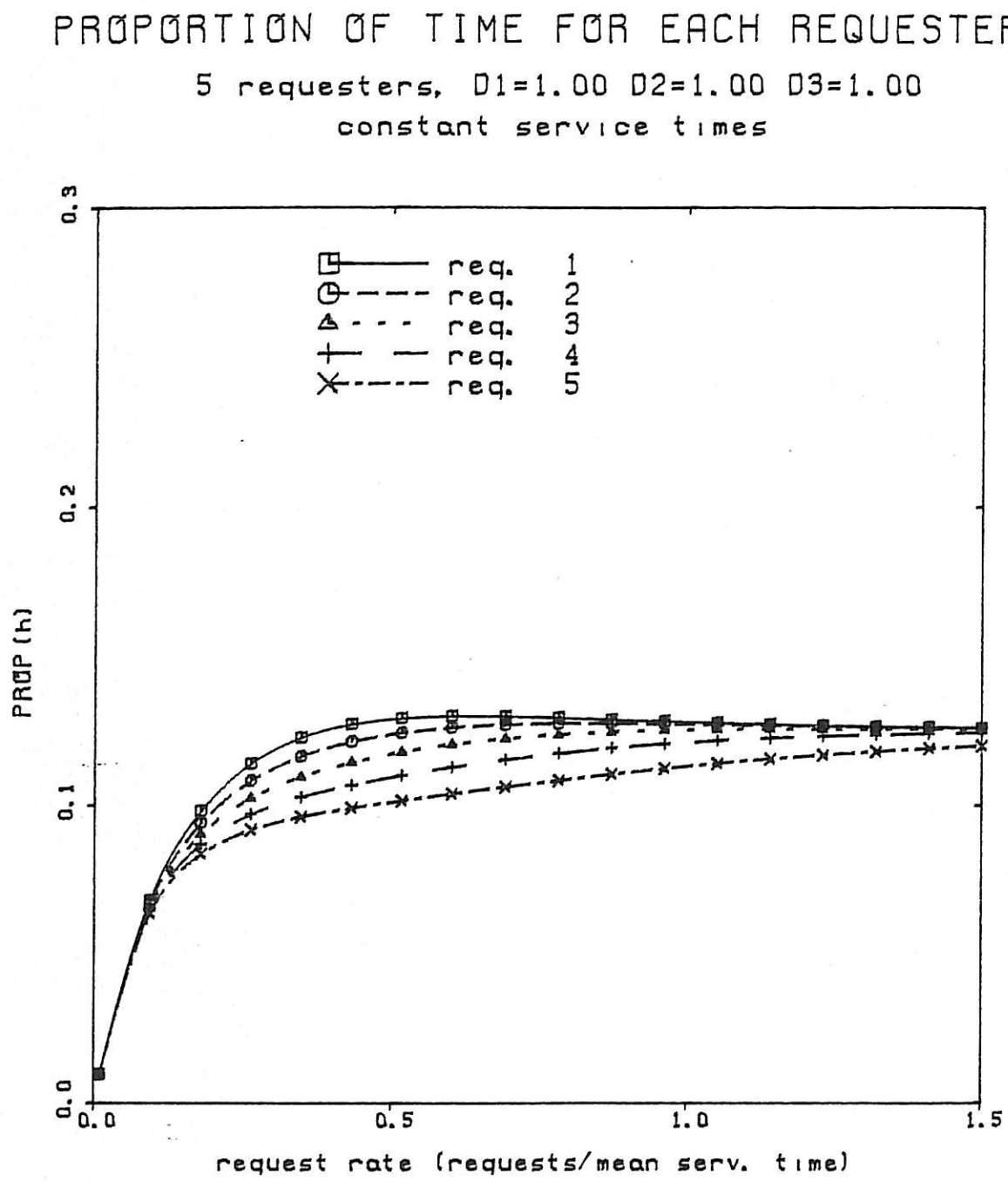
GRAPH 5.2


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20 D3=0.20
constant service times
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.3


PROPORTION OF TIME FOR EACH REQUESTER
Batched fixed priority, 5 requesters, D1=0.20 D2=0.20 D3=0.2
Constant service times, Monte-Carlo 20000 req/point
req. 1
req. 2
req. 3
req. 4
req. 5
proportion of time for each requester
0.0
0.1
0.2
0.3
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.4


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=1.00 D2=1.00 D3=1.00
constant service times
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.5

5.7.3 Mean Waiting Time for Each Requester

Graphs 5.6, 5.7, 5.8, 5.9 and 5.10 show the mean waiting time, MWT(h), plotted against mean request rate, for each requester in a 5 requester arbiter. In all plots a higher priority requester has a shorter mean waiting time than a lower priority requester.

The relative differences between requesters for mean waiting times is more pronounced than results for proportion of time seen in the previous section. For example, a range of over 100% occurs at 0.25 request/service time. If fairness is measured in terms of waiting time, the batched arbiter may not be considered fair for all request rates less than 0.25 request/service time. Notice the difference in fairness conclusions based on proportion of time allocated and those based on mean waiting time. The difference can be explained by observing that large differences in waiting times can occur when requests occur simultaneously even though equal proportions of time are allocated to the requesters by the arbiter. The mean waiting time performance parameter is more sensitive to contention amongst requesters than the proportion of time.

In Graph 5.6 where $D_1 = D_2 = D_3 = 0$, the two lowest priority requesters are again seen to be disproportionately disadvantaged under heavy request loading conditions, in line with previous remarks in Section 5.6.5. Except for large inter-batch times as in Graph 5.10, it can be concluded that the batching arbiter displays considerable bias towards high priority requesters in terms of waiting time in moderate to heavy request loading condition. In Graph 5.10, the larger inter-batch times allow all requesters time to re-request before the next batching point under heavy request loading.

Numerical results in Graphs 5.7 and 5.9 compare constant and exponential service time distributions. The effect of exponential distribution can be seen to be a slight lowering of contention compared with constant service times. The Markov analysis is validated against Monte-Carlo simulation in Graphs 5.7 and 5.8, where excellent agreement can be seen again.

5.7.4 Metastable Failure Rate

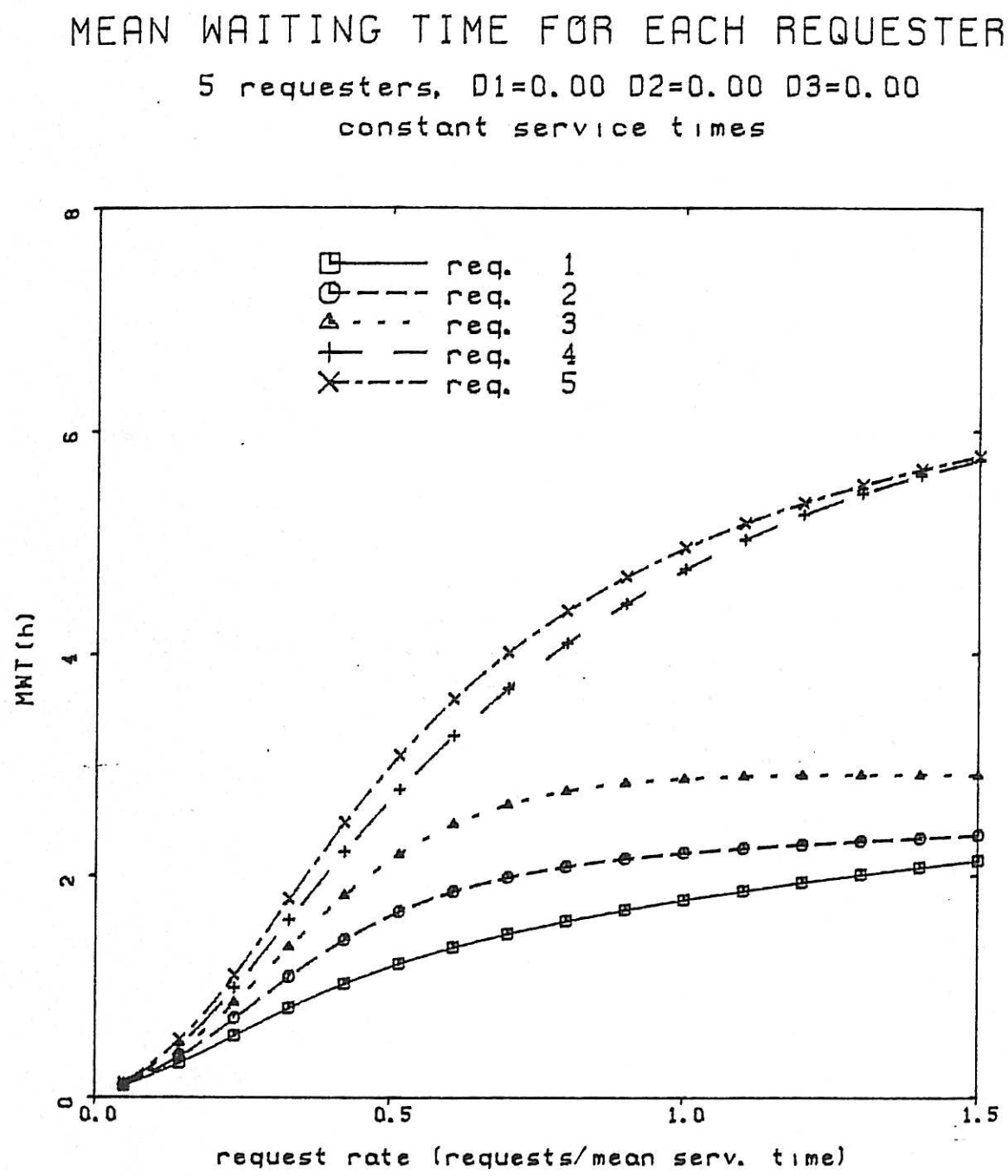
The graphs in Graphs 5.11, 5.12, 5.13 and 5.14 show the normalised metastable failure rate, NMFR, for batched arbiters with 2 to 5 requesters. Each arbiter has a peak failure rate corresponding to a request rate near the onset of saturation ($\cong 1/(k-1)$ request/service time for a k requester arbiter). The peak value occurs when the arbiter is resolving maximum contention between requests. At very heavy request rates, few requests occur at the beginning of a batch where aperture events occur, since batches are long and requesters request soon after releasing the resource. Requests are serviced in an orderly round robin fashion and little arbitration synchronisation occurs.

The effect of increasing D_1, D_2 and D_3 is to reduce the metastable failure rate, as can be seen in Graphs 5.12, 5.13 and 5.14. (However, this is not recommended as a technique for improving reliability - this can be better achieved by increasing the metastable settling time τ_2). Increasing D_2 has the effect of saturating the arbiter for smaller request rates, hence aperture masking effects occur for smaller request rates reducing the failure rate. Increasing $D_1 + D_3$ allows requesters more opportunity to request before aperture events at the beginning of the next batch, increasing the probability of aperture masking by high

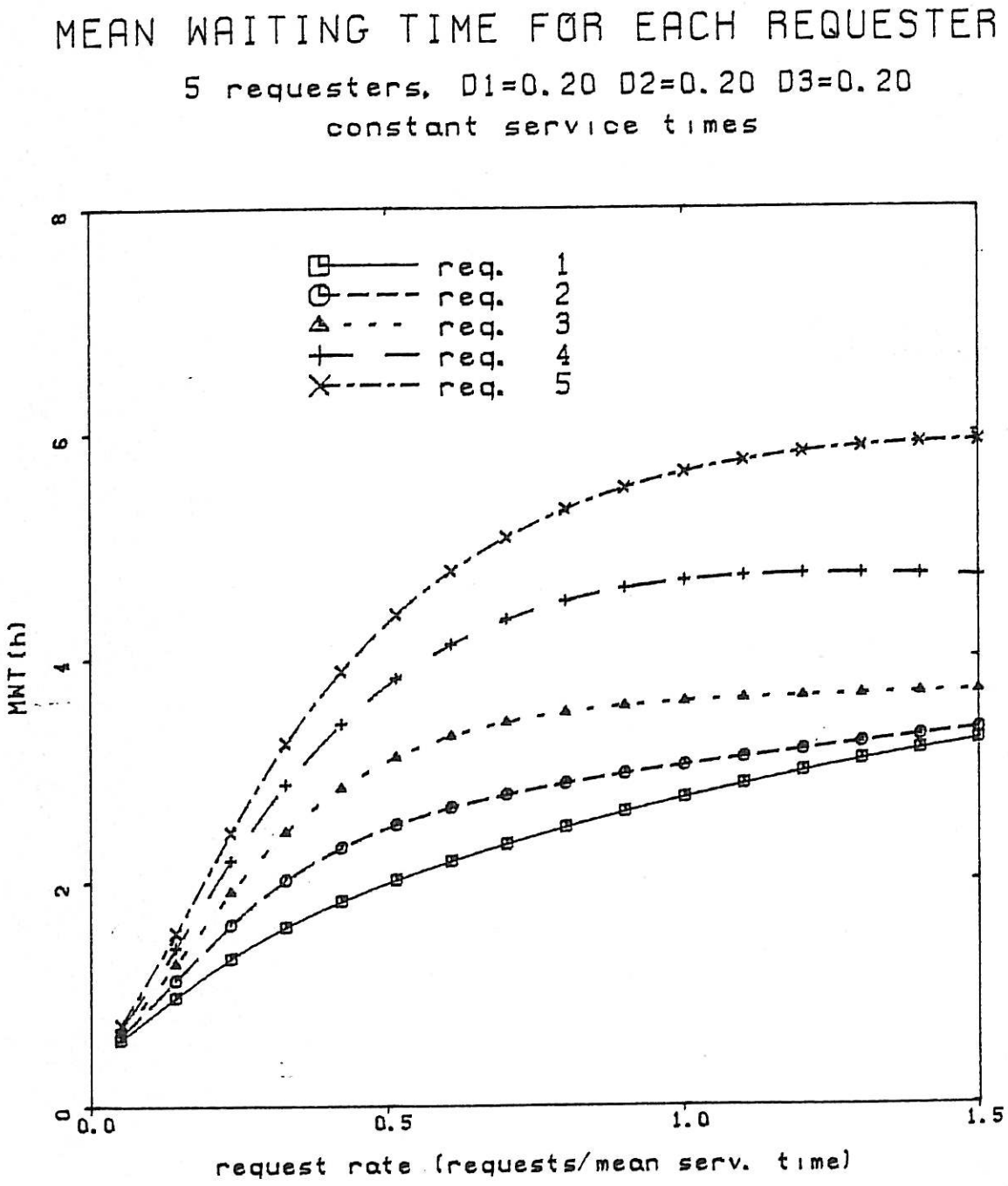
priority latched requests and also reducing the number of unlatched requests which could possibly have given rise to an aperture event.

Notice that the failure rate of a two requester arbiter with $D_1 + D_3 = 0$ is unbounded with increasing request rates (see Graph 5.11). This is because every non zero batch is singleton and so an aperture event occurs on average at the beginning of every second batch. The metastable failure rate is then proportional to the request rate when saturated.

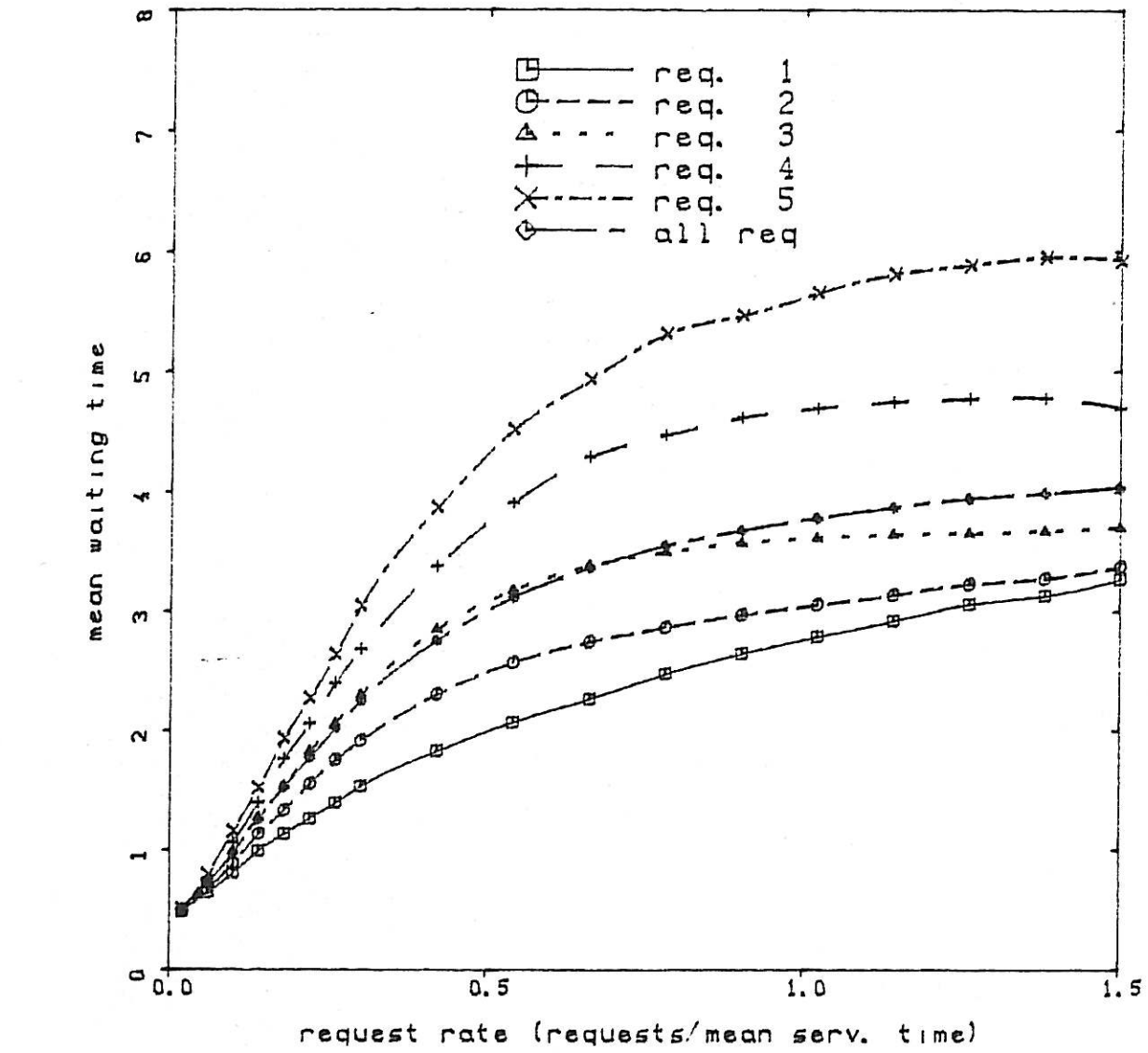
Exponential service times are examined in Graph 5.14 which is to be compared to Graph 5.13. Again a slight drop in contention is observed for a given request rate.


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00 D3=0.00
constant service times
req. 1
req. 2
req. 3
req. 4
req. 5
MWT (h)
0
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

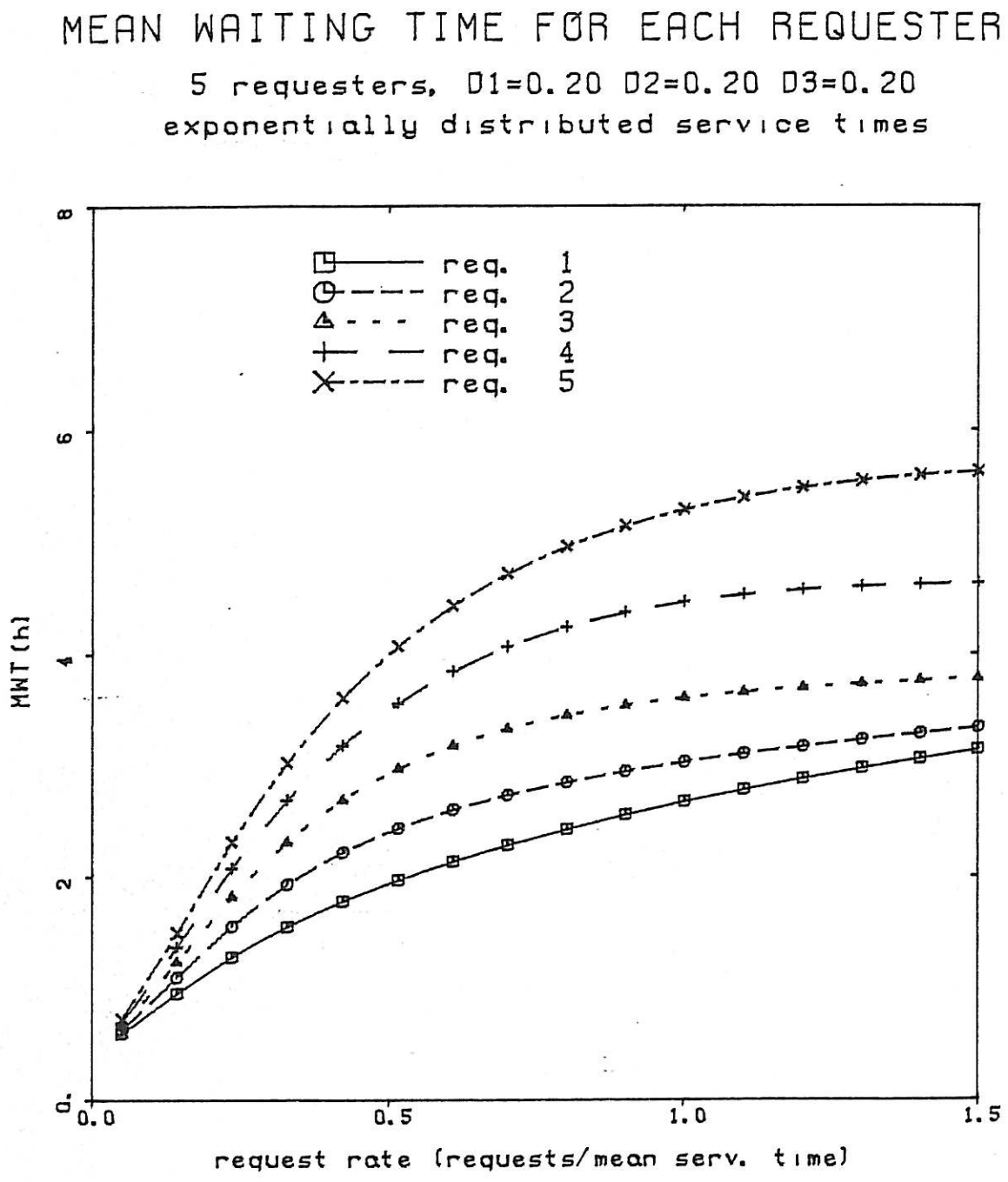
GRAPH 5.6


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20 D3=0.20
constant service times
req. 1
req. 2
req. 3
req. 4
req. 5
MWT(h)
0
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

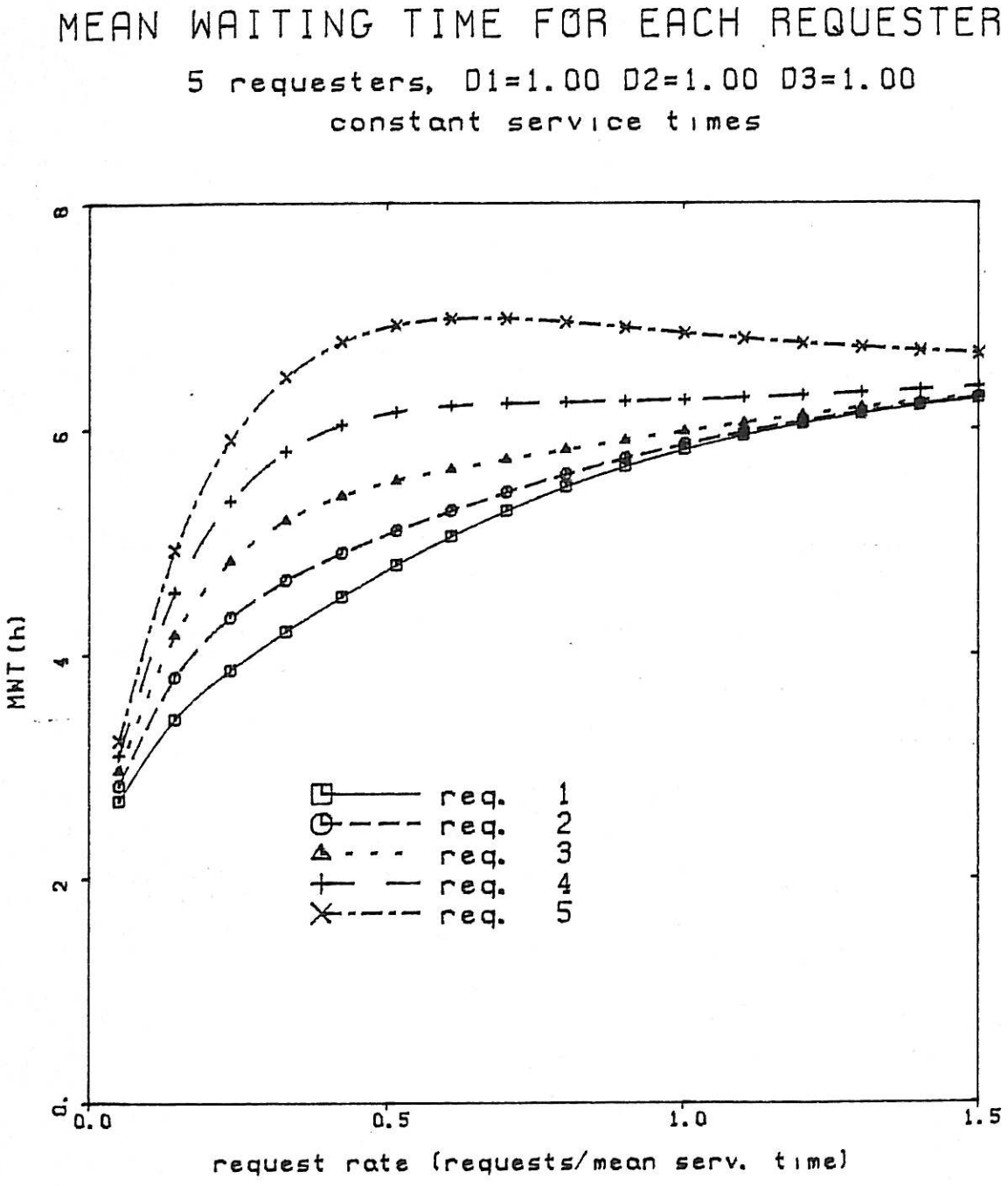
GRAPH 5.7


MEAN WAITING TIME FOR EACH REQUESTER
Batched fixed priority, 5 requesters, D1=0.20 D2=0.20 D3=0.2
Constant service times, Monte-Carlo 20000 req/point
req. 1
req. 2
req. 3
req. 4
req. 5
all req
mean waiting time
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

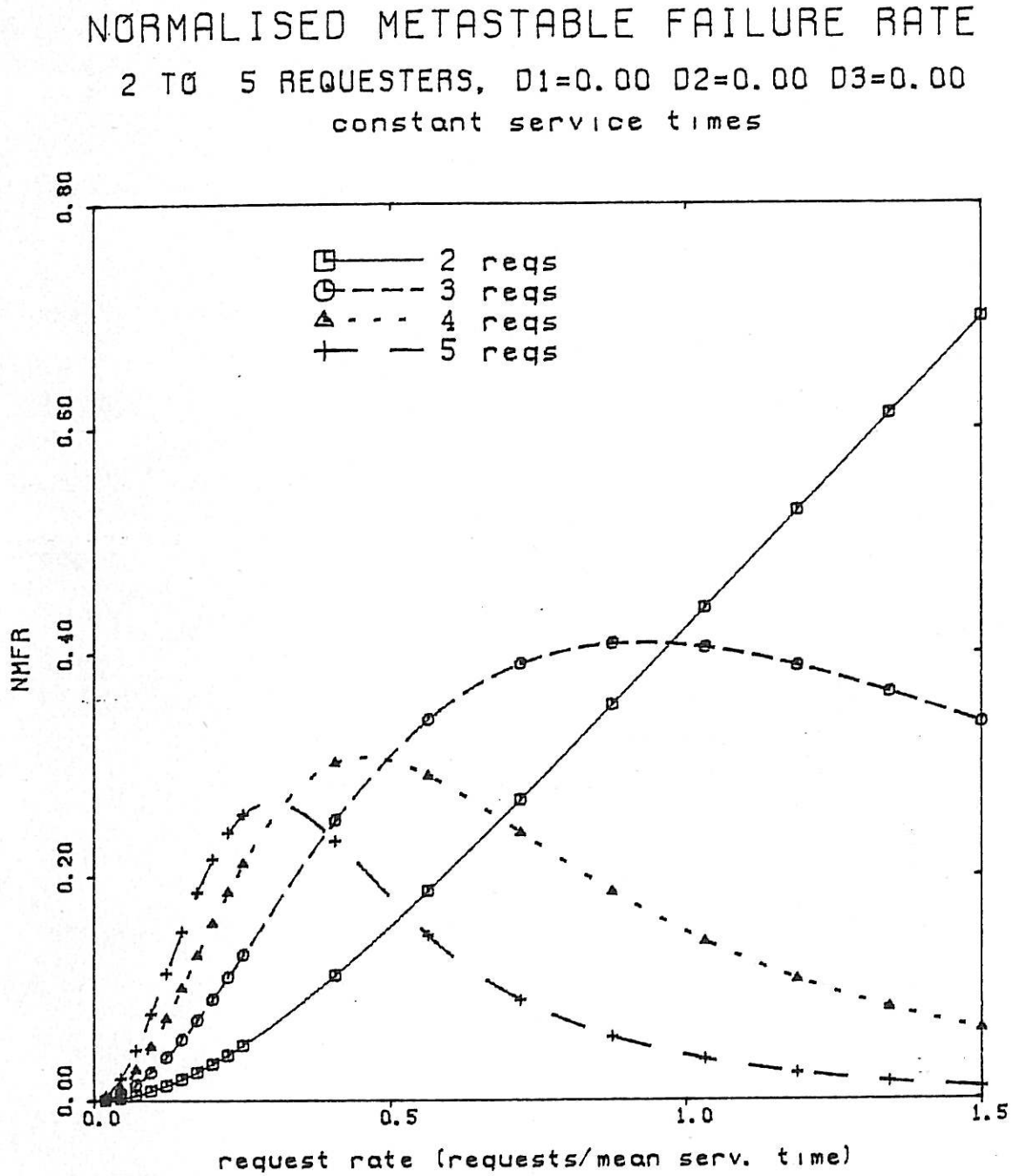
GRAPH 5.8


MEAN WAITING TIME FØR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20 D3=0.20
exponentially distributed service times
req. 1
req. 2
req. 3
req. 4
req. 5
MWT (h)
0.
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

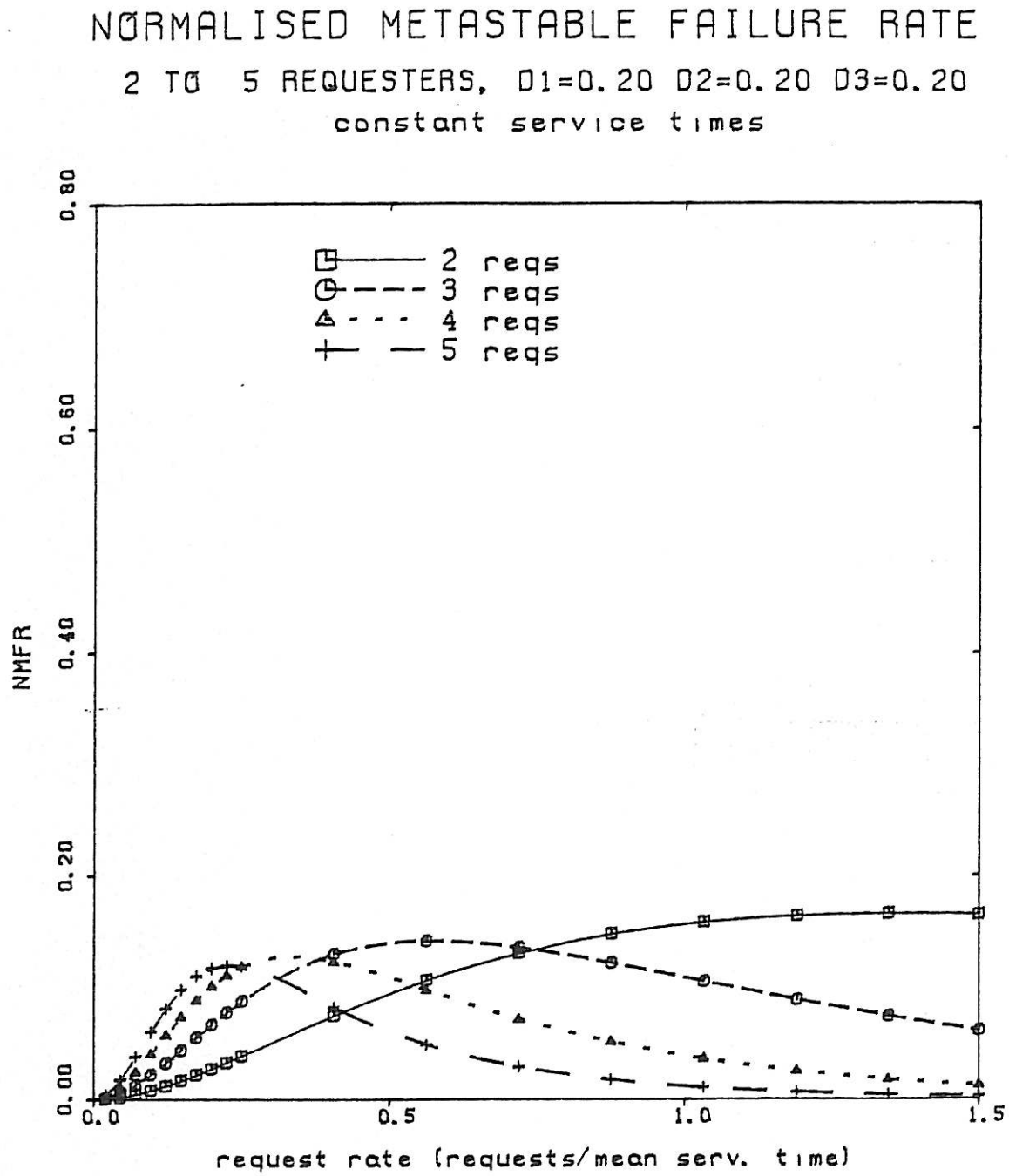
GRAPH 5.9


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=1.00 D2=1.00 D3=1.00
constant service times
MWT (h)
0.
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

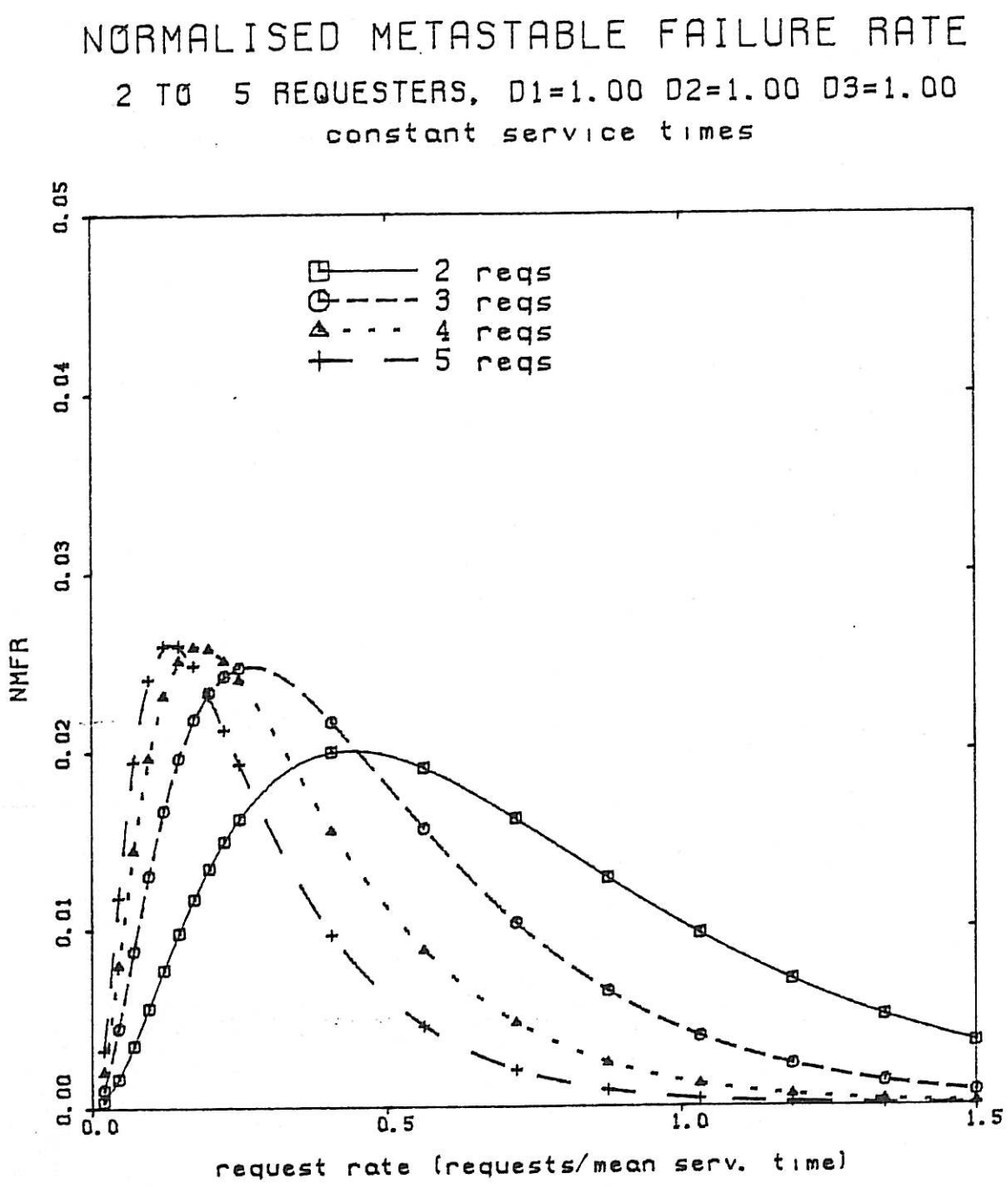
GRAPH 5.10


NORMALISED METASTABLE FAILURE RATE
2 TO 5 REQUESTERS, D1=0.00 D2=0.00 D3=0.00
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.20
0.40
0.60
0.80
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

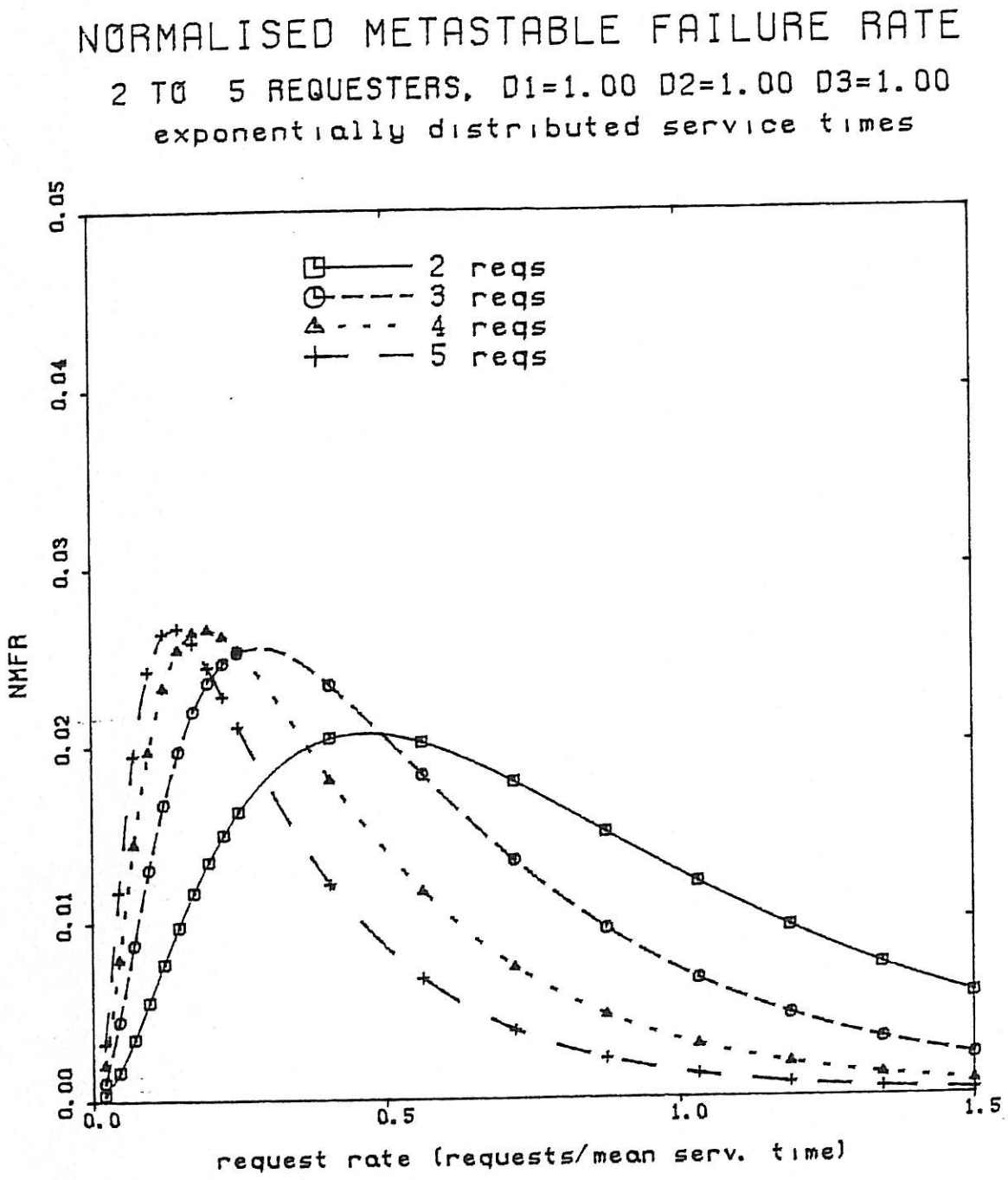
GRAPH 5.11


NORMALISED METASTABLE FAILURE RATE
2 TO 5 REQUESTERS, D1=0.20 D2=0.20 D3=0.20
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.20
0.40
0.60
0.80
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.12


NORMALISED METASTABLE FAILURE RATE
2 TO 5 REQUESTERS, D1=1.00 D2=1.00 D3=1.00
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.01
0.02
0.03
0.04
0.05
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.13


NORMALISED METASTABLE FAILURE RATE
2 TO 5 REQUESTERS, D1=1.00 D2=1.00 D3=1.00
exponentially distributed service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.01
0.02
0.03
0.04
0.05
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 5.14

5.8 CONCLUSION

This chapter has presented theoretical and numerical results on the performance of fixed priority batched arbiters. The modelling and analysis presented has been shown to give results of practical interest for computer system designers. Particular attention has been given to studying the failure rate of the arbiter due to metastable behaviour. Factors which affect this failure rate have been identified, such as settling time, module masking and request rates. A maximum failure rate has been shown to exist, corresponding to maximum request contention and zero inter-batch times.

In the theoretical and numerical utilisation results, an important feature of fixed priority batched arbiters has been highlighted : the two lowest priority requesters receive approximately half the servicing compared with other requests, under heavy loading and small inter-batch times. Another feature which may apply more generally than to batched arbiters, is that the "fairness" of the arbiter depends very much on the criterion for assessment. In terms of proportion of time each requester received, the arbiter is fair up to the saturation levels of requesting. However, in terms of relative mean waiting times for requesters, the arbiter is not fair for even moderate request loadings. Thus, for throughput applications the arbiter allocates the resource fairly but *not* for applications sensitive to access times of the requesters. In all cases batched arbiters have bounded waiting times independent of request loadings - a property fixed priority non batched arbiters lack.

CHAPTER 6

ANALYSIS OF NON BATCHED FIXED PRIORITY ARBITERS

6.1 INTRODUCTION

The previous chapter presented an analysis of batched fixed priority arbiters where state transitions are defined to occur at batching points. *Non* batched arbiters arbitrate before *each* service, and state transitions are defined at these points. Apart from this difference, the analysis of non batched arbiters presented in this chapter is similar to the analysis of batched arbiters, and the presentation follows the structure of the previous chapter in a more condensed fashion. The model and requester excitation assumptions can be found in Chapter 4, Section 4.4.

The analysis presented in this chapter is new and has not, to the author's knowledge, appeared elsewhere. Although the analysis is restricted to the fixed priority discipline, the technique is generalisable to disciplines with a Markov state representation as discussed in Section 4.4, by adding the discipline state as another dimension of the request state.

The organisation of the chapter is as follows. In Section 6.2 the state of the fixed priority arbiter model introduced in Chapter 4 is defined precisely and shown to lead to a discrete time homogeneous Markov chain. The transition probabilities are derived in Section 6.3 and limiting results considered in Section 6.4, where the Markov chain is shown to be irreducible and primitive and thus, unique steady state probabilities of the states exist, independent of the initial probability vector. From these steady state probabilities, performance measures and their limiting values are derived in Section 6.5. These measures will be seen to apply directly to disciplines other than fixed priority. The chapter ends with computer results and conclusions.

6.2 STATE DEFINITION AND MARKOV PROPERTY

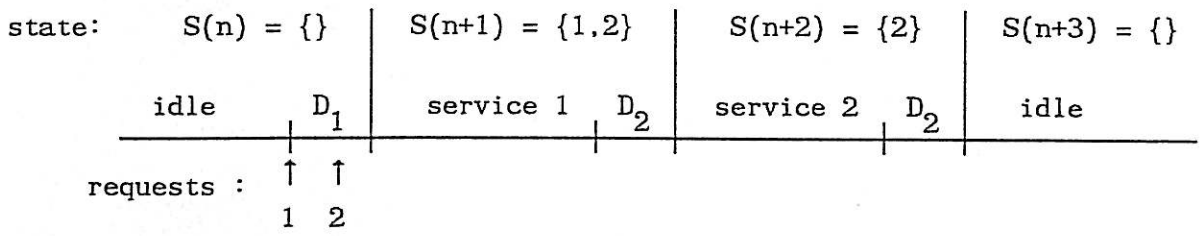


FIGURE 6.1 State Definition Example.

The regeneration points of the imbedded Markov chain occur at the start of each service and idle period (i.e. end of each D_2), an example of which is shown in Figure 6.1. The state of the Markov chain is defined to be the set of pending requests at each regeneration point. The corresponding integer representation for the set of pending requests as defined in Section 5.2 is used interchangeably with the set representation.

The Markov property of the sequence of states follows from the observation that the probability of the next state is dependent only on the set of pending requests at the start of the previous state and those requests occurring during this state (which are dependent only on the state composition due to the memoryless property of the re-request time distributions). The state transition probabilities are time independent and so the Markov chain is homogeneous.

6.3 STATE TRANSITION PROBABILITIES

The derivation of state transition probabilities is similar to that of Section 5.4 and only brief explanations are given in this section. Firstly, some notation is introduced:

For a non zero state j:

$$hp(j) \triangleq \text{highest priority requester} \in j \tag{6.1}$$

$$Q(j, h) \triangleq \text{prob(requester } h \text{ has no request pending at the end of the state } j)$$

$$= \begin{cases} e^{-\lambda_h D_2} \int_0^\infty f_g(t) e^{-\lambda_h t} dt & , h \notin j \\ e^{-\lambda_h D_2} & , h = g \\ 0 & , h \in j \setminus \{g\} \end{cases} \tag{6.2}$$

where $g = hp(j)$

and $j \setminus \{g\}$ is state j without requester g.

Note that a request h pending at the start of the state j and not serviced during the state j, is assumed to be pending at the end of the state j giving $Q(j, h) = 0$.

For the constant service time of $\frac{1}{\mu_g}$:

$$\int_0^\infty f_g(t)\, e^{-\lambda_h t}\, dt = e^{-\frac{\lambda_h}{\mu_g}} \tag{6.3}$$

and for exponentially distributed service times with mean $\frac{1}{\mu_g}$:

$$\int_0^\infty f_g(t)\, e^{-\lambda_h t}\, dt = \frac{\mu_g}{\mu_g + \lambda_h} \tag{6.4}$$

The state transition probabilities are given by

$$P_{ij} = \begin{cases} \prod_{h \in i} (1 - Q(j,h)) \prod_{h \notin i} Q(j,h) & , \quad j \neq 0 \\ \dfrac{\sum_{h \in i} \left\{ \lambda_h \prod_{\substack{g \in i \\ g \neq h}} (1 - e^{-\lambda_g D_1}) \right\} \prod_{f \notin i} e^{-\lambda_f D_1}}{\sum_{\ell=1}^{k} \lambda_\ell} & , \quad j = 0 \end{cases} \tag{6.5}$$

where a product of no terms is 1 and a sum of no terms is 0.

6.4 LIMITING PROPERTIES OF THE PROBABILITY TRANSITION MATRIX

This section examines steady state limiting behaviour and light/heavy request loading limits of the probability transition matrix. Again, results presented here parallel the corresponding results for the batched version and detail is kept to a minimum.

6.4.1 Steady State Limiting Results

Theorem 6.1 The probability transition matrix, P, of the fixed priority arbiter model has a unique positive limiting probability vector independent of the initial probability vector provided $\mu_h , \lambda_h > 0$ for h=1, .., k and $D_1 + D_2 > 0$.

The proof of Theorem 6.1 can be found in Appendix G. The case of $D_1 + D_2 = 0$ is treated separately since the full state (all requests

pending) cannot be reached and P is then reducible. A new matrix P_1, is defined when $D_1 + D_2 = 0$, as P without the full state. Of course, P_1 is a probability transition matrix only when $D_1 + D_2 = 0$.

Theorem 6.2 For $D_1 + D_2 = 0$, the probability matrix P_1, has a unique positive limiting probability vector independent of the initial probability vector, provided $\mu_h, \lambda_h > 0$ for $h=1, .., k$.

The proof of Theorem 6.2 is presented in Appendix G.

6.4.2 Light and Heavy Request Loading Limits

As usual, limits are taken with fixed request rate ratios as defined in Section 5.5.2.

$$\lim_{\lambda \to 0} Q(j,h) = \begin{cases} 0 & , \quad h \in j \setminus \{hp(j)\} \\ 1 & , \quad \text{otherwise} \end{cases} \tag{6.6}$$

From (6.5) and (6.6) it follows that

$$\lim_{\lambda \to 0} p_{ij} = \begin{cases} 1 & , \quad |j| \geq 1 \ , \quad i = j \setminus \{hp(j)\} \\ r_g & , \quad j = 0 \ , \quad |i| = 1 \ , \quad g \in i \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.7}$$

where r_g is defined in (5.20).

The state transition is deterministic when the state contains more than one requester. That is the next state consist of the remaining requests after the highest priority requester is serviced.

After at most k transitions a 2-cycle is reached where the arbiter alternates between zero states and singleton states as follows. Let the probability vector of the n^{th} state be $p(n) = [p_i(n)]^T$ then for $n = 2w$, some integer w, $n > k$.

$$p_i(2w) = \begin{cases} \alpha\, r_g & , \quad i = \{g\}, g = 1, \ldots, k \\ 1 - \alpha & , \quad i = 0 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.9}$$

where

$$\alpha = \sum_{|i| \text{odd}} p_i(0) \tag{6.10}$$

Note the dependence of (6.10) on the initial probability vector.

The heavy request loading limits are considered below.

$$\lim_{\lambda \to \infty} Q(j,h) = \begin{cases} 1 & D_2 = 0 \quad , \quad h = hp(j) \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

Four cases are now considered for D_1, D_2: (i) D_1 , $D_2 > 0$, (ii) $D_1 > 0$, $D_2 = 0$, (iii) $D_1 = 0$, $D_2 > 0$, (iv) $D_1 = D_2 = 0$. Equations (6.12), (6.13), (6.14) and (6.15) are obtained from (6.11) and (6.5).

(i) D_1, $D_2 > 0$

$$\lim_{\lambda \to \infty} p_{ij} = \begin{cases} 1 & , \quad i = \text{full state} \\ 0 & , \quad i \neq \text{full state} \end{cases} \tag{6.12}$$

That is, the full state is reached after one transition and only requester 1 is always serviced at the exclusion of all other requesters.

(ii) $D_1 > 0,\ D_2 = 0$

$$\lim_{\lambda \to \infty} p_{ij} = \begin{cases} 1 & , \quad i = \text{full state} & , \quad j = 0 \\ 1 & , \quad i = \text{full state} \setminus \{hp(j)\} & , \quad j \neq 0 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.13}$$

After one transition the highest two priority requesters alternatively receive service at the exclusion of all others.

(iii) $D_1 = 0,\ D_2 > 0$

$$\lim_{\lambda \to \infty} p_{ij} = \begin{cases} r_g & , \quad i = \{g\} & , \quad j = 0 \\ 1 & , \quad i = \text{full state} & , \quad j \neq 0 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.14}$$

After at most two transitions requester 1 is serviced in every state to the exclusion of all other requesters.

(iv) $D_1 = D_2 = 0$

$$\lim_{\lambda \to \infty} p_{ij} = \begin{cases} r_g & , \quad i = \{g\} & , \quad j = 0 \\ 1 & , \quad i = \text{full state} \setminus \{hp(j)\} & , \quad j \neq 0 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.15}$$

After at most two transitions, the highest two priority requesters are alternatively serviced with all others excluded.

6.5 PERFORMANCE PARAMETERS

The definitions of the performance parameters can be motivated in a similar manner as in Section 5.6. The definitions only are given here. This is done in terms of the unique steady state probability vector shown to exist in the previous section:

$$p = [p_0, p_1, p_2, \cdots, p_{m-1}]^T \tag{6.16}$$

6.5.1 Utilisation Performance

Let t_j be the mean length of state j

$$t_j \triangleq \begin{cases} D_2 + \dfrac{1}{\mu_g} & , \quad j \neq 0 \quad , \quad g = hp(j) \\ D_1 + \dfrac{1}{\sum_{h=1}^{k} \lambda_h} & , \quad j = 0 \end{cases} \tag{6.17}$$

Then

$$PROP(h) \triangleq \frac{\sum_{\substack{i \text{ such that} \\ hp(i) = h}} \dfrac{p_i}{\mu_h}}{\sum_{i=0}^{m-1} p_i t_i} \tag{6.18}$$

$$\text{IDLE} \triangleq 1 - \sum_{h=1}^{k} \text{PROP}(h) \tag{6.19}$$

6.5.2 Mean Waiting Times

The analysis of MWT(h) is complicated by the fact that the waiting times may extend over an unbounded number of states for lower priority requesters. The approach taken is to assume that request h is serviced in the n^{th} state = j and then evaluate cmwtf(j,h), the conditional mean waiting time of the fixed priority arbiter model given that requester h receives service in state = j. It follows that in the limit as $n \to \infty$

$$\text{MWT}(h) = \lim_{n\to\infty} \sum_{\substack{j \text{ such that} \\ h = hp(j)}} \text{cmwtf}(j,h)\ \text{prob}(n^{th}\text{state} = j \mid h \text{ serviced in } j)$$

$$= \frac{\displaystyle\sum_{\substack{j \text{ such that} \\ h = hp(j)}} \text{cmwtf}(j,h)\ p_j}{\displaystyle\sum_{\substack{j \text{ such that} \\ h = hp(j)}} p_j} \tag{6.20}$$

In Appendix H, an expression is derived for cmwtf(j,h).

6.5.3 Metastable Reliability Performance

The metastable failure rate of the distributed daisy-chained fixed priority arbiter shown in Figure 4.7 is evaluated in this section along the same lines as in Section 5.6.3. The same definition of failure is employed here, namely the arbiter fails if an Ack output is sensitised to a batch exhibiting metastable behaviour. Also, the same aperture

modelling is employed as in Section 5.6.3. Aperture events are assumed to occur just after the start of a state, as shown in Figure 6.2 in an analogous manner to the batched case.

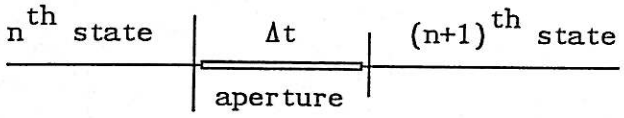


FIGURE 6.2 Assumed Position of Aperture.

It can be observed that aperture events associated with allocated settling times of τ_2 before failure results occur only at the start of non zero states and in modules with priority higher than all pending requests. The quantity apert(i) is defined as the sum of the request rates of those requesters capable of causing failure at the beginning of state i:

$$\text{apert}(i) \triangleq \begin{cases} \sum\limits_{\substack{h \notin i \\ h < hp(i)}} \lambda_h \quad , & i \neq 0 \\ 0 \quad , & i = 0 \end{cases} \tag{6.21}$$

it follows in an exactly analogous way to Section 5.6.3 that

$$\text{NMFR} = \frac{\sum\limits_{i=1}^{m-1} \text{apert}(i) p_i}{\sum\limits_{j=0}^{m-1} t_j p_j} \tag{6.22}$$

where t_j is defined in (6.17) and p_j is defined in (6.16). The

expression for NMFR can be generalised to dynamic priority schemes as will be seen in Chapter 8.

6.5.4 Limiting Performance Parameters

It is shown in Appendix I that

$$\lim_{\lambda \to 0} p = \lim_{N \to \infty} \lim_{\lambda \to 0} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \tag{6.23}$$

where the $\lim_{\lambda \to 0}$ is taken keeping request ratios fixed. Applying results of (6.8) and (6.9) gives

$$\lim_{\lambda \to 0} p_i = \begin{cases} \frac{1}{2} & , \quad i = 0 \\ \frac{r_g}{2} & , \quad |i| = \{g\} \quad , \quad g = 1, \ldots, k \end{cases} \tag{6.24}$$

It follows from (6.18)

$$\lim_{\lambda \to 0} \text{PROP}(h) = 0 \quad , \quad h = 1, \ldots, k \tag{6.25}$$

and from (6.19)

$$\lim_{\lambda \to 0} \text{IDLE} = 1$$

and from Appendix H, that

$$\lim_{\lambda \to 0} \text{MWT}(h) = D_1 \tag{6.26}$$

and from (6.22) and (6.21) that

$$\lim_{\lambda \to 0} \text{NMFR} = 0 \tag{6.27}$$

The heavy loading limits are divided into two cases: (i) $D_2 = 0$ and (ii) $D_2 > 0$.

Case (i) $D_2 = 0$

In Appendix I it is shown that for fixed request ratios

$$\lim_{\lambda \to \infty} p = \lim_{N \to \infty} \lim_{\lambda \to \infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \tag{6.28}$$

Applying results from (6.13) and (6.15) it follows that

$$\lim_{\lambda \to \infty} p_i = \begin{cases} \frac{1}{2} & , \quad i = \text{full state} \setminus \{g\} \quad , \quad g = 1 \text{ or } 2 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.29}$$

It follows from (6.18) that

$$\lim_{\lambda \to \infty} \text{PROP}(h) = \begin{cases} \frac{\mu_2}{2(\mu_1 + \mu_2)} & , \quad h = 1 \\ \frac{\mu_1}{2(\mu_1 + \mu_2)} & , \quad h = 2 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.30}$$

That is, only the highest two priority requesters share the resource in proportion to their mean service times.

From (6.19) it follows that

$$\lim_{\lambda \to \infty} \text{IDLE} = 0 \tag{6.31}$$

and from Appendix H, that

$$\lim_{\lambda \to \infty} \text{MWT}(h) = \begin{cases} \frac{1}{\mu_2} & , \quad h = 1 \\ \frac{1}{\mu_1} & , \quad h = 2 \\ \infty & , \quad \text{otherwise} \end{cases} \tag{6.32}$$

That is, all but the highest two priority requesters wait indefinitely without receiving service.

From (6.21), (6.22) and (6.29) it follows that

$$\lim_{\lambda \to \infty} \text{NMFR} = \lim_{\lambda \to \infty} \frac{\lambda_1 \mu_1 \mu}{\mu_1 + \mu_2} = \infty \tag{6.33}$$

The limit of NMFR is unbounded due to requester 1 always requesting in the aperture immediately following its service.

Case (ii) $D_2 > 0$

In Appendix I it is shown that for fixed request ratios and $D_2 > 0$

$$\lim_{\lambda \to \infty} p = \lim_{N \to \infty} \lim_{\lambda \to \infty} \frac{1}{N} \sum_{n=0}^{\infty} p(n) \tag{6.34}$$

Applying results from (6.12) and (6.14)

$$\lim_{\lambda \to \infty} p_i = \begin{cases} 1 & , \quad i = \text{full state} \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.35}$$

The following results apply from (6.35) and definitions of appropriate performance parameters

$$\lim_{\lambda \to \infty} \text{PROP}(h) = \begin{cases} \dfrac{1}{1 + \mu_1 D_2} & , \quad h = 1 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{6.36}$$

$$\lim_{\lambda \to \infty} \text{IDLE} = \frac{\mu_1 D_2}{1 + \mu_1 D_2} \tag{6.37}$$

$$\lim_{\lambda \to \infty} \text{MWT}(h) = \begin{cases} D_2 & , \quad h = 1 \\ \infty & , \quad \text{otherwise} \end{cases} \tag{6.38}$$

$$\lim_{\lambda \to \infty} \text{NMFR} = 0 \tag{6.39}$$

NMFR approaches zero because requester 1 is always pending at the start of each state.

6.6 COMPUTER STUDY AND RESULTS

Intermediate request loading between the light and heavy request loading limits is investigated in the numerical results presented in this section. Also, the limiting behaviour predicted in Section 6.5.4 can be observed in these results. All requesters are assumed identical as follows

$$\begin{aligned} \mu_h &= \mu \\ \lambda_h &= \lambda \quad , \quad h = 1, 2, \ldots, k \end{aligned} \tag{6.40}$$

Along the lines of Section 5.7, three sets of values of D_1 and D_2 are selected to cover a wide range of applications

$$\begin{aligned} &\text{(i)} \quad D_1 = D_2 = 0 \\ &\text{(ii)} \quad D_1 = D_2 = 0.2 \\ &\text{(iii)} \quad D_1 = D_2 = 1 \end{aligned} \tag{6.41}$$

6.6.1 Proportion of Time Allocated to Each Requester

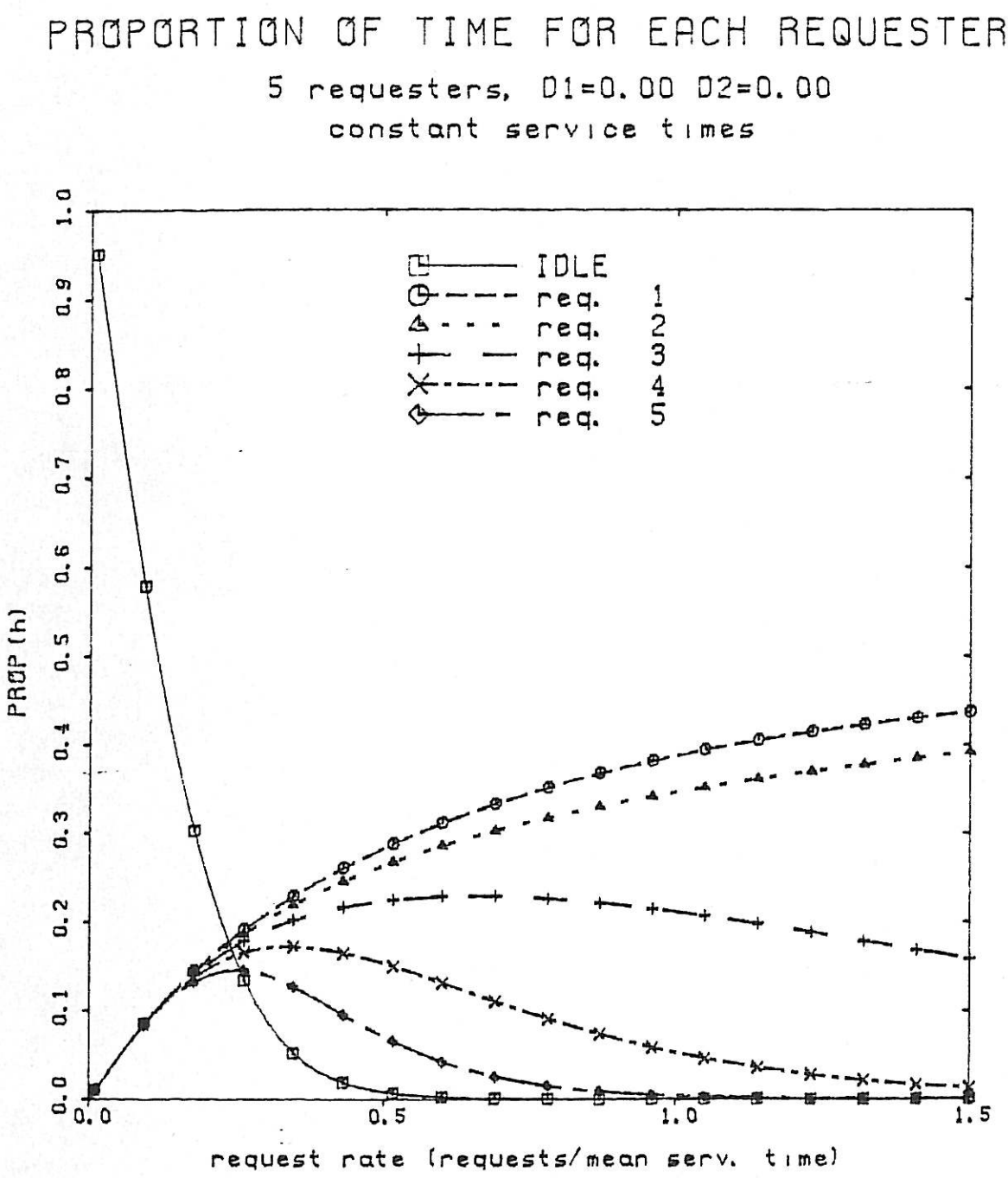
Graphs 6.1, 6.2, 6.3, 6.4 and 6.5 show PROP(h) and IDLE results against the request rate. It is clear from the results that low priority requesters are given considerably less proportion of time than higher priority requesters. A comparison of these results with the corresponding results for batched arbiters in the previous chapter is worthwhile : As the inter-service times increase, the fixed priority non batched arbiter distributes the resource less evenly as shown in Graphs 6.1, 6.3 and 6.5. This effect is due to the higher priority requesters having more time to request before the decision point after each service when D_2 is larger. High priority hogging of the resource results. In contrast, increasing inter-batch times of the batched arbiter results in a *more* even distribution of the resource. The difference can be attributed to the decision point occurring after *low* priority services in the batched arbiter, and after *high* priority services in the non batched arbiter.

Graphs 6.1 and 6.2 show the effect of the service distribution on PROP(h) results. The effect of the exponential distribution is to slightly decrease the level of saturation in the arbiter for a given mean request rate and mean service time. This can be explained in terms of introducing more randomness in the arbiter request-service cycle as discussed in Section 5.7.2 where the same situation arises for the batched

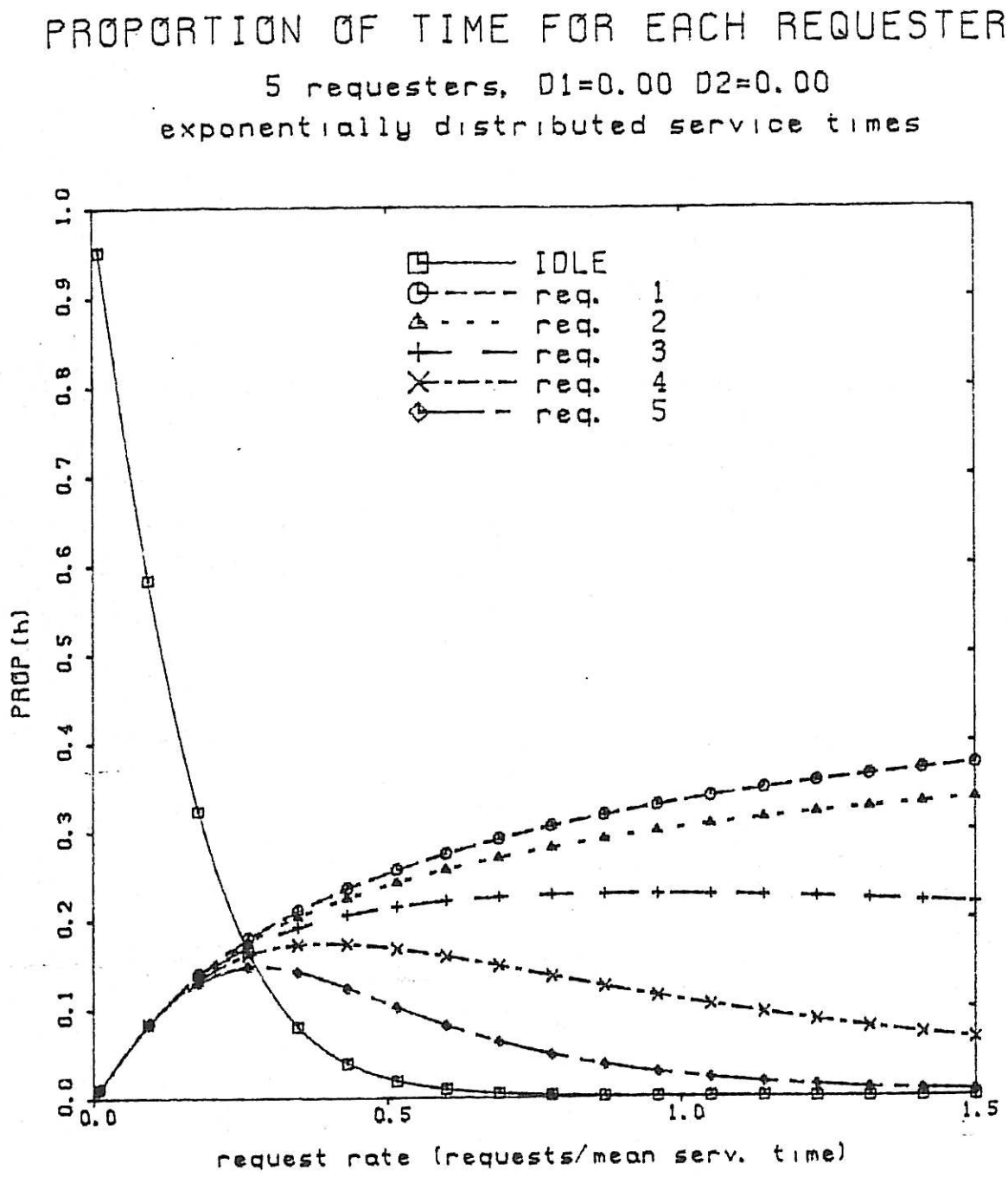
arbiter. The IDLE time is slightly higher for the exponentially distributed service times case, as can be seen from Graphs 6.1 and 6.2. Since the mean length of zero state is $1/k\lambda$, independently of the service time distributions, the zero state occurs slightly more frequently when service times are exponentially distributed. This is consistent with more "random" behaviour. It is due to the higher frequency of idle periods that low priority requests receive slightly greater access to the resource as is observed in Graphs 6.1 and 6.2.

The IDLE time approaches zero as request rates increase in Graphs 6.1 and 6.2 since inter-service times are zero. When $D_2 = 0.2$, as in Graph 6.3, IDLE approaches $0.2/(1 + 0.2) = 0.167$ and $1/(1 + 1) = 0.5$ in Graph 6.5. When $D_1 = D_2$ the difference between IDLE and its limiting value is a measure of the mean time of "no request pending" and indicates the level of request loading on the arbiter. As inter-service times increase from Graph 6.1 to Graph 6.5, the loading increases for any given request rate due to "wasted" time of D_1 and D_2.

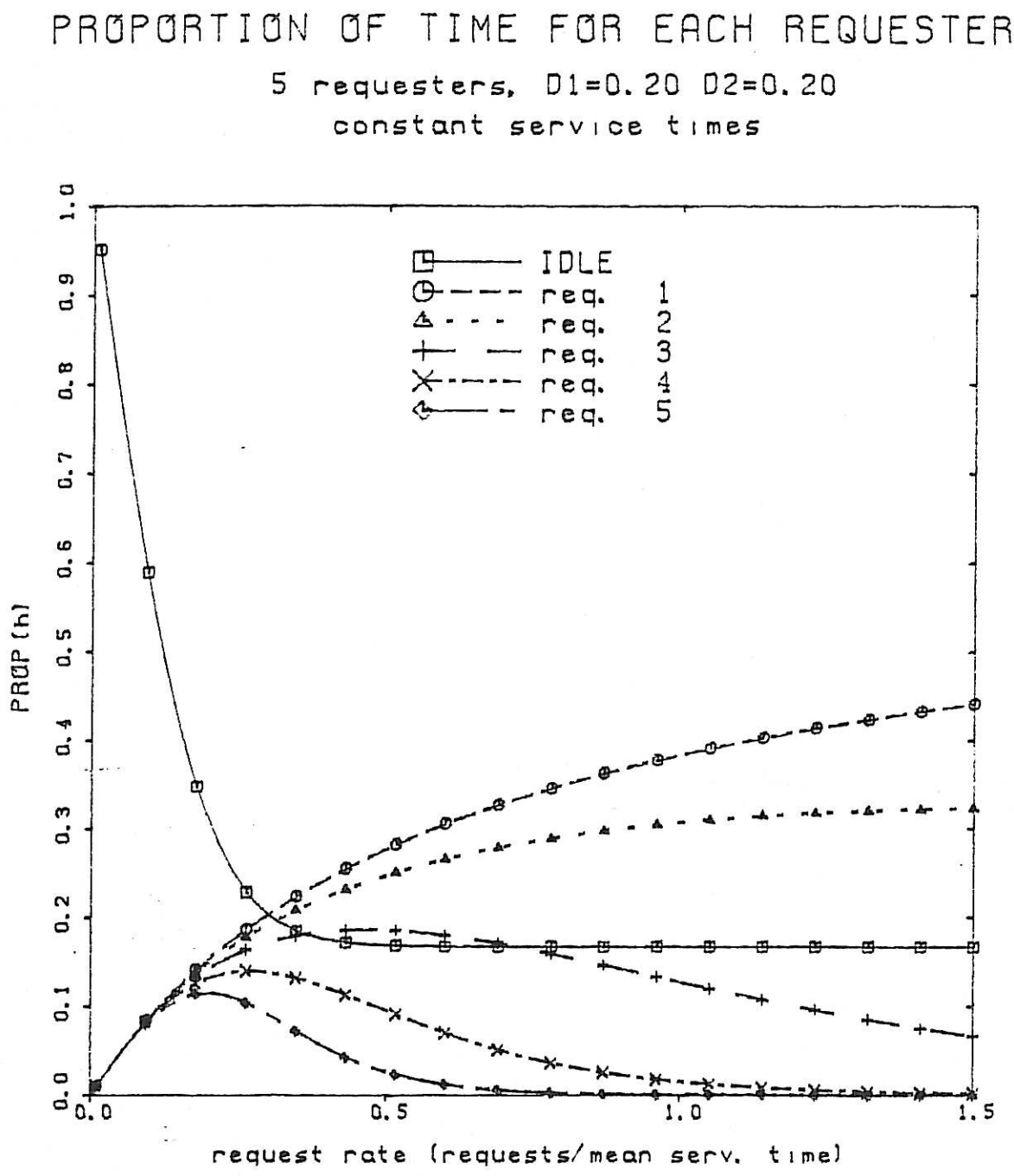
Graphs 6.3 and 6.4 compare the Markov results with Monte-Carlo simulation results and agreement within statistical variation is obtained.


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00
constant service times
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

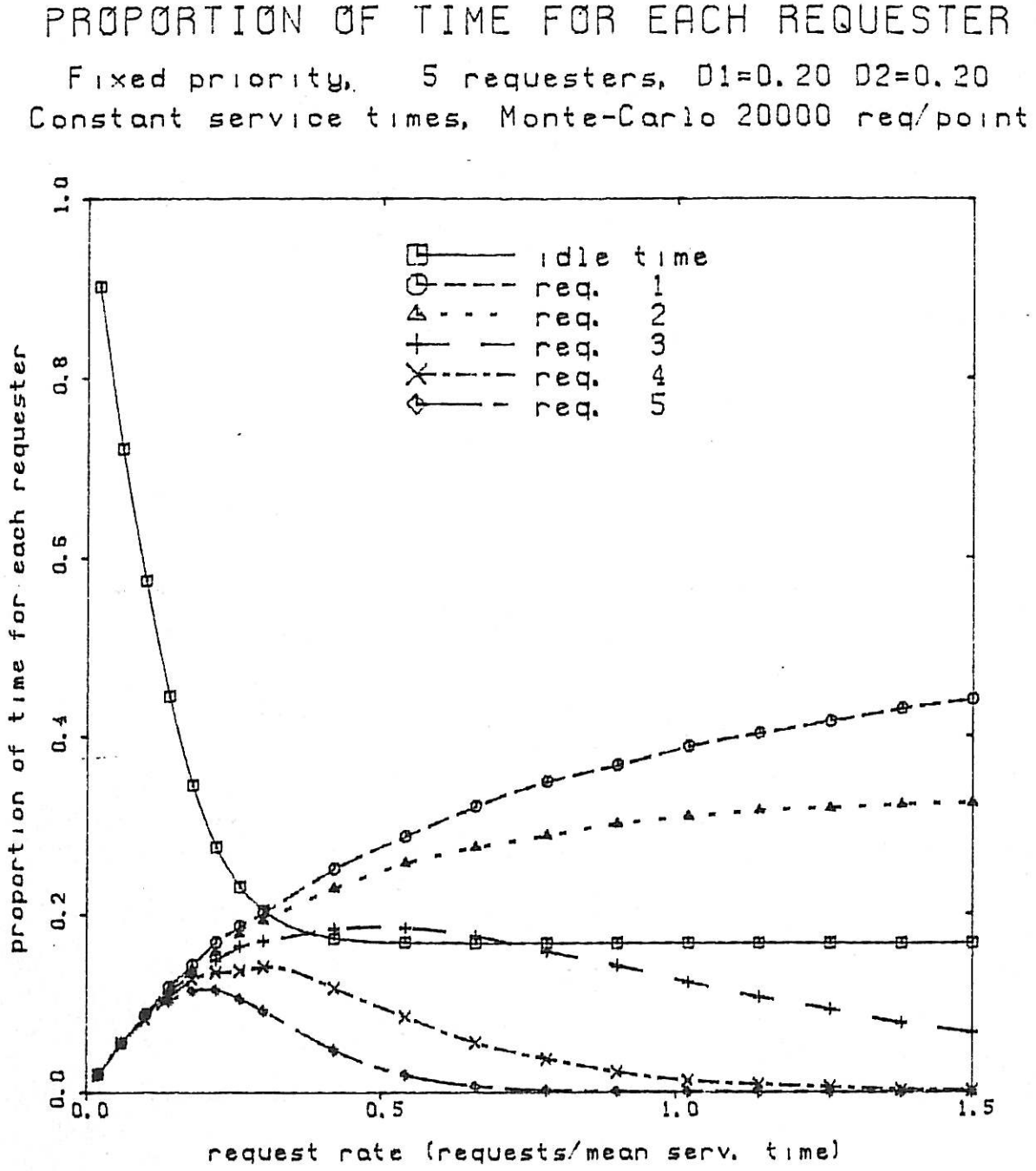
GRAPH 6.1


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00
exponentially distributed service times
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

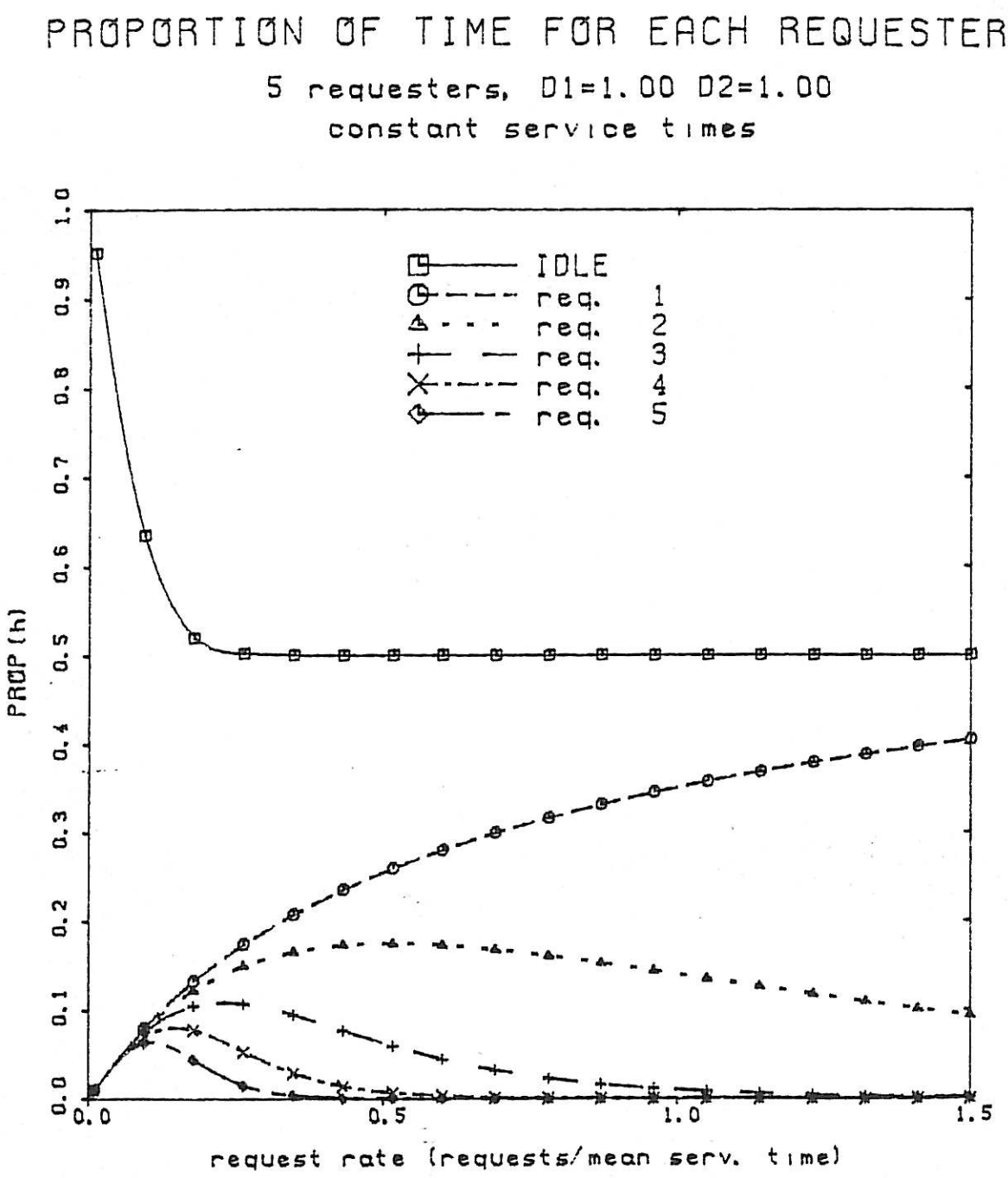
GRAPH 6.2


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20
constant service times
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 6.3


PROPORTION OF TIME FOR EACH REQUESTER
Fixed priority, 5 requesters, D1=0.20 D2=0.20
Constant service times, Monte-Carlo 20000 req/point
idle time
req. 1
req. 2
req. 3
req. 4
req. 5
proportion of time for each requester
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 6.4


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, D1=1.00 D2=1.00
constant service times
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

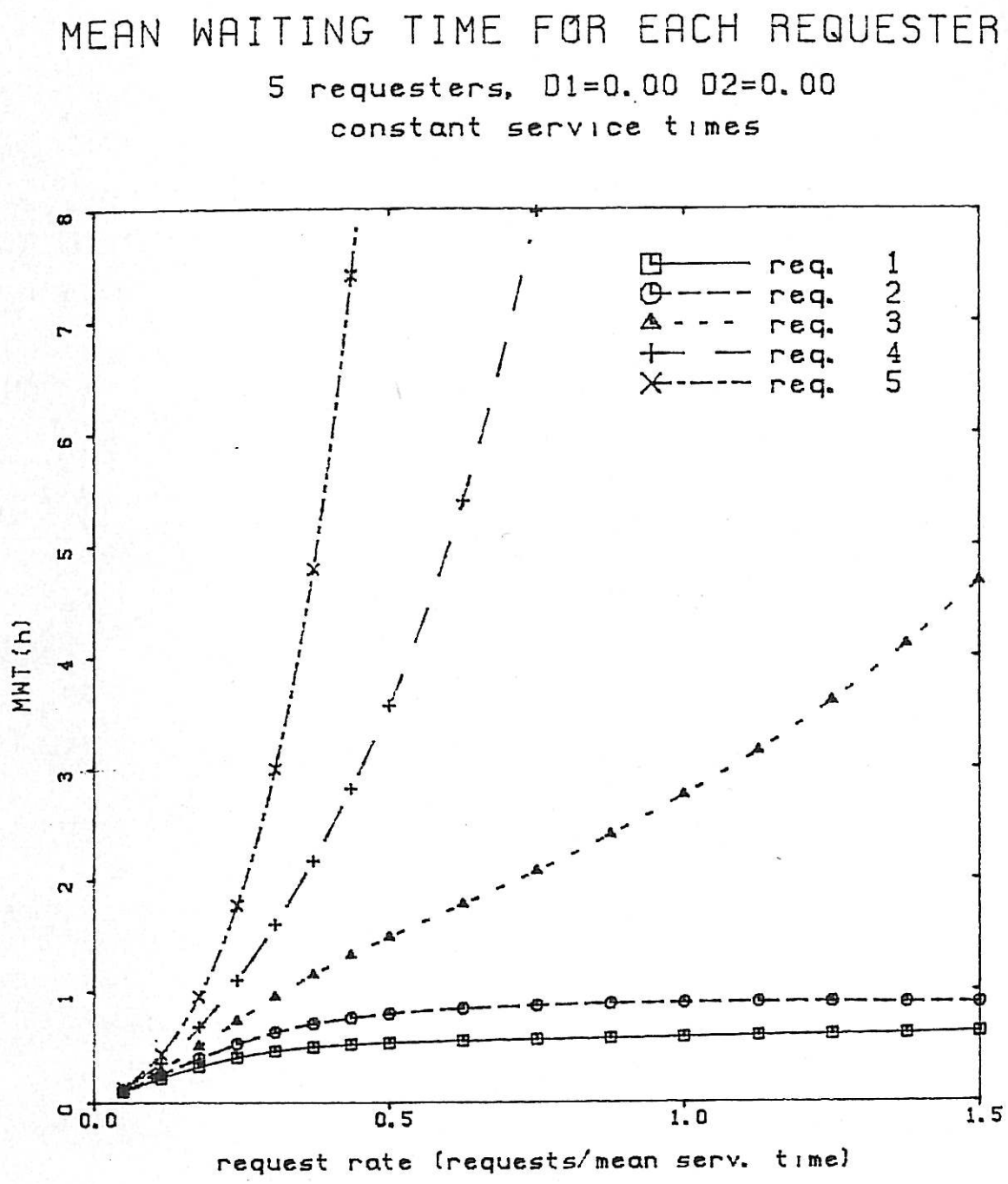
GRAPH 6.5

6.6.2 Mean Waiting Time for Each Requester

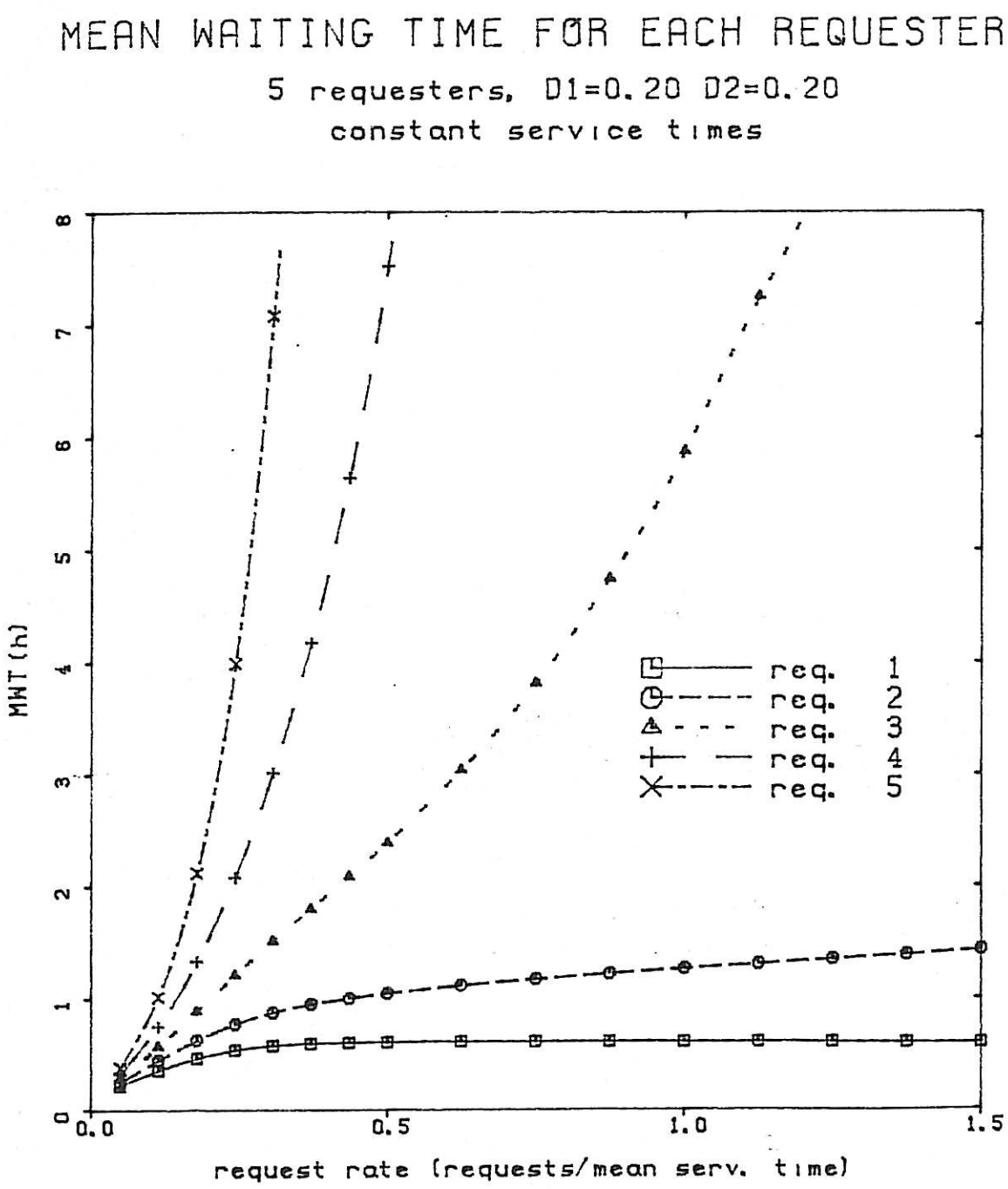
Graphs 6.6, 6.7, 6.8, 6.9 and 6.10 show MWT(h) plotted for each requester in 5 request fixed priority arbiters. The most striking feature is the dramatic increase in MWT(h) for $h > 2$. These lower priority requesters are locked out of consideration by the two highest priority requesters hogging the resource. The time between neither requester 1 nor 2 accessing the resource increases dramatically as request rates increase, and so requester 3 must wait during these busy periods. The situation is worsened as inter-service times increase because the probability of consecutive services for requester 1 increases. In Graph 6.10, the large value of D_2 equalling 1.00 causes requester 2 to be locked out by requester 1 being serviced consecutively. As a result of requester 1 lodging requests before the decision point with greater probability as the request rate increases, its mean waiting time decreases at the penalty of other requesters.

A comparison between exponential and constant service times can be seen from Graphs 6.7 and 6.9. As expected from previous discussion, lower priority waiting times decrease sligthly in the exponential case due to more "openings" appearing in the higher priority request traffic because of the more random request-service cycle.

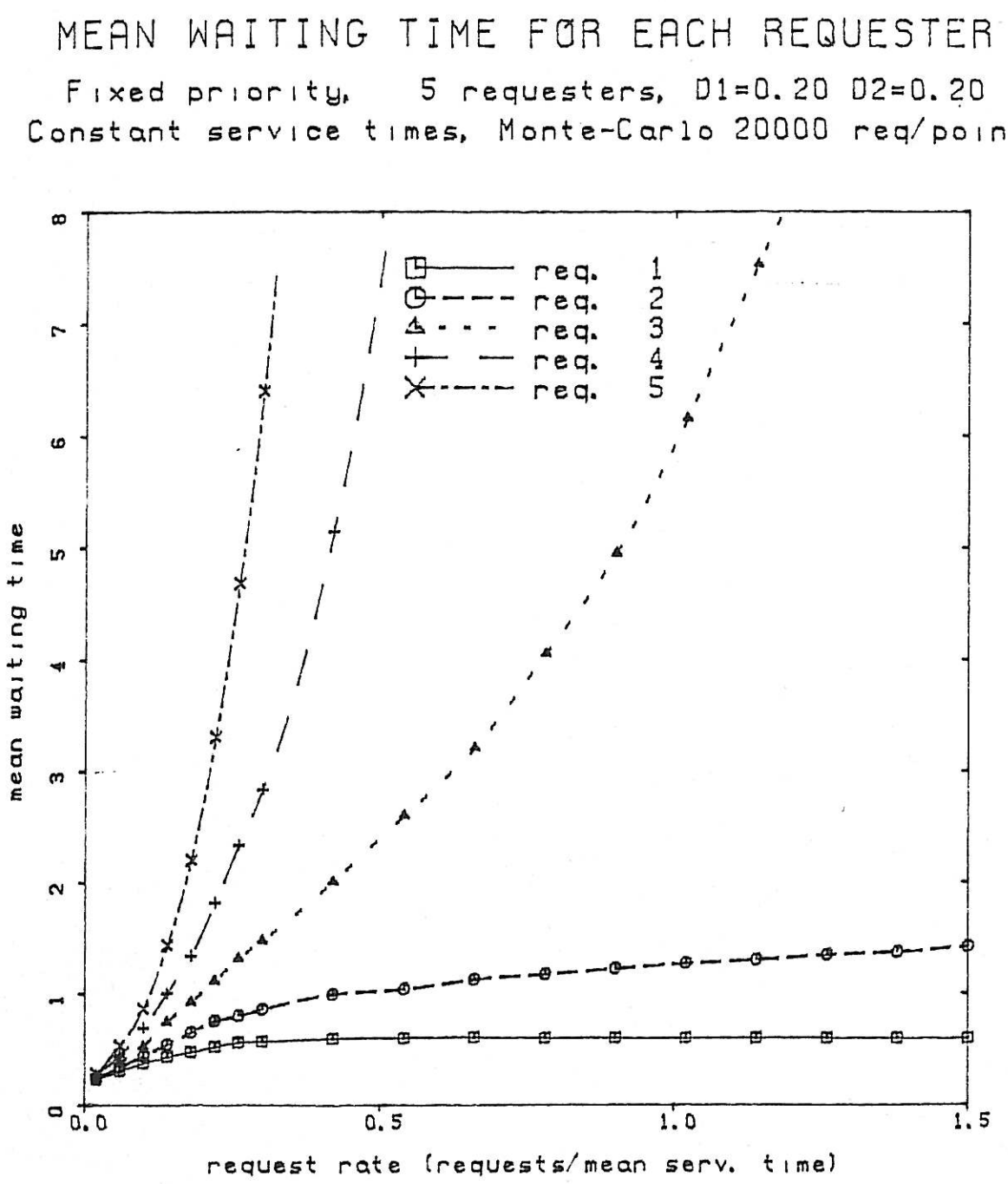
As a further reassurance, ~~to the critical reader,~~ Monte-Carlo simulation results agree well with the Markov results as seen in Graphs 6.7 and 6.8.


MEAN WAITING TIME FØR EACH REQUESTER
5 requesters, D1=0.00 D2=0.00
constant service times
req. 1
req. 2
req. 3
req. 4
req. 5
MWT(h)
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

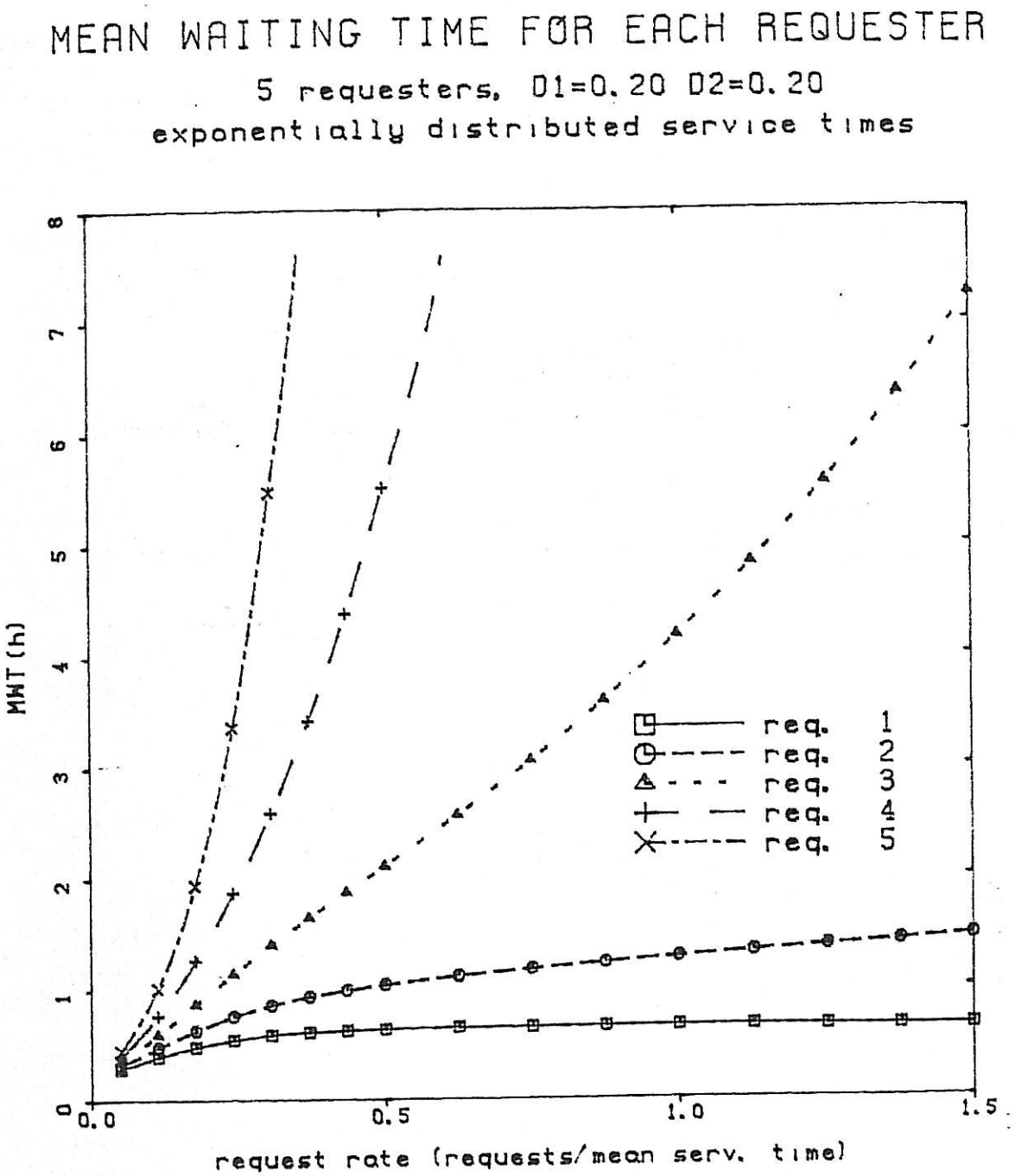
GRAPH 6.6


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20
constant service times
MWT (h)
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

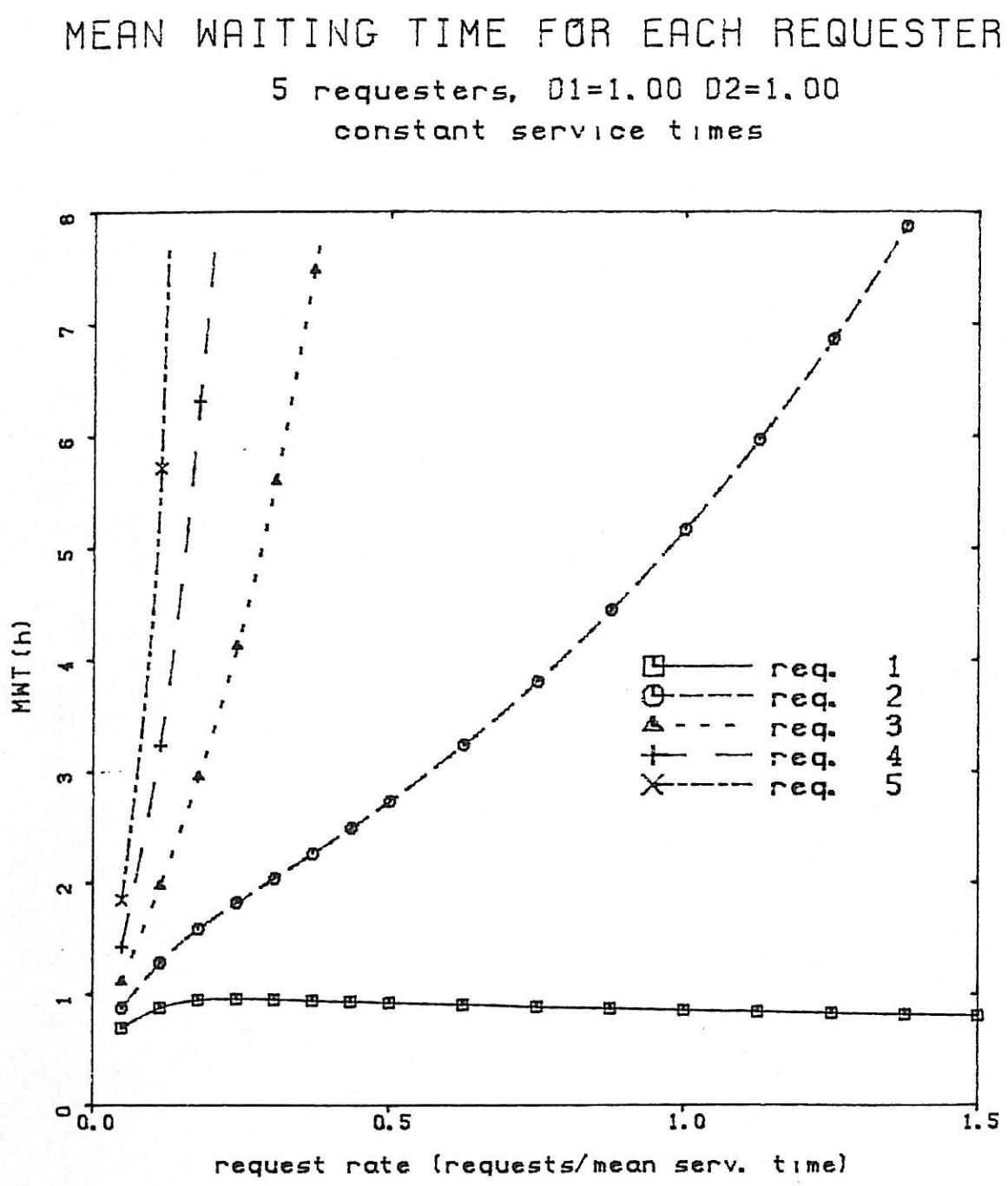
GRAPH 6.7


MEAN WAITING TIME FOR EACH REQUESTER
Fixed priority, 5 requesters, D1=0.20 D2=0.20
Constant service times, Monte-Carlo 20000 req/point
req. 1
req. 2
req. 3
req. 4
req. 5
mean waiting time
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 6.8


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=0.20 D2=0.20
exponentially distributed service times
MWT (h)
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

GRAPH 6.9


MEAN WAITING TIME FOR EACH REQUESTER
5 requesters, D1=1.00 D2=1.00
constant service times
MWT (h)
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5

GRAPH 6.10

6.6.3 Metastable Failure Rate

Graphs 6.11, 6.12, 6.13 and 6.14, show the normalised metastable failure rate defined in equation (6.22) for fixed priority arbiters ranging from 2 requesters to 5. When the arbiter is lightly loaded ($\lambda << 1/(k-1)$), NMFR is a function λ^2 as follows : Each requester is equally likely to be serviced in a singleton state with idle periods before and after. The probability of a singleton state is approximately $1/(2k)$ for small λ as can be seen from (6.24). Also, the zero state has probability $1/2$ and mean duration of $1/k\lambda$ and so it follows from (6.22) that for $k\lambda << \dfrac{1}{\left[D_1 + D_2 + \frac{1}{\mu}\right]}$

$$NMFR \cong \frac{k(k-1)\lambda^2}{\left[2(1+k\lambda)\,D_1 + D_2 + \frac{1}{\mu}\right]} \qquad (6.42)$$

$$\simeq \frac{k(k-1)\lambda^2}{2}$$

This relationship is evident in Graphs 6.11 to 6.14 for light request loadings. As the arbiters saturate the failure rate changes characteristic since high priority requesters dominate. Assuming service alternates from requester 1 to 2, as is approximately the case for a heavily loaded arbiter with small inter-service times, NMFR will increase linearly as $\lambda/2$. This cannot be precisely observed in Graph 6.11, since lower priority requesters such as 3 and 4 in the 5 requester arbiter still receive a significant, and dropping, proportion of time as can be seen in Graph 6.1. This tends to diminish the increase of NMFR to about $\lambda/3$ as can be observed in Graph 6.11 for $\lambda > 0.5$ for $k = 5$. Note, however, that NMFR is unbounded when $D_1 = D_2 = 0$.

As D_1 and D_2 increase, NMFR decreases for heavy request loadings as can be seen in Graphs 6.11, 6.12 and 6.13. This can be attributed directly to requester 1 dominating the services, masking the possibility of metastable behaviour. When requester 1 has a request pending at a decision point no indecision or metastable behaviour can occur. In the extreme case of $D_1 = D_2 = 1.00$ as in Graphs 6.13 and 6.14, the effect of diminished NMFR is pronounced. As predicted by limiting results of Section 6.5.4, NMFR approaches zero as $\lambda \to \infty$ for non zero D_2 as can be observed in the extended request rate scale of Graph 6.14. Note, also, in Graph 6.14 the convergence of all arbiters to the one decreasing NMFR for $\lambda > 1.5$. This can be explained by observing that requesters 3, 4 and 5 play little role for heavy request loadings because they are almost completely locked out of contention by requesters 1 and 2. The following approximation for NMFR assumes requester 1 is serviced nearly all the time and requester 2 is serviced whenever requester 1 has no request pending after D_2 following its service. All other requesters are assumed to play an unsignificant role.

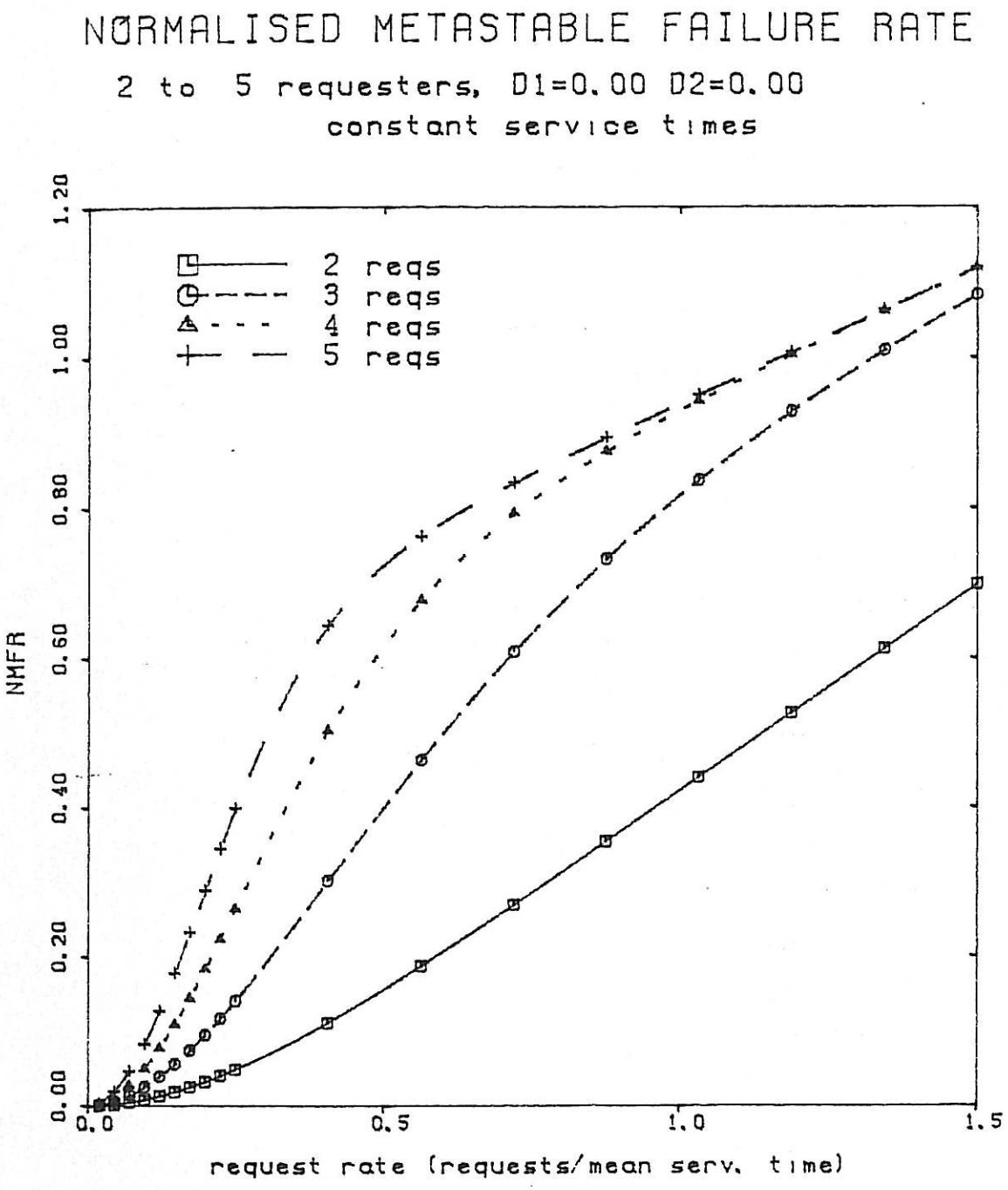
$$p_1 \simeq 1.0$$

$$p_2 \simeq e^{-\lambda D_2}$$

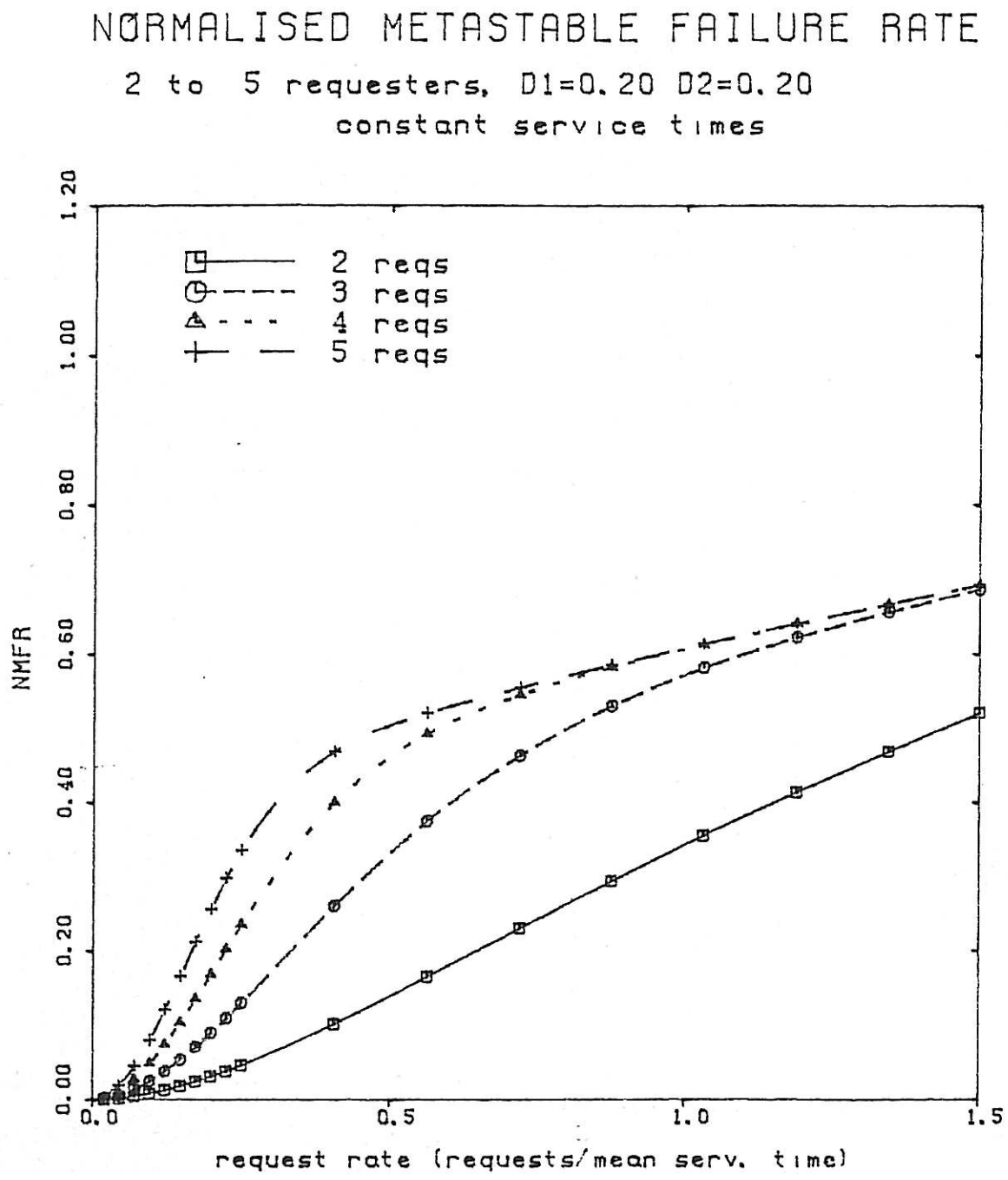
Thus, for $\lambda >> 1/(k-1)$ and $D_2 >> 0$

$$NMFR \simeq \frac{-\lambda D_2}{1 + D_2} \qquad (6.43)$$

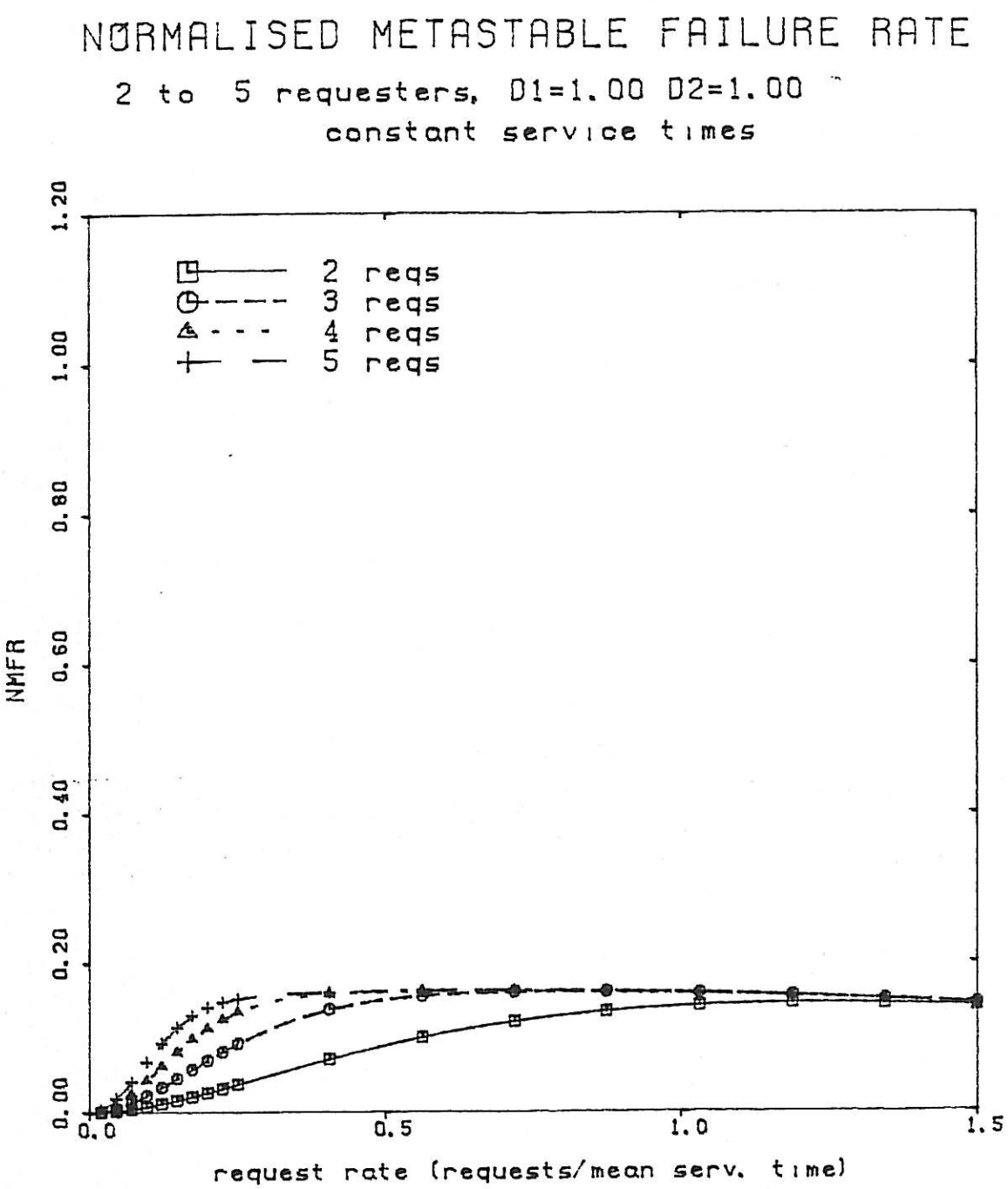
In Graph 6.14, NMFR is approximately described by the exponential decay of (6.43) for $\lambda > 1.5$.


NORMALISED METASTABLE FAILURE RATE
2 to 5 requesters, D1=0.00 D2=0.00
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.20
0.40
0.60
0.80
1.00
1.20
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

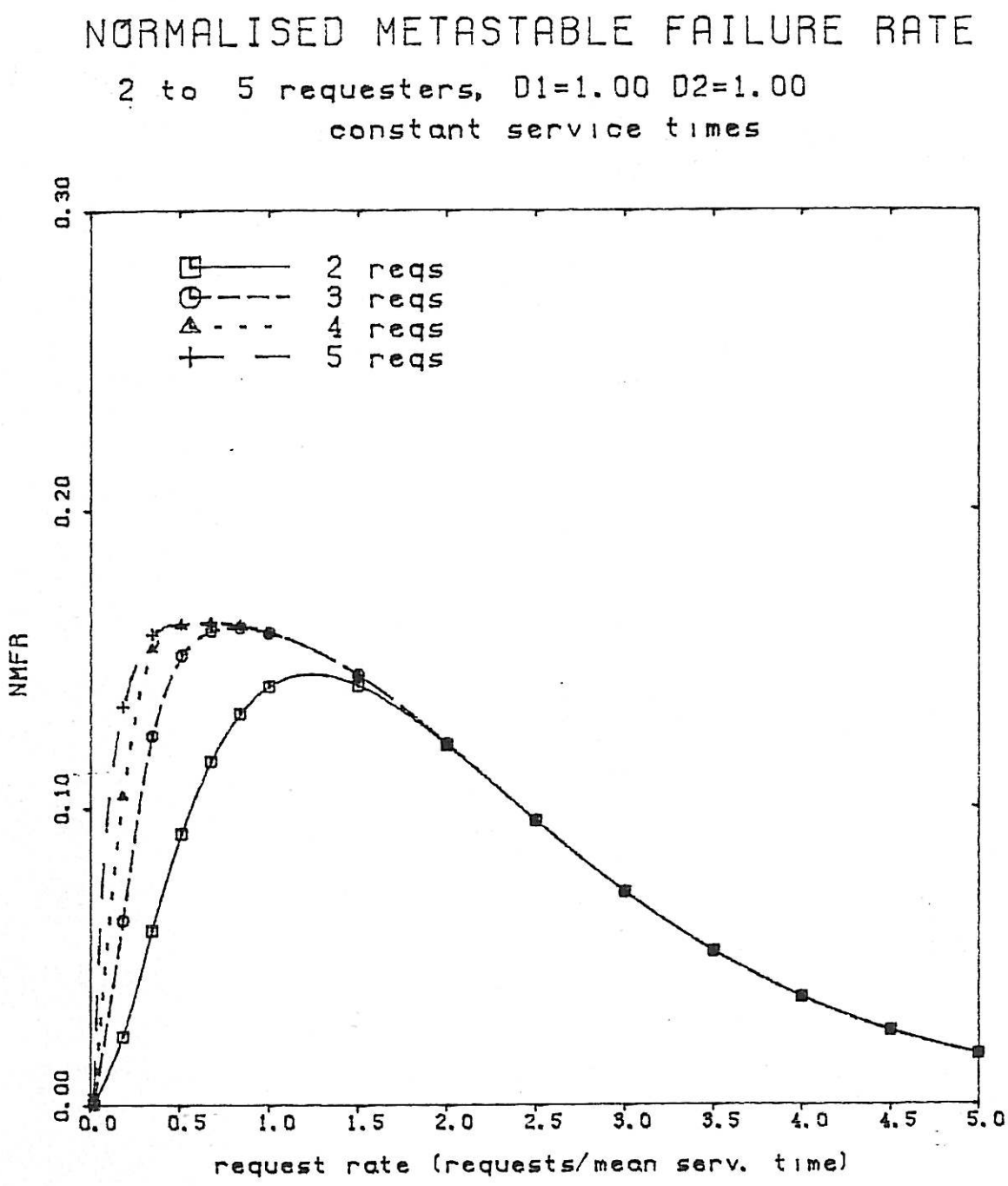
GRAPH 6.11


NORMALISED METASTABLE FAILURE RATE
2 to 5 requesters, D1=0.20 D2=0.20
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.20
0.40
0.60
0.80
1.00
1.20
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 6.12


NORMALISED METASTABLE FAILURE RATE
2 to 5 requesters, D1=1.00 D2=1.00
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.20
0.40
0.60
0.80
1.00
1.20
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 6.13


NORMALISED METASTABLE FAILURE RATE
2 to 5 requesters, D1=1.00 D2=1.00
constant service times
2 reqs
3 reqs
4 reqs
5 reqs
NMFR
0.00
0.10
0.20
0.30
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
request rate (requests/mean serv. time)

GRAPH 6.14

6.7 CONCLUSION

This chapter has presented theoretical and numerical results on the performance of non batched arbiters, in particular the fixed priority arbiter. The techniques of analysis in this chapter are similar to those in the previous chapter, but the results have proved to be in great contrast. The fixed priority arbiter has been shown to possess some characteristics not seen in batched arbiters that may be undesirable in many applications. Low priority requesters have been shown to be severely disadvantaged under moderate to heavy request loading in both throughput and response time sensitive applications. The mean waiting times for low priority requesters increases rapidly for request rates above $1/(k-1)$ requests for service time. In fact, the mean waiting time has been shown to be unbounded as request rates increase.

Another feature of fixed priority arbiters is monotonically increasing NMFR with request rates for small inter-service times. No peak failure rate occurs at realistic request rates unless significant inter-service times are present. The reader will recall from Chapter 5 this is in contrast with the peaking of NMFR found in batched arbiters. Chapter 8 pursues the comparisons of different arbitration disciplines further.

CHAPTER 7

CLOCKED BATCHED ARBITERS - MODELLING AND METASTABLE FAILURE MODES

7.1 INTRODUCTION

The previous two chapters have examined asynchronous arbiter circuits and their modelling. Since the design methodology for synchronous digital circuits is usually simpler, many arbiter circuits employ a clock to sequence internal state transitions. This chapter discusses the modelling, analysis and performance of clocked batched arbiters. Three specific examples of decentralised clocked batched arbiters are examined in detail. The three examples differ in the degree of synchronisation of the resetting strategy which determines the completion of service. Metastable behaviour modes are discussed in detail for each resetting strategy and a comparison of each stategy is carried out. Factors affecting the reliability are identified and design considerations relevant to metastable behaviour are presented. The analysis and modelling techniques developed by the author for the three examples enable insight to be obtained into clocked arbiters in general and some general observations are stated in the conclusions of the chapter.

The organisation of the chapter is as follows: In Section 7.2, examples of centralised and decentralised clocked batched arbiters are presented. In Section 7.3 modelling assumptions and definitions are clearly stated to allow a mathematical basis for Section 7.4, where the zero to non zero batch transition probabilities are derived. These transition probabilities are the same for each resetting strategy. In Sections 7.5, 7.6 and 7.7, the three resetting strategies are modelled and

analysed for non zero to non zero batch transitions. With the state definition defined in Section 7.3, the direct resetting strategy model is non Markovian. However, in Section 7.7 an approximate Markovian model is developed and shown to be accurate through verification by Monte-Carlo simulation employing the exact timing model. In Section 7.8, the service performance of the three resetting strategies is compared and numerical results are presented from the Markov modelling. In Section 7.9 the metastable behaviour of the clocked batched arbiters is discussed. All failure modes are treated and the significance of various circuit features are identified. In Section 7.10, the metastability performance of the three resetting strategies is compared. The general conclusions concerning clocked arbiters are stated in Section 7.11.

7.2 EXAMPLES OF CLOCKED BATCHED ARBITERS

In this section some examples of clocked batched arbiters are introduced. These examples will be used later in this chapter to construct models to describe the behaviour of the clocked batched arbiters.

7.2.1 Example of a Centralised Clocked Batched Arbiter

An example of a centralised clocked batched arbiter is shown in Figure 7.1. With reference to Figure 7.2, the circuit operates as follows: Requests are synchronised to the clock by latch 1. One clock cycle is allowed for metastable settling of latch 1 before further latching occurs in latch 2. The purpose of latch 2 is to hold requests for priority resolution in the encoder-decoder circuit. The encoder has "don't care" conditions which ensure that only the highest priority

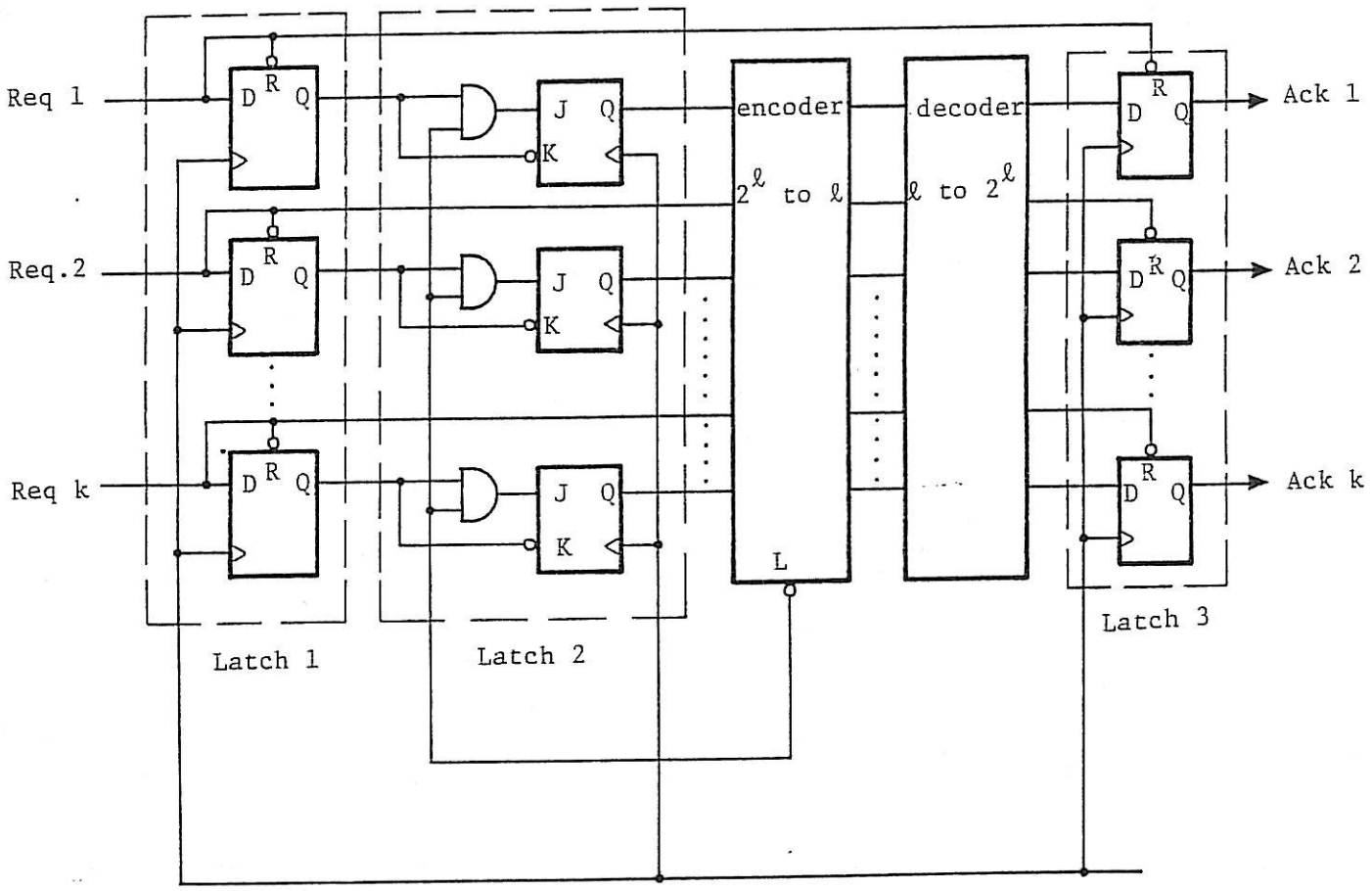


FIGURE 7.1 A Centralised Clocked Batched Arbiter.

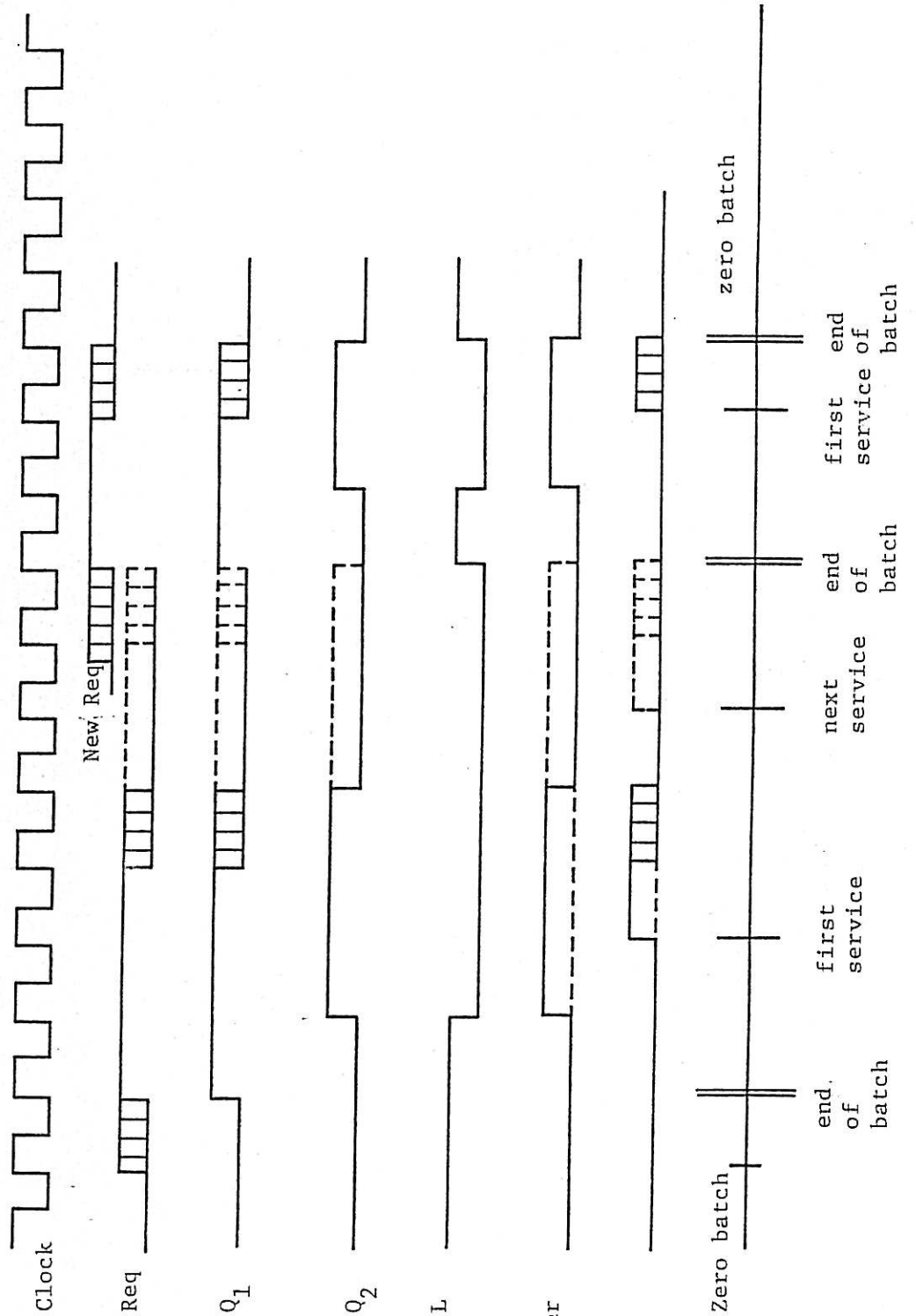


FIGURE 7.2 Timing of Centralised Clocked Batching Arbiter

request is enclosed. As a result, the decoder asserts only the highest priority output. This output is sampled on the next clock edge by latch 3 to prevent transient glitches from the encoder-decoder propagating to the acknowledge outputs.

Batching of requests is achieved by using a lock out mechanism in latch 2. The L output of the encoder is asserted low when at least one encoder input is high. This arrangement prevents any further requests being latched by latch 2 until all the currently latched requests have been serviced.

The dropping of a Req input causes direct resetting of latches 1 and 3, which speeds up passing on of the resource to the next priority requester waiting in a batch and drops the acknowledge signal as early as possible to allow immediate re-requesting. The issuing of the next Ack signal cannot be done immediately due to possible glitches on the decoder output, and hence latches 2, 3 must be used to synchronise the output transition. Latch 2 is not directly reset in order the allow metastable settling time of one clock cycle less propagational delay of the encoder-decoder.

7.2.2 Examples of Decentralised Clocked Batched Arbiters

This section introduces three forms of a decentralised clocked batched arbiter based on a design presented in [C.2]. Associated geographically and logically with each requester is a circuit module. The distributed modules are connected via a common line, a clock line and a daisy chain as shown in Figure 7.3.

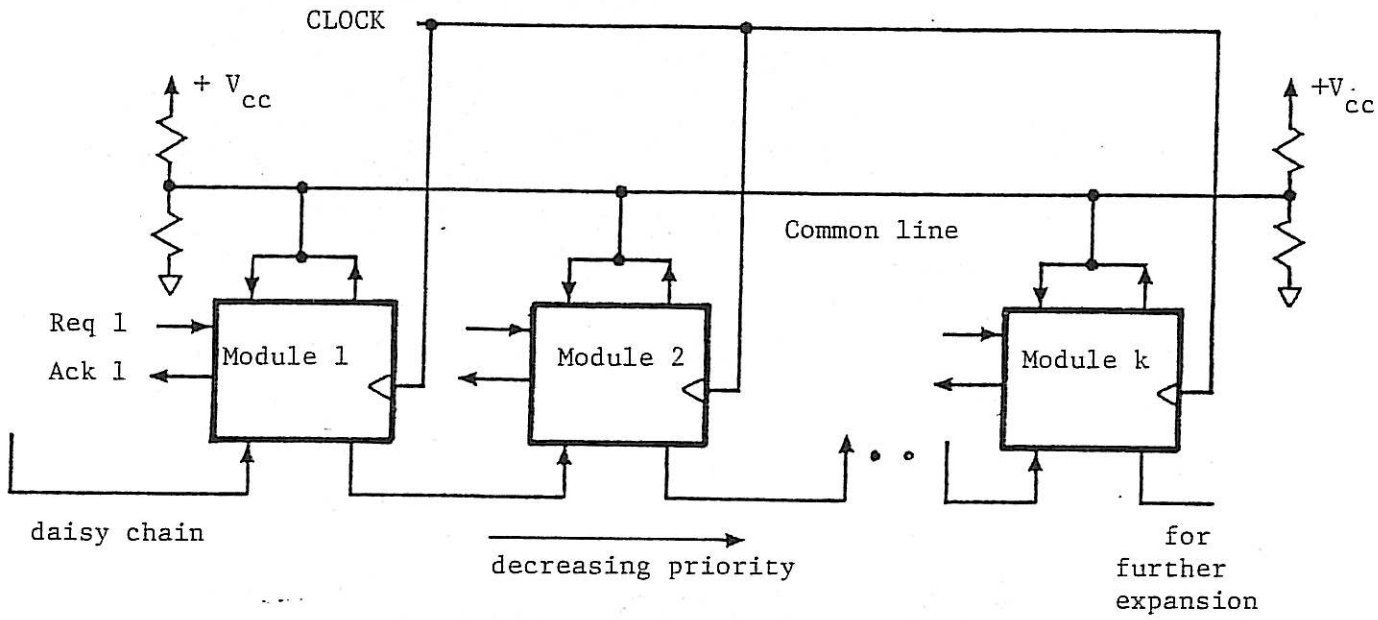


FIGURE 7.3 Module Interconnection Structure for a Decentralised Clocked Batched Arbiter.

The wired-or common line is asserted when at least one request is latched by a module and provides the lock out mechanism which ensures batching of requests. The daisy chain implements the priority structure within a batch of requests. The head of the daisy chain is connected to logic 1, and it is this end that marks the highest priority in the chain.

Details of a module are shown in Figure 7.4.

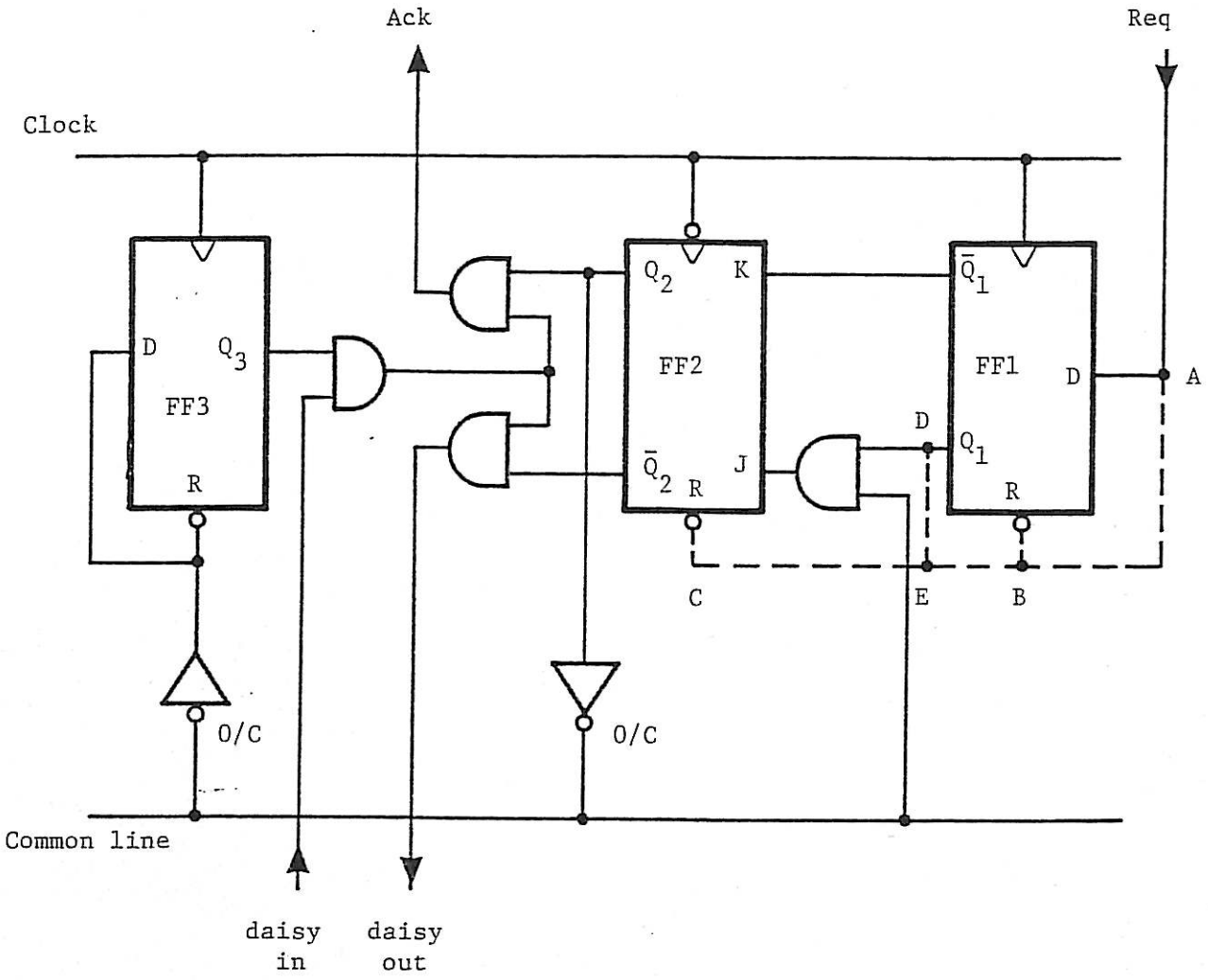


FIGURE 7.4 Module Circuit for Clocked Decentralised Batched Arbiter.

(i) Direct resetting - include ABC only
(ii) Half resetting - include DEC only
(iii) No resetting - delete all dashed lines.

The different forms of resetting request to flip-flops FF1 and FF2 are labelled. The effect of these different strategies is discussed later in this chapter.

7.3 MODELLING ASSUMPTIONS AND DEFINITIONS

This section presents assumptions and definitions relating to: the timing; request and service properties; and state definition for the clocked arbiters considered.

7.3.1 Timing Assumptions

An idealised model is adopted for the logic circuits. The effect of non ideal characteristics is discussed in Appendix J after the idealised model has been examined when the significant parameters can be clearly identified. The idealised assumptions are:-

(i) No clock skew between modules.

(ii) Zero rise and fall times.

(iii) Zero logic propagation delays.

(iv) Zero transmission delays along wires.

(v) All flip-flops have identical metastable characteristics.

7.3.2 Request and Service Modelling

The request-acknowledge protocol described in Chapter 2 applies in this chapter. Re-request times are assumed to be exponentially distributed as in Chapters 4,5 and 6. The service time for requester h, denoted $t_s(h)$, is assumed to be distributed arbitrarily with a density function denoted by f_h as in previous chapters.

A new time denoted $t_a(h)$, is defined as the duration of Ack h is asserted as illustrated in Figure 7.5. This is longer than $t_s(h)$ due to delays within the arbiter.

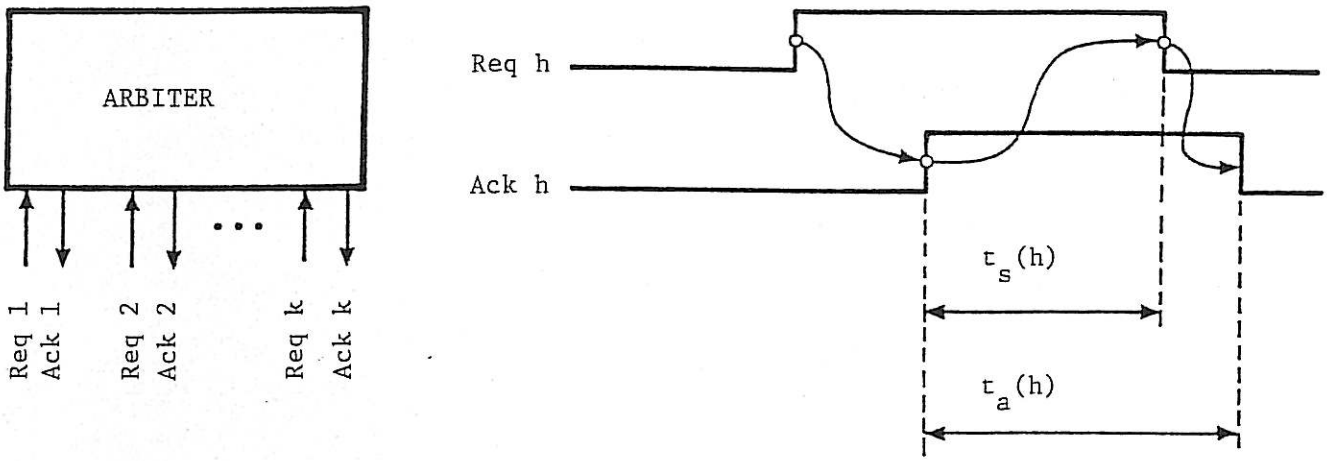


FIGURE 7.5 Arbiter Request-Acknowledge Protocol.

7.3.3 State Definition

As in Chapter 5 each batch has an associated state formed from the set of pending requests at the batching point or the end of the previous batch. The end of batch points will be clearly identified in timing diagrams. The n^{th} state is denoted by S(n).

7.4 ZERO TO NON ZERO BATCH TRANSITION

All clocked batched arbiter examples presented have the same zero to non zero batch transition probability because they all latch requests on the first rising clock edge after the first request is made during the idling period. Figure 7.6 shows the general form of the transition.

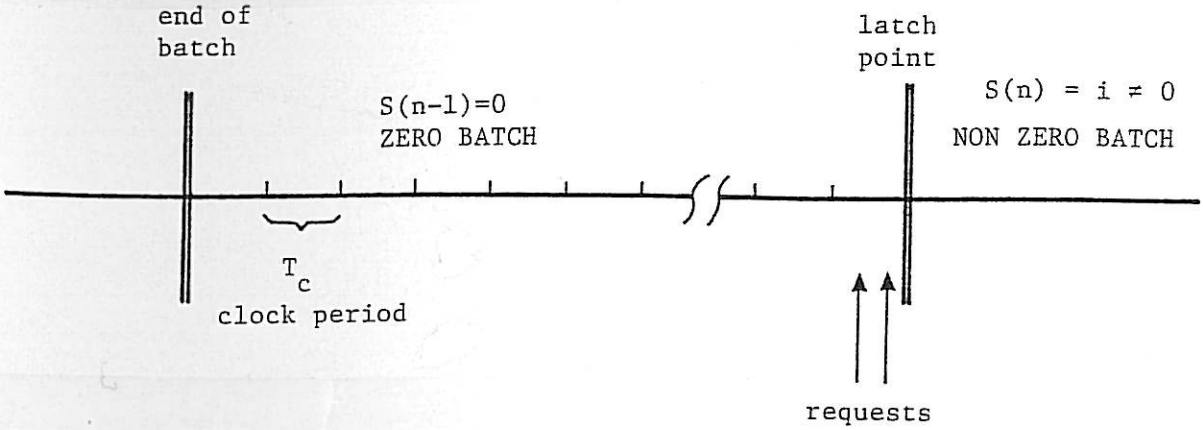


FIGURE 7.6 Zero to Non Zero Batch Transition.

In this section the probability that the n^{th} state is the non zero state i is derived given that the $n-1^{th}$ state is zero.

$$\text{prob}[S(n) = i \mid S(n-1) = 0]$$

$$= \sum_{d=1}^{\infty} \text{prob}\left[\begin{array}{l} S(n)=i \text{ and } S(n-1) \text{ lasts} \\ \quad d \text{ clock cycles} \end{array} \;\middle|\; S(n-1) = 0\right]$$

$$= \sum_{d=1}^{\infty} \prod_{h=1}^{k} e^{-\lambda_h T_c (d-1)} \prod_{g \in i} (1-e^{-\lambda_g T_c}) \prod_{f \notin i} e^{-\lambda_f T_c} \qquad (7.1)$$

where T_c = clock period. Equation (7.1) is an infinite geometric series with a term to term ratio of $\prod_{h=1}^{k} e^{-\lambda_h T_c}$. Since T_c is positive, the ratio is less than one and hence the series converges to

$$\text{prob}[S(n)=i \mid S(n-1)=0] = \frac{\prod_{g\in i}(1 - e^{-\lambda_g T_c}) \prod_{f\notin i} e^{-\lambda_f T_c}}{1 - \prod_{h=1}^{k} e^{-\lambda_h T_c}} \qquad (7.2)$$

Equation (7.2) can be checked by comparing its limiting case of $T_c \to 0$ with $\text{prob}[S(n)=i \mid S(n-1) = 0]$ in equation (5.9) for the unclocked model with $D_3 = 0$. From (7.2) it follows that:

$$\lim_{T_c\to 0^+} \text{prob}[S(n)=i \mid S(n-1)=0] = \lim_{T_c\to 0^+} \frac{\prod_{g\in S(n)} \lambda_g T_c \cdot \prod_{p\notin S(n)} 1}{\sum_{h=1}^{k} \lambda_h T_c}$$

$$= \begin{cases} \dfrac{\lambda_g}{\sum_{h=1}^{k} \lambda_h} & \text{, } S(n) \text{ a singleton state} \\ 0 & \text{, otherwise (see note 1)} \end{cases} \qquad (7.3)$$

Equation (7.3) agrees with the asynchronous model with $D_3=0$ in (5.9).

Note 1

If $g_1 \in S(n)$ and $g_2 \in S(n)$ and $g_1 \neq g_2$

then

$$\lim_{T_C\to 0^+} \text{prob}[S(n)=i \mid S(n-1)=0] = \lim_{T_c\to 0^+} \frac{\lambda_{g_1} \lambda_{g_2} T_c}{\sum_{h=1}^{k} \lambda_h} = 0$$

7.5 NO RESETTING CLOCKED BATCHED ARBITER

The distributed clocked batched arbiter of Section 7.2.2, Figure 7.3 and Figure 7.4 with ABCDE deleted is modelled and analysed in this section. The strategy of no resetting results in the synchronisation of dropping requests after service. This introduces delay in passing on the resource from requester to requester, since this can only occur at discrete times.

The timing diagram shown in Figure 7.7 illustrates the operation of the circuit for transitions between batches and allocation of the resource within batches. During a zero batch, requests which occur during the same clock period in which the first request occurs, are batched together and serviced in the batch following.

At the start of a non zero batch, a clock period occurs in which no Ack signal is asserted. This is designated D_2 and matches D_2 of the asynchronous batched arbiter model of Chapters 4 and 5. Following D_2 are the time periods $t_a(h_1)$ and $t_a(h_2)$ where requesters h_1 and h_2 are serviced (h_1 has a higher priority than h_2). Note that $t_a(h_1)$ is a function of the service time $t_s(h_1)$. A careful examination of the timing reveals that the first value of t_a in a batch is a different function of t_s than subsequent t_a values in the batch. This occurs because the first t_a starts on a rising clock edge whilst subsequent t_a periods start on a falling clock edge. It follows that

$$t_a(h) = \begin{cases} \left(\frac{3}{2} + \left[\frac{t_s(h)}{T_c}\right]\right)T_c & , \text{ h first in batch} \\ \left(1 + \left[\frac{t_s(h)}{T_c} + \frac{1}{2}\right]\right)T_c & , \text{ otherwise} \end{cases} \qquad (7.4)$$

where $[x] \triangleq$ greatest integer $\leq x$.

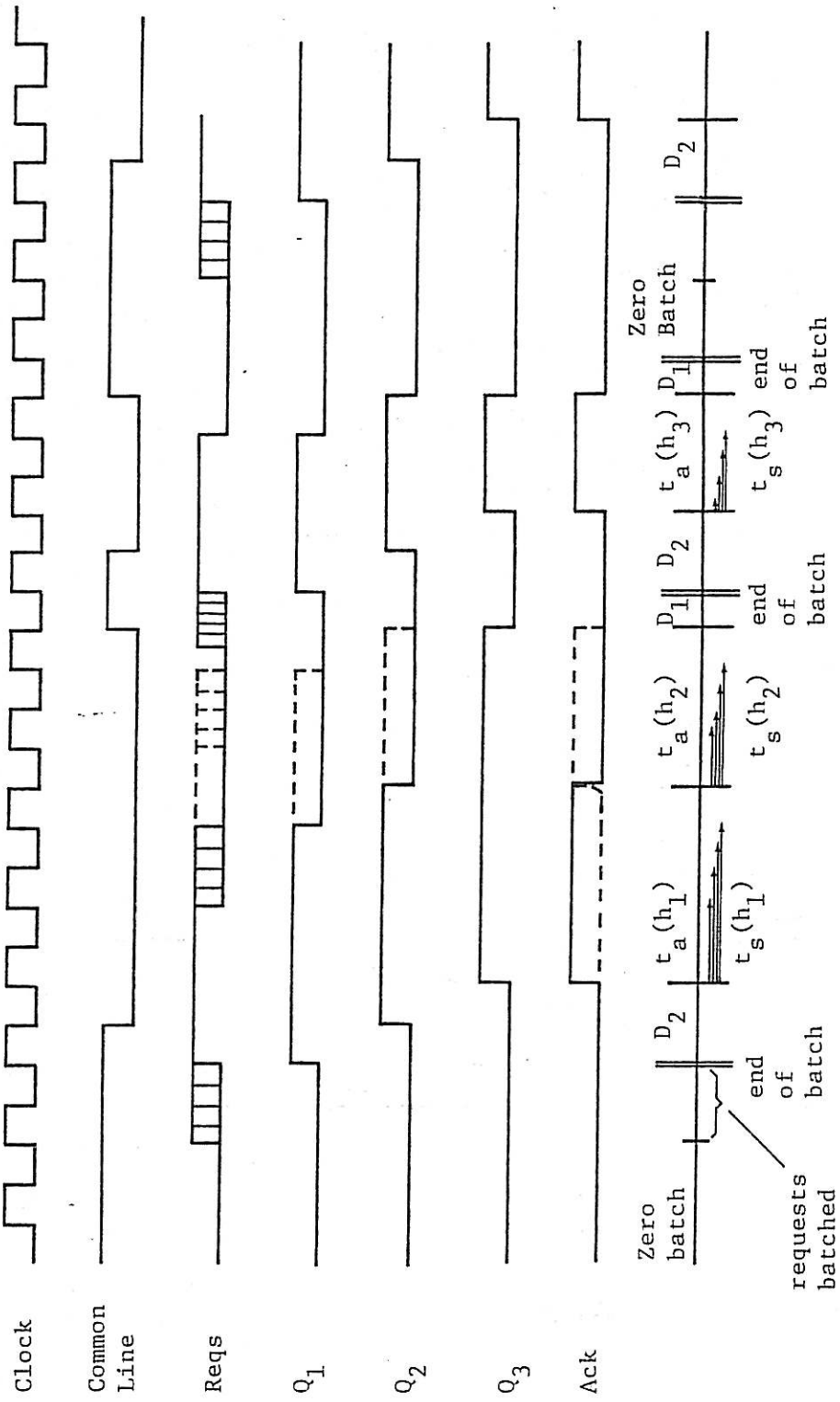


FIGURE 7.7 Timing of No Direct Resetting Clocked Batching Arbiter.

After all batched requests have been serviced, a time duration of half a clock cycle, labelled D_1, occurs before a new set of requests are once again latched. The asynchronous batched arbiter model has a corresponding time duration also labelled D_1. Note that the clocked model has no time duration corresponding to D_3 of the asynchronous model.

The differences between the asynchronous and no resetting clocked model of this section are:

(i) zero to non zero batch transition (see Section 7.4)

(ii) $D_2 = T_c$, $D_1 = \frac{T_c}{2}$, $D_3 = 0$ for clocked model

(iii) the service times are defined differently as in (7.4) for the clocked model.

With these modifications, the probability transitions is derived directly from the asynchronous model in Appendix K. Note that (7.4) implies that $t_a(h)$ will have a discrete probability distribution derived from the continuous distribution of $t_s(h)$. Also, $t_a(h)$ is larger than $t_s(h)$ from $T_c/2$ up to $3T_c/2$. This idle time is not present in the asynchronous arbiter, and occurs because of the synchronisation of dropping requests to the clock.

7.6 HALF RESETTING CLOCKED BATCHED ARBITER

The no resetting arbiter circuit of Section 7.5 wastes time due to synchronisation of dropping request during a batch. This section considers a modification to the module circuit to speed up this synchronisation process. In Figure 7.4 connection DEC directly resets FF2 once a dropping request is latched by FF1.

With reference to Figure 7.8, the following model is formed:

(i) Zero to non zero batch transition is described by Section 7.4.

(ii) An initial batch duration $D_2 = T_c$ occurs but no end of batch duration is present $(D_1 = 0)$.

(iii) The service times are given by

$$t_a(h) = (1 + \left[\frac{t_s(h)}{T_c}\right])T_c \qquad (7.5)$$

The time allocated for servicing a request, t_a, is now from zero to T_c larger than t_s which is an improvement of $T_c/2$ over the no resetting strategy of Section 7.5. Also a saving of $T_c/2$ occurs at the end of a batch due to the absence of D_1. The probability transition between states can easily be derived along the lines of Appendix K, with modifications to $t_a(h)$ as in (7.5) and putting $D_1 = 0$.

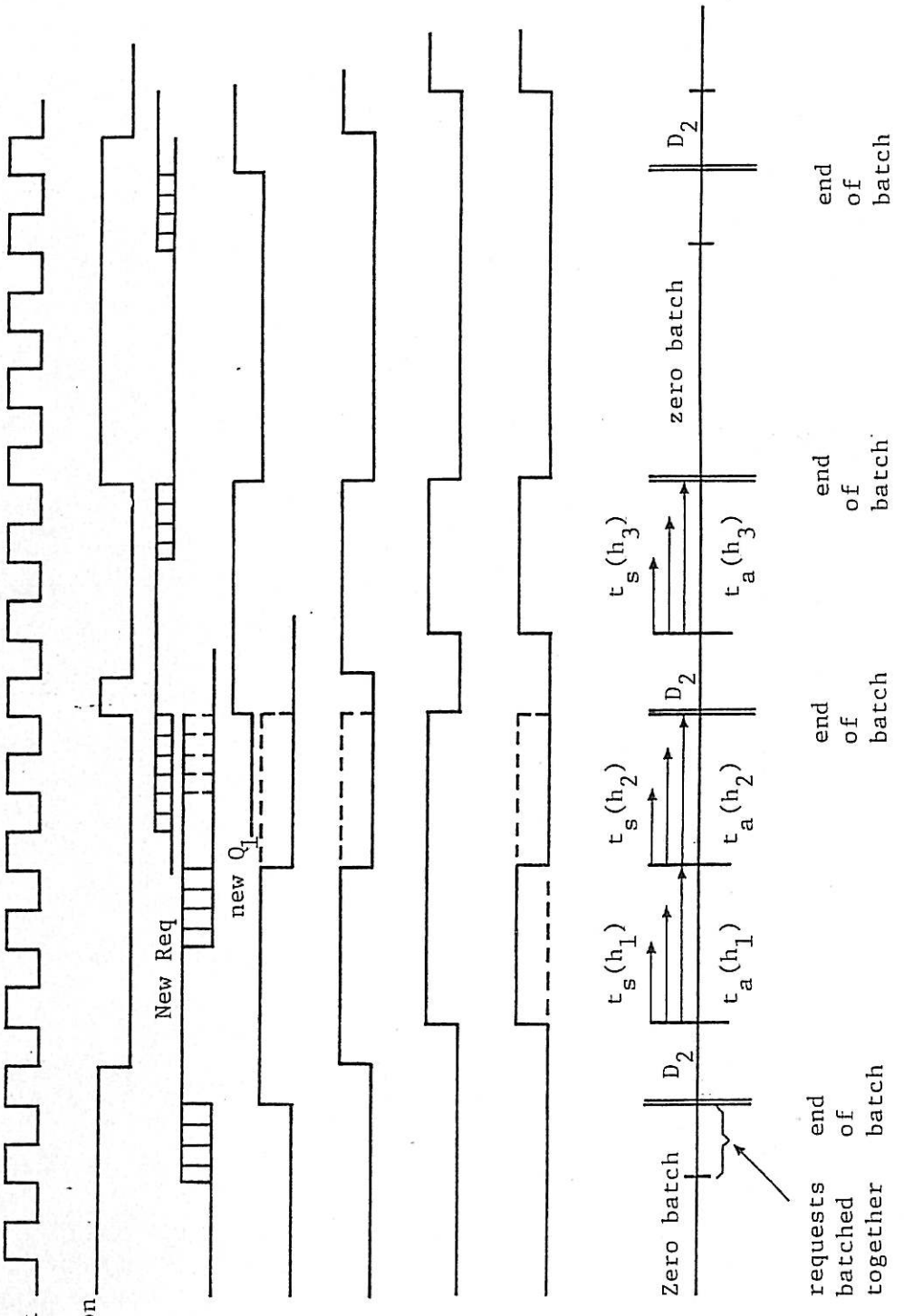


FIGURE 7.8 Timing of Half Resetting Clocked Arbiter.

7.7 DIRECT RESETTING CLOCKED BATCHED ARBITER

The direct resetting strategy is the most loosely synchronised of all three. The dropping of a request is no longer synchronised to the system clock, but directly results in the next latched request down the daisy chain being serviced. Flip-flops FF1 and FF2 in Figure 7.5 are directly reset via the connection ABC.

Figure 7.9 shows the timing of the direct resetting strategy.

A few points should be noted:

(i) The end of batch point is defined by the last time at which new requests can be latched so that they are serviced in the next batch. In Figure 7.9 this end of batch point can actually occur *before* the last requester has completed its service in the batch. Thus, an overlap in $t_a(2)$ and D_2 can occur, as shown. This will be discussed more fully in Section 7.7.1.

(ii) The end of batch always occurs on a rising clock edge. At this point the arbiter synchronises to the clock. Thus, each *batch* is synchronised to the clock, as opposed to each *service*, within a batch as in Sections 7.6 and 7.5.

(iii) As a result of (ii), no time is spent idling between servicing of requests within a batch. A transfer of the resource within a batch involves no delay necessary to synchronise.

Thus

$$t_a(h) = t_s(h) \qquad (7.6)$$

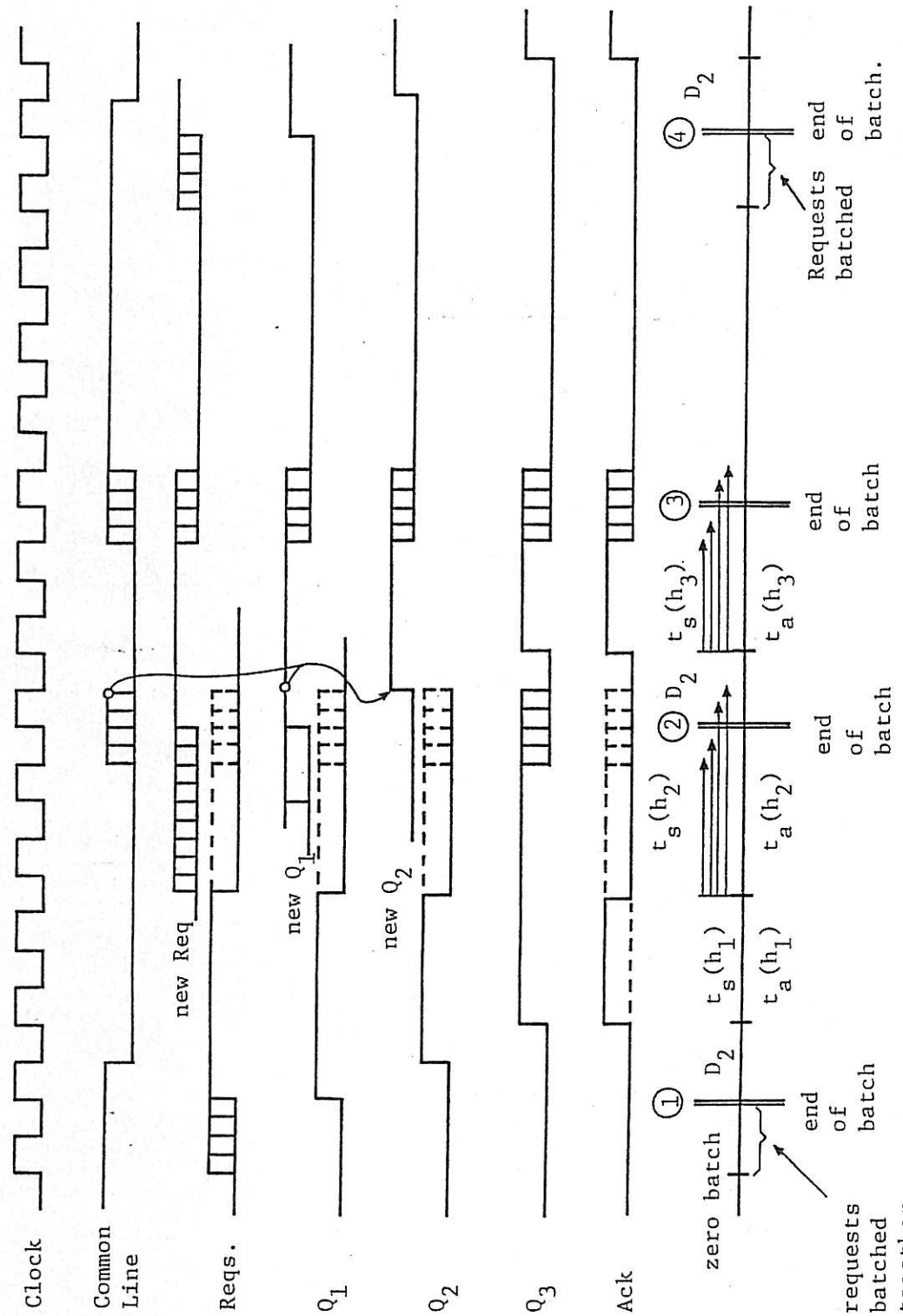


FIGURE 7.9 Timing of Direct Resetting Clocked Batching Arbiter.

7.7.1 Markov Property of the Direct Resetting Version

The clocked arbiters with no direct resetting and half direct resetting both are Markov processes when the state definition of Section 7.3.3 and exponential re-request distributions are assumed. This follows because all the information necessary to determine the probability of a transition from state j to state i is contained in the state j.

However, with the direct reset, a problem arises at the end of a batch. As shown in Figure 7.9, requester h_2 may finish accessing the resource after the latching point for the next batch (labelled (2) in Figure 7.9). The time remaining for h_2 before the latch point (3) is a function of the state (2)(3) *and* state (1)(2). The transition probabilities from state (2)(3) are not Markovian, because they depend on the previous *two* states.

A new state definition could be adopted that contained the previous two batch compositions. Transition probabilities would be

$$\text{Prob}\left[S(n),\ S(n-1)\ \middle|\ S(n-2),\ S(n-3)\right] \qquad (7.7)$$

This however, involves 2^{2k} states, where k is the number of requesters, in the Markov process, which is considerably more complex than the original model.

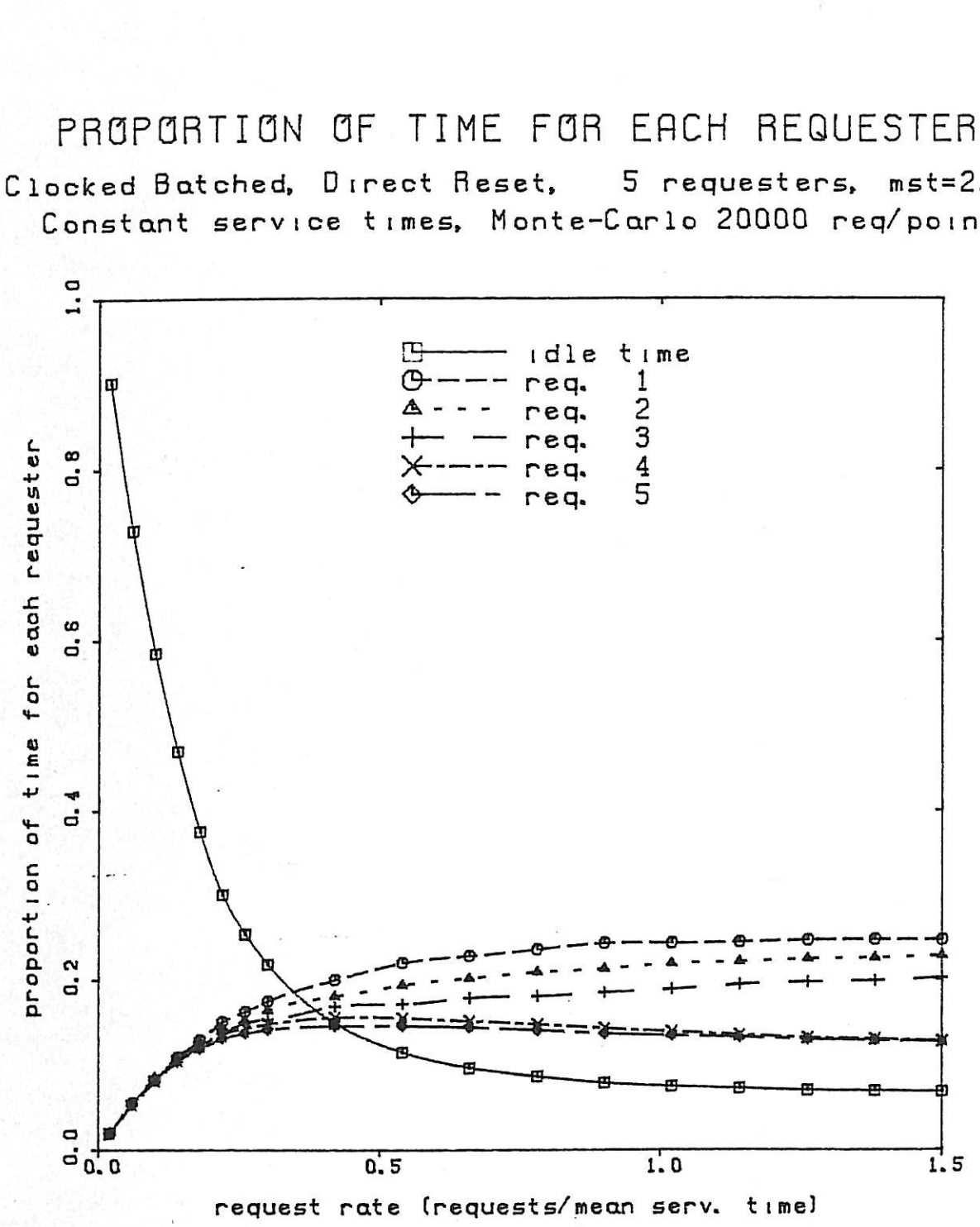
An approximation could be employed when the clock period is small compared with batch lengths. The approximation is to ignore the overlap altogether and assume the last requester in a batch finishes on the rising clock edge nearest to the actual finishing time. Thus t_a is defined as

$$t_a(h) \triangleq \begin{cases} t_s(h) & \text{, } h \text{ not serviced last in the batch.} \\ \left[\dfrac{\frac{1}{2} + \sum\limits_{h_1 \in \text{batch}} t_s(h_1)}{T_c}\right] T_c - \sum\limits_{\substack{h_1 \in \text{batch} \\ h_1 \neq h}} t_s(h_1) & \text{, } h \text{ not serviced last in the batch} \end{cases} \tag{7.8}$$

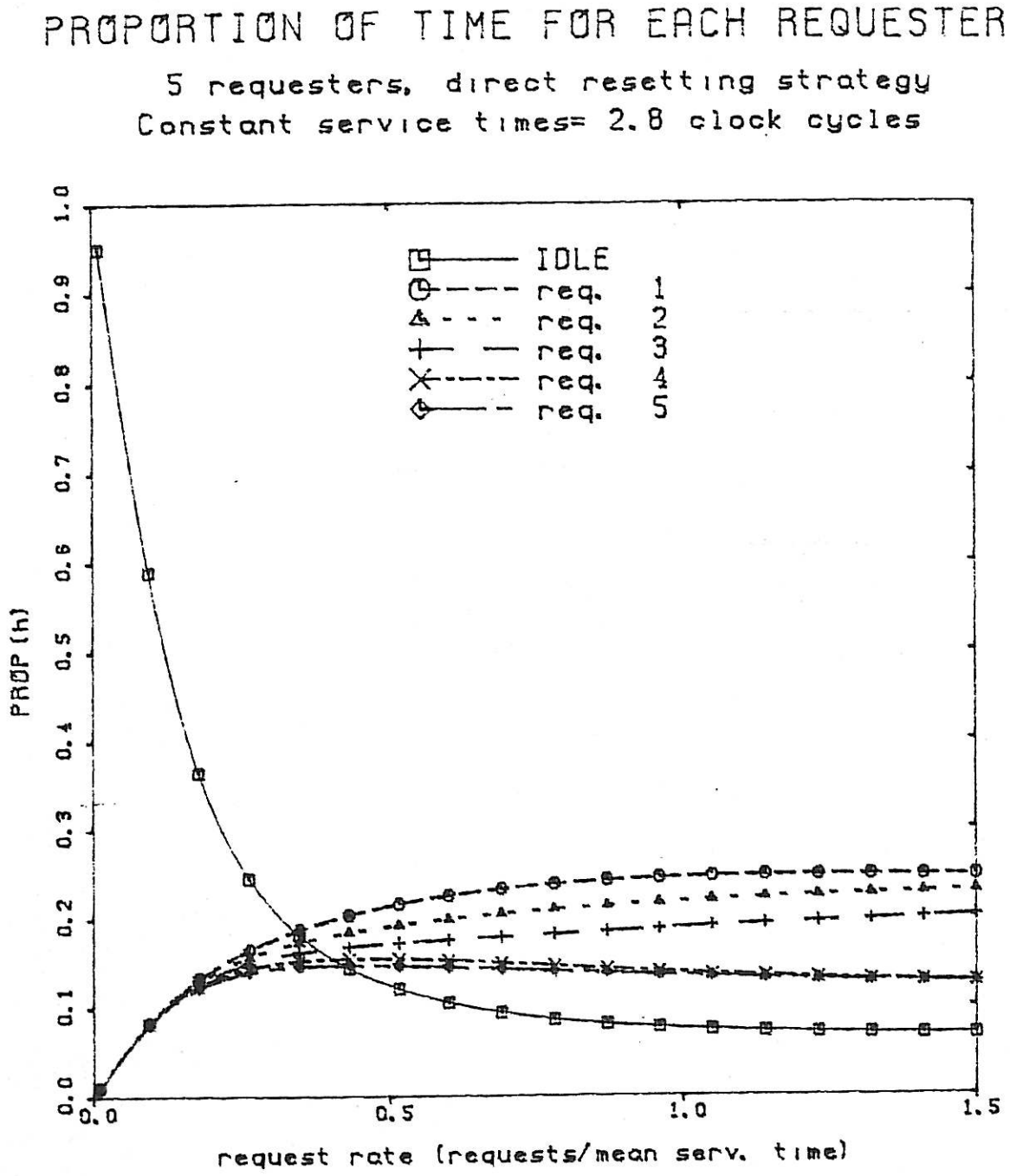
With this definition, the asynchronous model of Chapter 5 can be applied with $D_2 = T_c$, $D_3 = D_1 = 0$ and the zero to non zero transition probabilities modified as in Section 7.4.

7.7.2 Assessment of the Markov Approximation

The approximate Markov model for the direct resetting version described in Section 7.7.1 and 7.4 has been used to generate numerical results. Monte-Carlo simulations based on the complete timing model of Figure 7.9 have been performed to assess the Markov approximation. Graphs 7.1 and 7.2 show a direct comparison in results with a service time of 2.80 clock cycles. The graphs are virtually indistinguishable. A more sensitive comparison is shown in Table 7.1, where mean waiting times are considered. Slight differences of the order of 1 to 2% can be observed between the Markov and Monte-Carlo results. The approximation is concluded to be reasonably accurate.


PROPORTION OF TIME FOR EACH REQUESTER
Clocked Batched, Direct Reset, 5 requesters, mst=2.80
Constant service times, Monte-Carlo 20000 req/point
idle time
req. 1
req. 2
req. 3
req. 4
req. 5
proportion of time for each requester
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 7.1


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, direct resetting strategy
Constant service times= 2.8 clock cycles
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 7.2

TABLE 7.1 DIRECT RESETTING STRATEGY.

	MWT(1)	MWT(2)	MNW(3)	MWT(4)	MWT(5)
Approximate Markov Results	4.681	6.002	7.670	9.742	10.73
Monte-Carlo Run 1 (100,000 Services)	4.730	6.038	7.618	9.432	10.472
Monte-Carlo Run 2 (100,000 Services)	4.741	6.079	7.667	9.447	10.546
Monte-Carlo Run 3 (100,000 Services)	4.746	6.081	7.635	9.391	10.519
Monte-Carlo Run 4 (100,000 Services)	4.474	6.069	7.626	9.412	10.510

All times normalised by the service time.

Request rate = 0.5 requests/service time

Service time = 2.5 clock cycles.

7.8 COMPARISON OF EFFICIENCY AND SERVICE PERFORMANCE

In this section the service and efficiency performance of the three resetting strategies is compared. Due to similarities with numerical results presented in Chapter 5, features pertaining to the synchronous nature of the arbiters are highlighted in the results. These include the efficiency of the arbiters measured in terms of IDLE, and the proportion of time allocated to each requester. Firstly, the differences in modelling parameters are discussed.

The more obvious differences lie in the wasted time between servicing of requests during a batch. This can be seen by examining $t_a(h) - t_s(h)$: These results are shown in Table 7.2 and a graphical representation is shown in Figure 7.10.

EFFICIENCY AND SERVICE PERFORMANCE COMPARISON

Strategy	$t_a(h)-t_s(h)$	D_1
no resetting	$\frac{T_c}{2} \rightarrow \frac{3T_c}{2}$	$\frac{T_c}{2}$
half resetting	$0 \rightarrow T_c$	0
direct resetting	0	$\simeq 0$

Table 7.2

The best is the direct resetting strategy with no time wasted between services during the batch.

Another difference is the inter-batch time denoted D_1. This is wasted time also, and the different strategies are compared in Table 7.2. D_1 has a function which is less obvious. It provides the opportunity for

the last requester serviced in a batch to request before the end of the batch. This makes the arbiter slightly fairer under heavy loading conditions, numerical examples of which can be seen in Chapter 5.

Results of a Markov computer study are shown in Graphs 7.3, 7.4, 7.5, 7.6, 7.7 and 7.8. To aid the reader, Figure 7.10 shows the variations of t_a with t_s for no resetting and half resetting strategies.

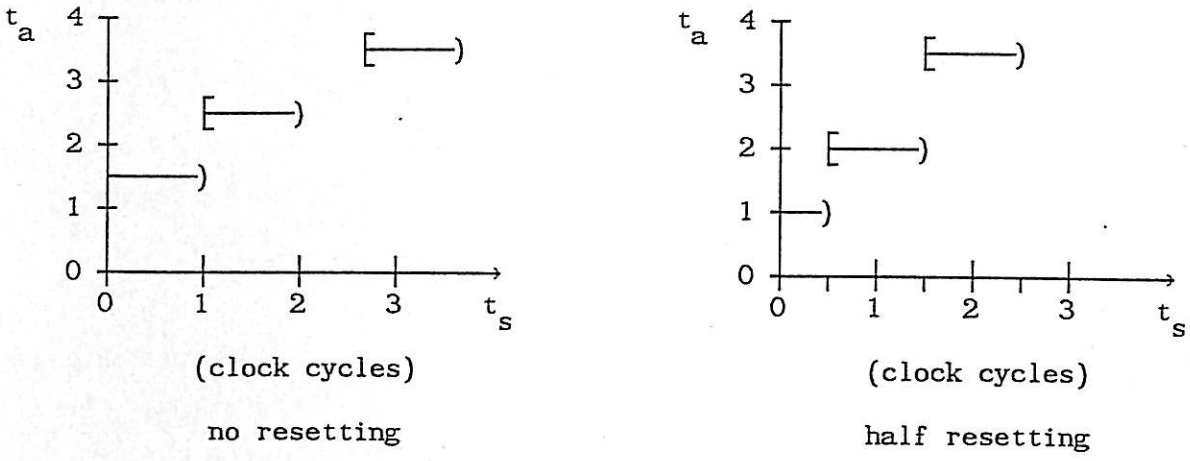
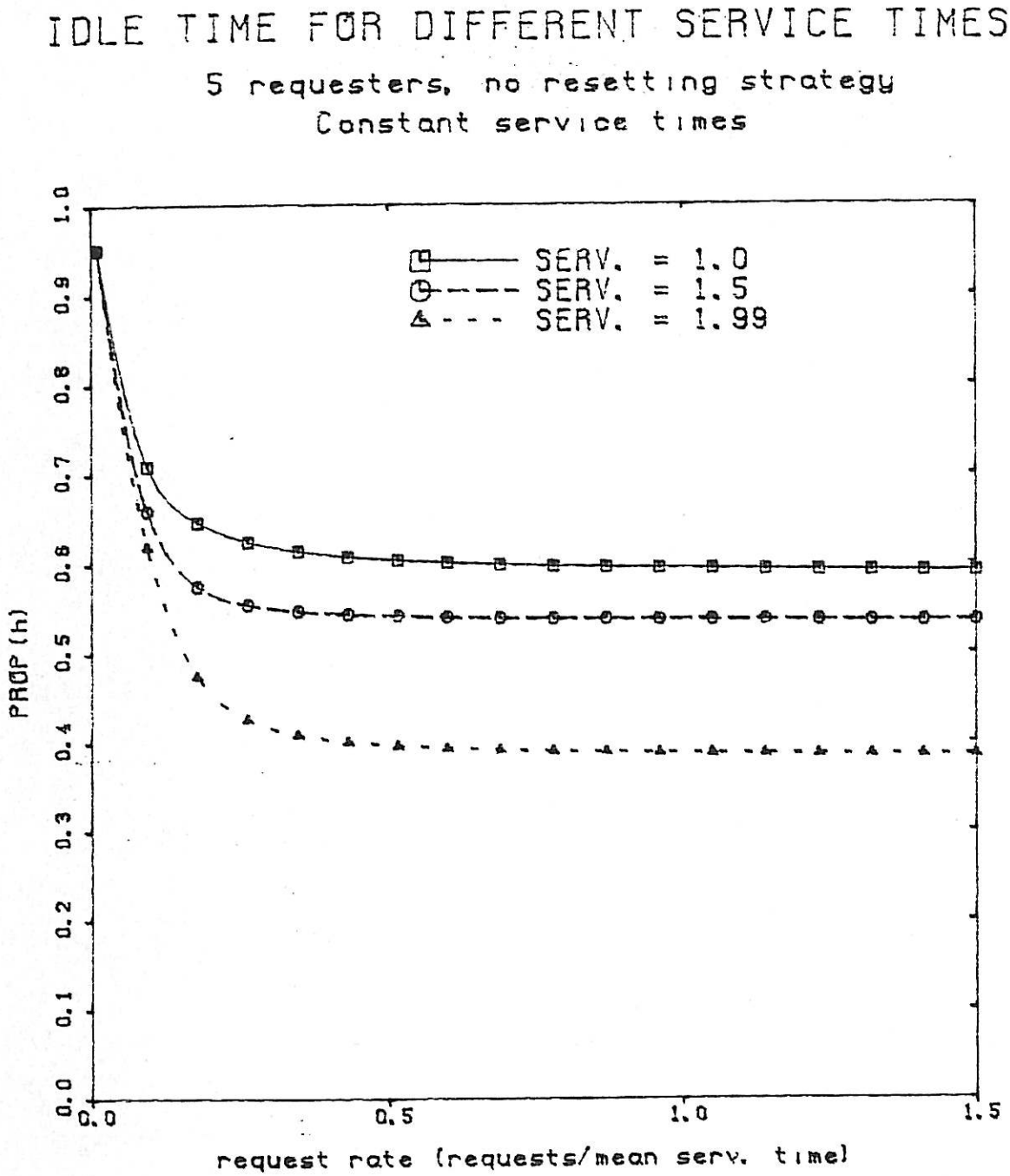


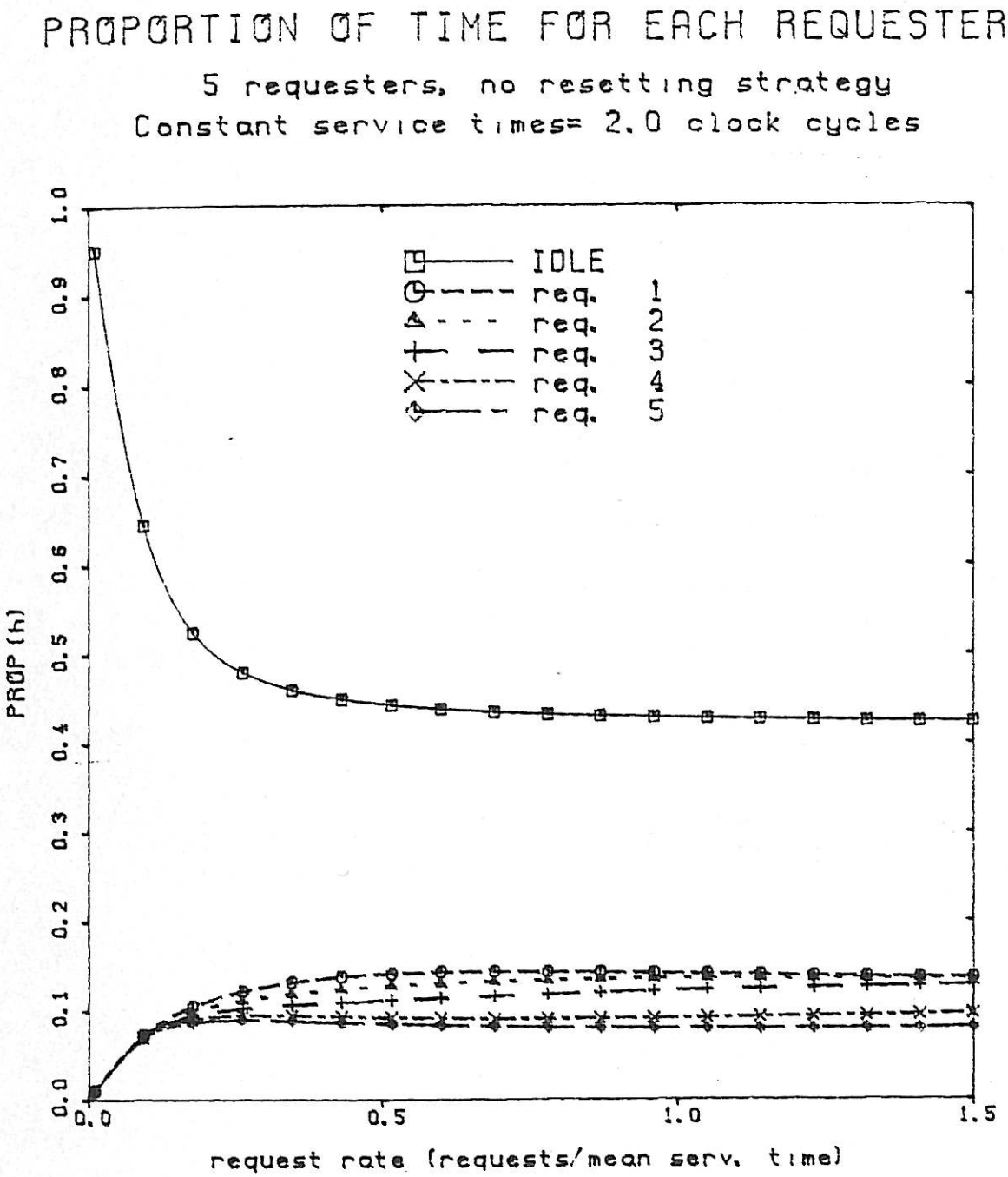
FIGURE 7.10 t_a Versus t_s.

For each strategy the IDLE time varies considerably as t_s varies due to the discretisation of the clock maintaining a constant t_a until t_s reaches the next clock period. As can be seen in Figure 7.10 for no resetting a threshold occurs at one clock cycle and for half resetting at 0.5 and 1.5 clock cycles. The effect of these thresholds is evident in the *worsening* of IDLE for t_s *increasing* from 1.99 to 2 in Graphs 7.3 and 7.4.

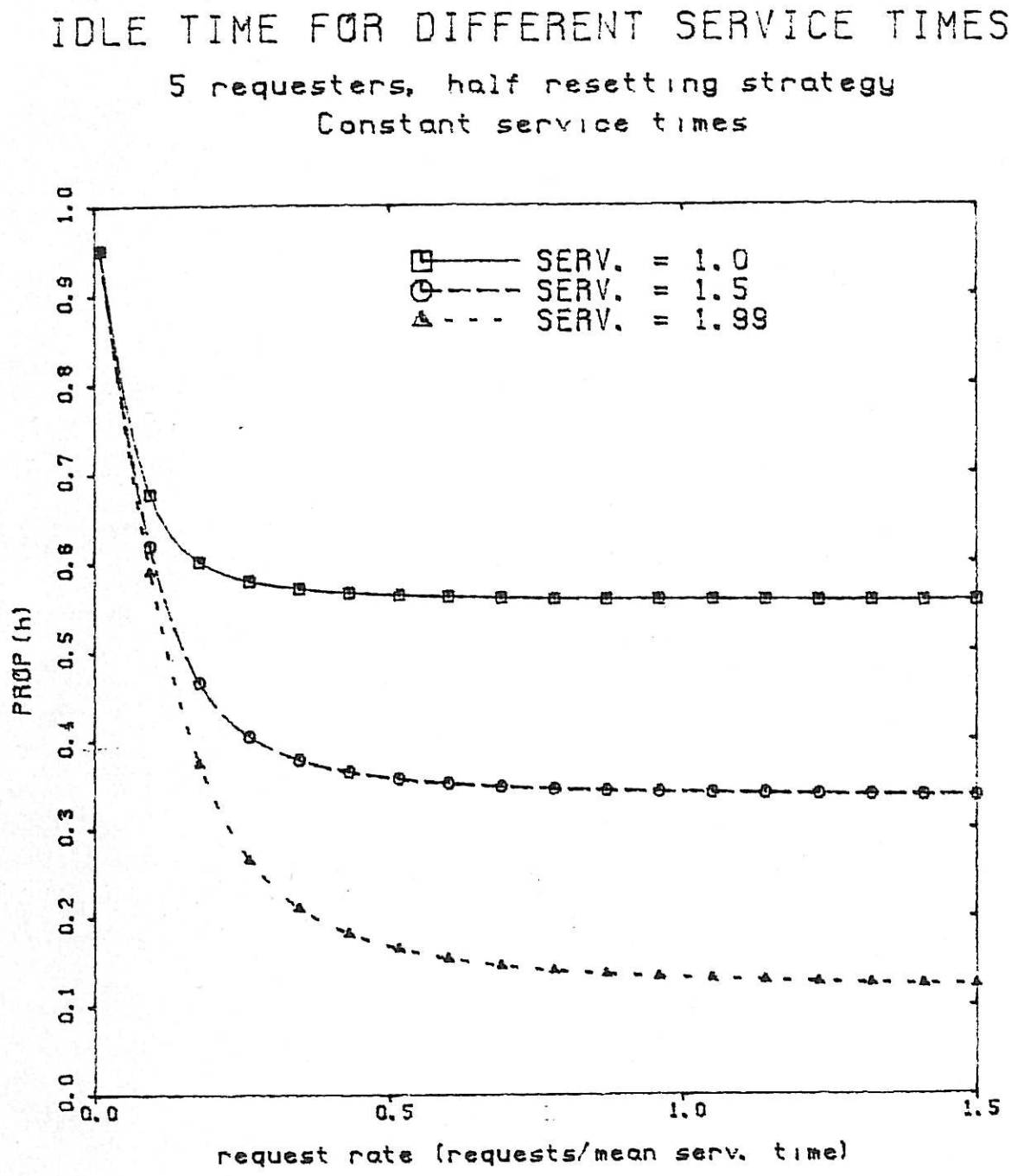
The conclusion is that the direct resetting strategy provides superior efficiency and utilisation performance over half and no resetting strategies.


IDLE TIME FØR DIFFERENT SERVICE TIMES
5 requesters, no resetting strategy
Constant service times
SERV. = 1.0
SERV. = 1.5
SERV. = 1.99
PRØP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

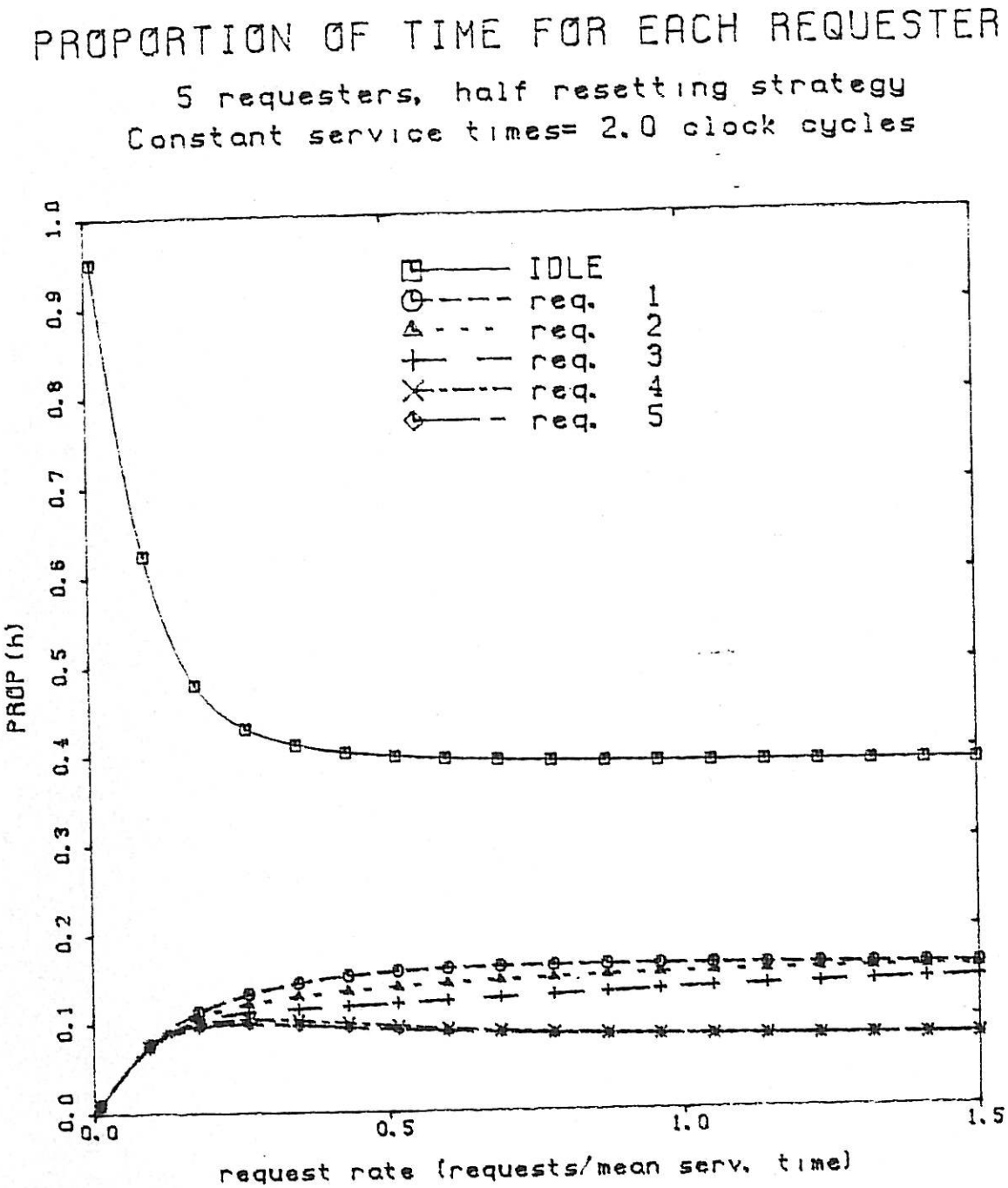
GRAPH 7.3.


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, no resetting strategy
Constant service times= 2.0 clock cycles
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

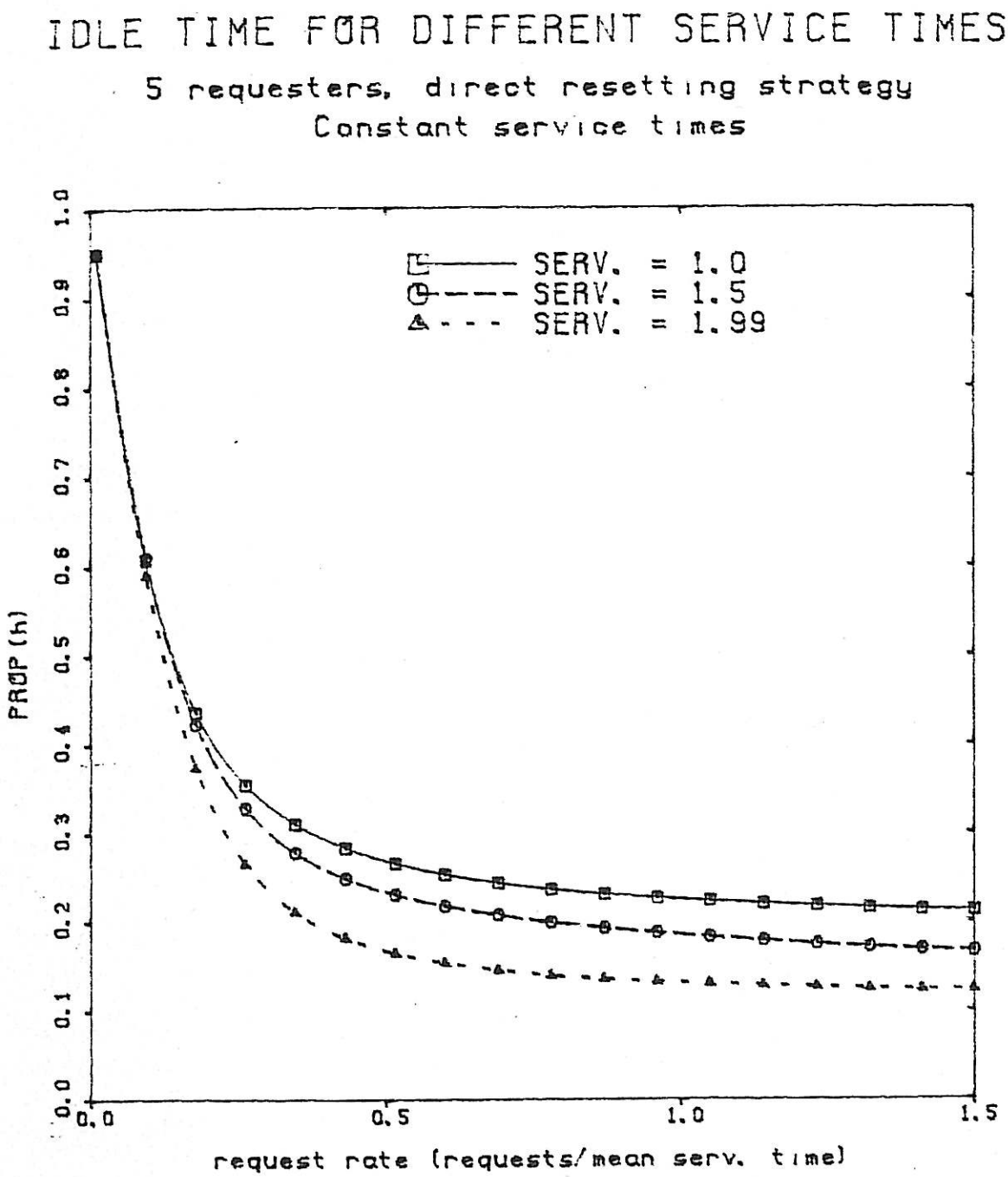
GRAPH 7.4.


IDLE TIME FØR DIFFERENT SERVICE TIMES
5 requesters, half resetting strategy
Constant service times
SERV. = 1.0
SERV. = 1.5
SERV. = 1.99
PRØP (h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

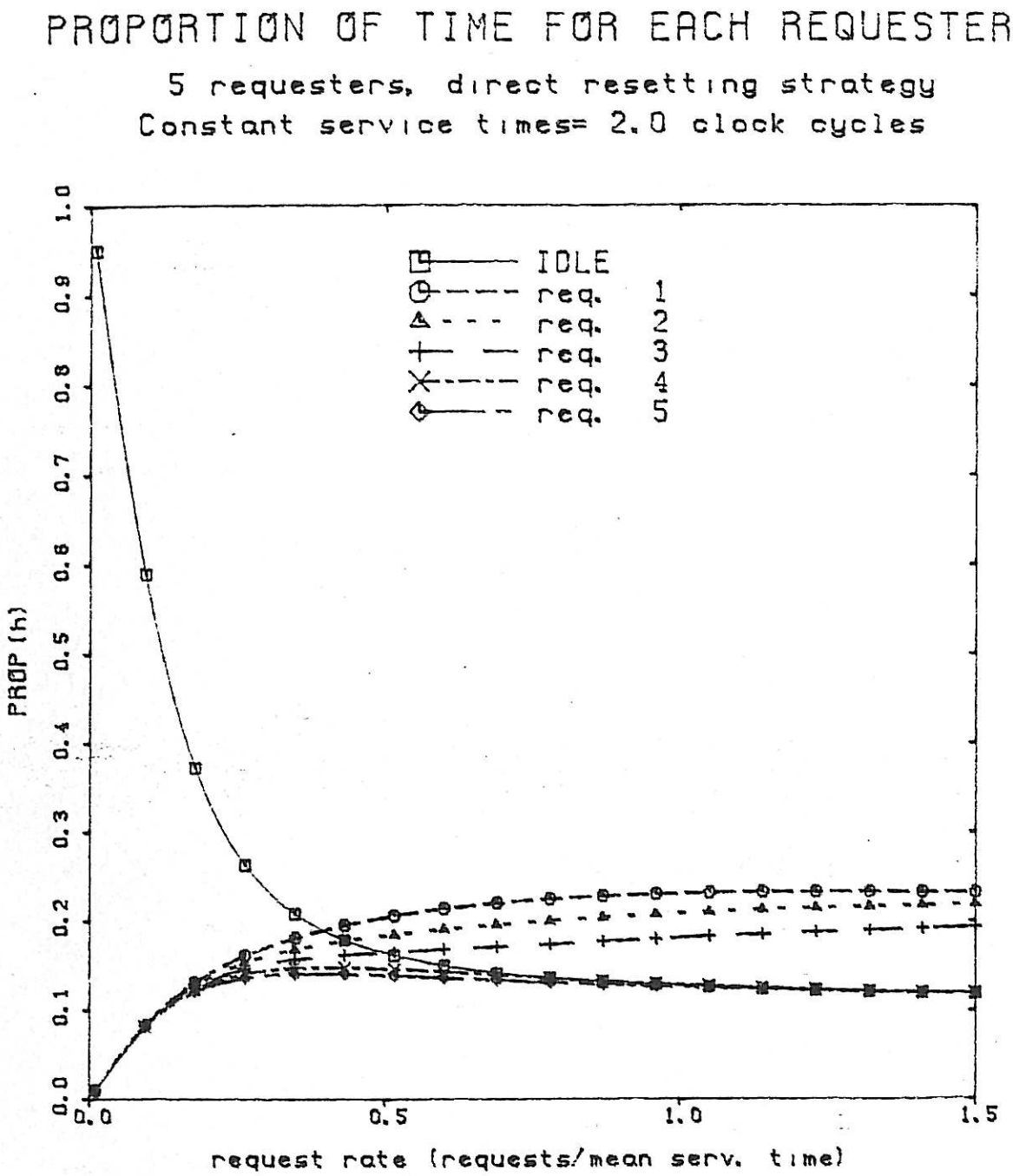
GRAPH 7.5.


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, half resetting strategy
Constant service times= 2.0 clock cycles
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 7.6.


IDLE TIME FØR DIFFERENT SERVICE TIMES
5 requesters, direct resetting strategy
Constant service times
SERV. = 1.0
SERV. = 1.5
SERV. = 1.99
PRØP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 7.7.


PROPORTION OF TIME FOR EACH REQUESTER
5 requesters, direct resetting strategy
Constant service times= 2.0 clock cycles
IDLE
req. 1
req. 2
req. 3
req. 4
req. 5
PROP(h)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 7.8.

7.9 METASTABLE BEHAVIOUR OF THE CLOCKED BATCHED ARBITERS

As mentioned in Chapter 3, metastable behaviour of synchronising elements or flip-flops within an arbiter can lead to failure of an arbiter. Failure may be seen as an Ack output undefined for a prolonged time or two Ack outputs interpreted as asserted simultaneously. When request timing is independent of internal arbiter timing or other request timing (i.e. asynchronous requests) the arbiter can be marginally triggered. This corresponds to either a request occuring whilst the particular request line is in the process of being synchronised in a clocked arbiter. As discussed in Chapter 3, the problem of metastable behaviour in arbiters is unavoidable, however an established technique for ensuring a small probability of failure is to design sufficient delay in the circuit to mask metastable behaviour with settling times less than the delay. Due to the exponential distribution of metastable settling times, dramatic improvements in failure rates can be achieved with small changes in delay. Similarly, within a single circuit metastable failure modes with differing amounts of delay for metastable settling may have very different probabilities of causing a failure. For example, if a full clock period is allowed for settling after synchronising rising edges and half a clock period for synchronising falling edges, then it would be reasonable to assume the falling edge synchronisation failure probability dominates the rising edge synchronisation failure probability. Critically associated with each synchronisation event is the allowed settling time. Only the shortest settling time modes of a circuit need be considered in a failure analysis.

Because the rising edge request synchronisation is identical in all forms of the clocked batched arbiter, it is considered first.

7.9.1 Failures due to Synchronisation of Rising Edges of Requests.

Only rising edges requests will occur at or near the end of batch need to be considered. All other synchronisation of rising edges occur during a batch and are allowed at least an extra clock cycle to settle before being unmasked by the common line on the J input to FF2 in Figure 7.4 at the end of a batch.

The timing diagram shown in Figure 7.11 illustrates the sequence of events that may occur at the end of batch synchronisation. A request is asserted just when requests are being sampled for the last time at the end of batch. This causes FF1 to enter a metastable state which lasts at least $T_c/2$. Q1 forms the input to FF2, and FF2 enters a metastable state lasting at least $T_c/2$ caused by sampling the undefined Q1. Provided the daisy in signal is asserted for the particular module (i.e. it is higher in the daisy chain than all others with latched requests), the daisy out and Ack outputs are undefined causing a lower Ack to be undefined. The resulting scenario is that two Ack signals are undefined and can be interpreted as asserted, constituting failure of the arbiter.

The sequence of events is conditional on a metastable state lasting $T_c/2$ and transferring to another flip-flop and lasting a further $T_c/2$. The relative probability of this with respect to a single flip-flop staying in a metastable state for a period of T_c is still a matter of research. It is assumed here that the two are of the same order of magnitude.

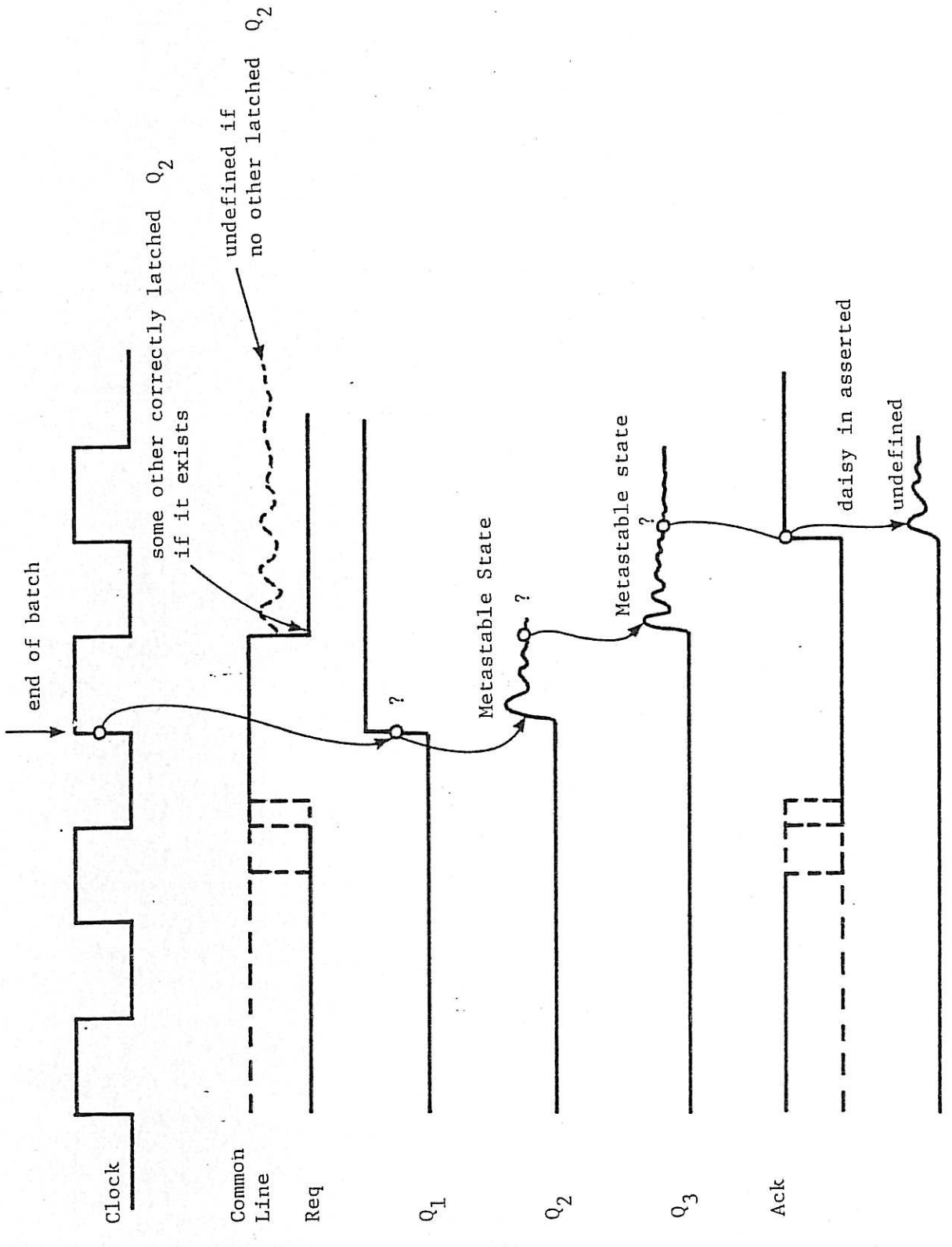


FIGURE 7.11 Metastable Failure on Rising Request Synchronisation.

7.9.2 Falling Request Edge Failure of No Resetting Clocked Batched Arbiter

As discussed previously, falling request edges are synchronised to the clock by the no resetting clocked batched arbiter. The timing diagram shown in Figure 7.12 illustrates a synchronisation failure when a request is dropped. FF1 enters a metastable state which lasts at least $T_c/2$ and causes FF2 to enter a metastable state. The Ack corresponding to the dropping Req becomes undefined, and if a lower priority FF1 is set (i.e. request latched), the corresponding Ack is undefined.

Note that half the settling time is allowed for dropping request synchronisation failure compared with the rising edge request synchronisation failure. Thus, the failure probability is dominated by the falling request synchronisation. During a batch of m services, $m-1$ synchronisation events occur that can result in double Ack assertions, and one synchronisation event that can result in a single undefined Ack signal. Note that other failures can occur where the settling time must exceed $T_c/2$ to cause failure but these are ignored.

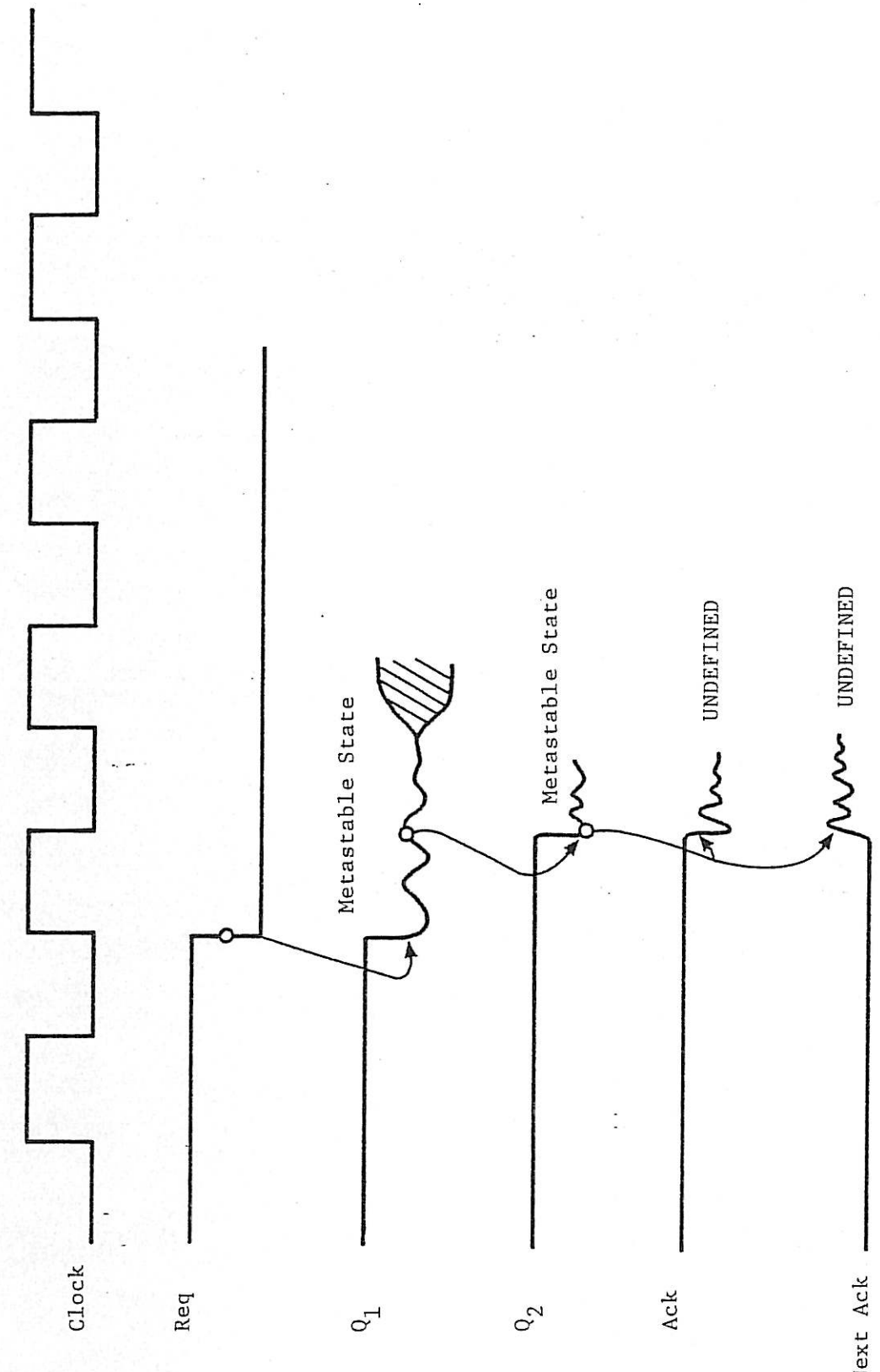


FIGURE 7.12 Falling Request Edge Failure of No Direct Resetting Clocked Batching Arbiter.

7.9.3 Falling Request Edge Failure of Half Resetting Clocked Batched Arbiter

During a batch, the half resetting clocked batched arbiter synchronises falling requests to the clock. The situation of Req dropping near the sampling clock edge is illustrated in the timing diagram of Figure 7.13. As shown, FF1 enters a metastable state which directly induces FF2 to enter a metastable state via the FF2 reset line. The probability of an undefined reset signal inducing metastable behaviour is not known, but certainly cannot be dismissed. However, the lack of any allowance for settling time for FF1 could be catastrophic for synchronisation reliability. This circuit is considered to be a very poor synchroniser.

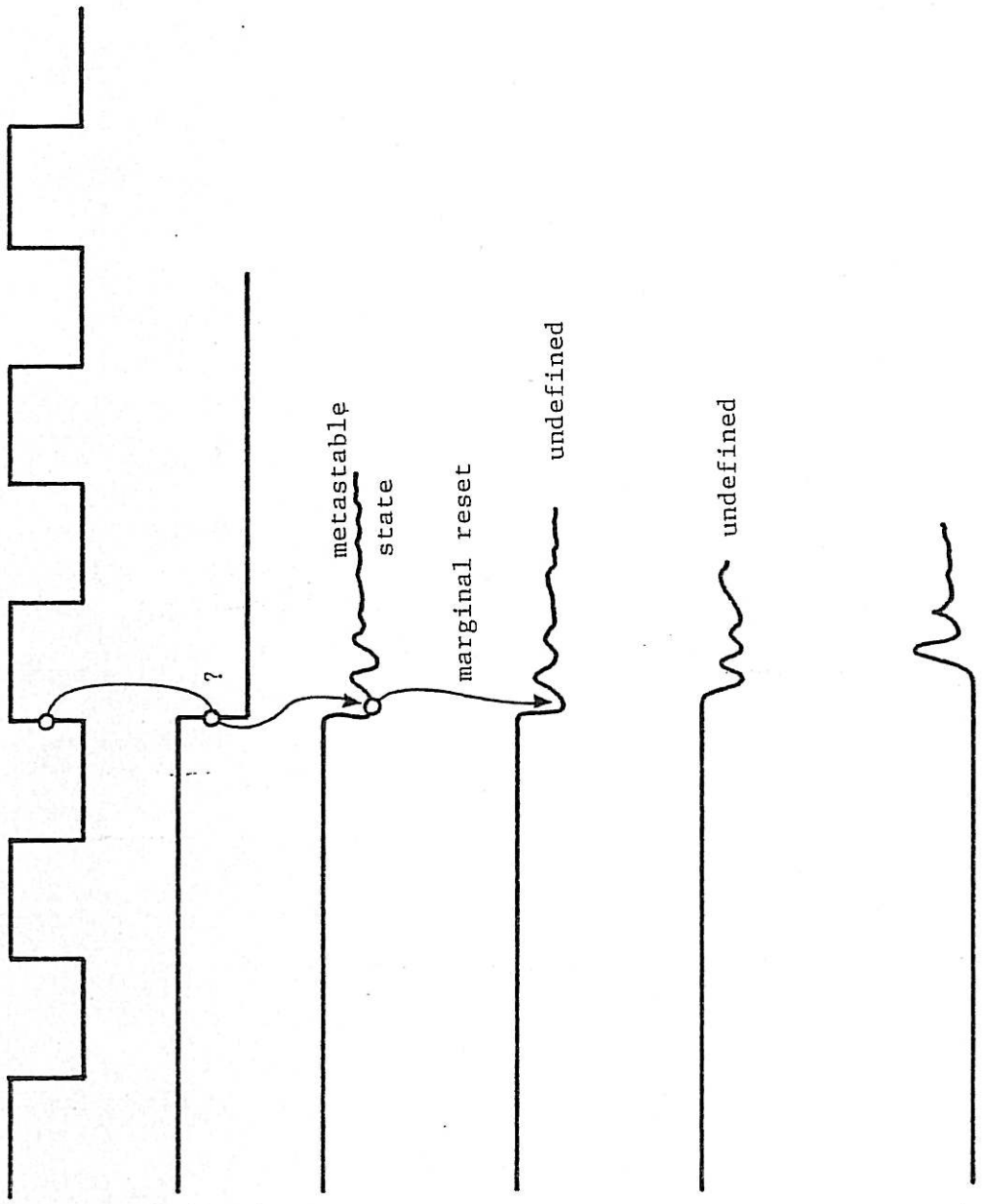


FIGURE 7.13 Falling Request Edge Failure of Half Resetting Clocked Batching Arbiter.

7.9.4 Falling Request Edge Failure of Direct Resetting Clocked Batched Arbiter

The direct resetting circuit has the apparent advantage of *not* synchronising falling request edges to the clock during a batch. Thus, no metastable failures can occur when a requester, not last in a batch, drops its request line.

The last requester in a batch can cause metastable failure if Req is dropped so that the common line goes high on a falling clock edge. As shown in Figure 7.14, the common line changing on a falling clock edge marginally triggers FF2 of a module latching a request at the start of the next batch. The settling time available for FF2 is $T_c/2$, after which the Ack and daisy outputs are affected, leading to failure of the arbiter. These events are all conditional on Daisy In being asserted for that particular module. The failure described cannot occur if a higher priority module successfully latches its request, thus blocking the daisy chain. The probability of failure is the sum of the probabilities of the following mutually exclusive events:-

1. The highest priority module with set FF1 fails.
2. The second highest priority module with set FF1 fails and all higher modules have a reset FF2.

.

.

m. The m^{th} module with set FF1 fails and all higher priority modules have a reset FF2.

Where m is the number of modules with FF1 set at the time of marginal triggering.

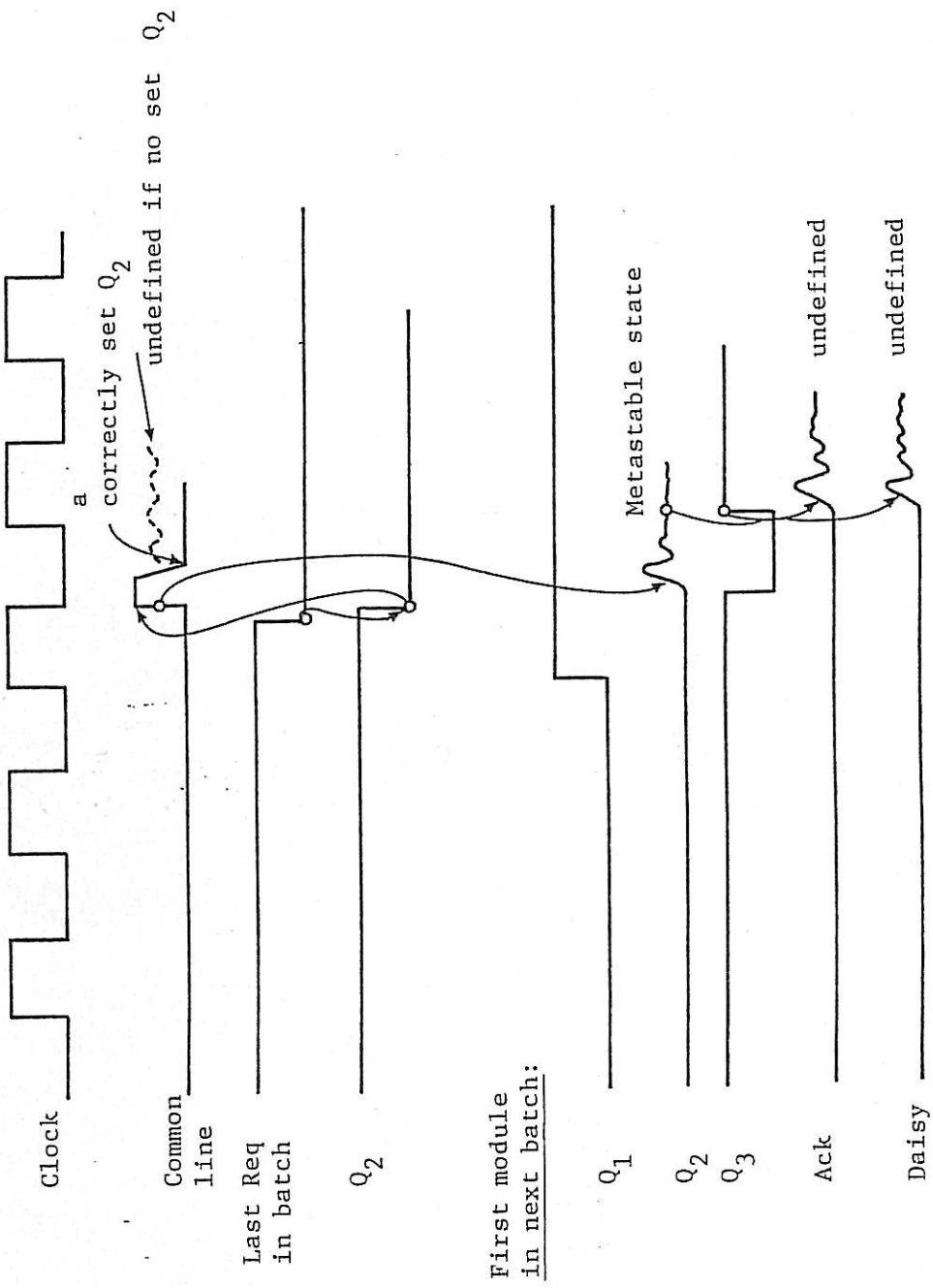


FIGURE 7.14 Failure of Direct Resetting Clocked Batching Arbiter due to Dropping Req.

Assume that the probability of a marginally triggered flip-flop not settling after $T_c/2$ and causing a failure is p_f and the probability of a marginally triggered flip-flop settling to a reset state before $T_c/2$ is $\frac{1}{2}$. (It is assumed that p_f is much smaller than 1 and so if there is equal probability of a marginally triggered flip-flop resolving to either state, the reset state will have probability $\frac{1-p_f}{2} \approx \frac{1}{2}$). The probability of event ℓ occurring is the $(\frac{1}{2})^{\ell-1} p_f$. Thus

$$\text{prop (failure at end of batch} \mid \text{common line changes near falling clock edge)}$$

$$= \sum_{\ell=1}^{m} (\tfrac{1}{2})^{\ell-1} p_f < 2p_f \qquad (7.9)$$

This failure probability should be compared with that of the no resetting strategy which is approximately equivalent to $m\ p_f$. This is providing the two failure processes are assumed to have identical probabilities which is not unreasonable considering both involve the same settling period of $T_c/2$. The above analysis suggests the direct resetting strategy is slightly superior in metastable reliability, under heavy request loading where batch lengths (m) are expected to be close to the total number of requesters. Under light request loading batch lengths equal to one would be expected, in which case both strategies give similar results.

7.10 COMPARISON OF THE METASTABLE RELIABILITY OF THE RESETTING STRATEGIES

The dominant failure mode was found to be the falling edge synchronisation of requests in all cases, because of the reduced metastable settling time offered by the arbitration circuitry. A particular poor circuit is the half resetting strategy which offers no settling time at all for dropping request synchronisation and thus should be avoided.

The other two resetting strategies:- no resetting and direct resetting performed similarly, the direct resetting slightly superior under heavy request loading and the assumptions outlined in Section 7.9.4. Both allowed equivalent setting times of $T_c/2$.

It is interesting to note that the asynchronous decentralised batched arbiter described and analysed in Chapters 4 and 5 is never subject to metastable failure on falling requests but only rising request edges which occur at the start of a batch. An isolated request during a zero batch *cannot* cause metastable failure in the asynchronous circuit but can in the clocked version.

7.11 GENERAL CONCLUSIONS

The clocked arbiters studied in this chapter differ from the asynchronous arbiter designs presented and analysed in the thesis in the following ways:

(i) The design of clocked arbiters is usually simpler and more easily verified. This may result in a more error free design process.

(ii) The clocked designs produce longer delays in acknowledging requests since the clock period must be designed for worst case delays in the circuit over all possible clock period contingencies. Thus, the idle time of clocked arbiters tends to be higher.

(iii) Requests must be synchronised to the clock on both the assertion of a request *and* the dropping of a request. The latter is absent from asynchronous circuits. The number of synchronisation events where metastable failure of the arbiter can result is greater in clocked designs since both request edges are synchronised.

(iv) In asynchronous designs, requests are synchronised with respect to each other, and k autonomous timing references exist, one for each requester. Clocked designs introduce an *additional* autonomous reference, the clock. Thus, k+1 timing references exist which must be synchronised with respect to one another within the clocked circuit. For example, when an asynchronous arbiter is idling, a single request cannot cause metastable failure. In an idling clocked arbiter circuit, a single request *may* cause metastable failure when the request is synchronised to the clock.

In conclusion, whilst the clocked design methodology is simpler than asynchronous design, a clocked design tends to suffer from a poorer service and metastability performance.

CHAPTER 8

PERFORMANCE COMPARISONS OF ARBITRATION DISCIPLINES

8.1 INTRODUCTION

Chapters 5, 6 and 7 presented results on the performance of batched fixed priority and non batched fixed priority arbiters. In this chapter, these results are extended to encompass additional performance measures and *all* the arbitration disciplines introduced in Chapter 2, with the aim of comparing the performance of each discipline. The Markov analysis that would be necessary to generate these results is too complex and difficult in most cases and, consequently, Monte-Carlo techniques are employed in this chapter. As discussed in Chapters 5, 6 and 7, the Markov analysis yields practically identical results to the Monte-Carlo simulations when sufficient service iterations are performed.

The disciplines examined in this chapter are all assumed to conform to models of asynchronous arbiters developed in Chapter 4. Clocked arbiters are not considered in this chapter due to their poorer performance and also greater complexity in modelling, as indicated in Chapter 7.

The performance measures examined in this chapter include the standard deviation of waiting times and the metastable failure rate. The Monte-Carlo simulations employed to obtain the metastable failure rate are based on analytical modelling and analysis techniques employed in Chapters 3, 5 and 6. The results presented in this chapter are new in that no previous study of arbitration disciplines has incorporated:

(i) non zero inter-service and inter-batch delays;

(ii) batched disciplines;

(iii) metastable reliability performance.

Some of the results of this chapter appear in [5].

The chapter is structured as follows : In Section 8.2, service performance results are presented together with a brief description of the Monte-Carlo simulation program. In particular, batched and non batched disciplines are examined. In Section 8.3, the techniques for evaluating the metastable reliability performance of disciplines are developed. Also, the results obtained from Monte-Carlo simulations based on these techniques are discussed and the metastable reliability performance of each discipline is compared in Section 8.3. Results incorporate the effects of inter-service and inter-batch delays likely to be present in practical circuits and application. Finally, the chapter concludes in Section 8.4 by summarising the performance comparisons and merits of various disciplines.

8.2 SERVICE PERFORMANCE

In this section, the service performance of the following arbitration disciplines is examined : first come first served (FCFS), batched fixed priority, fixed priority, next robin, batched next robin, least recently used (LRU), batched LRU, batched forward robin and batched reverse robin (both these non derived batched disciplines are discussed in Section 2.4.4) and random priority (a non batched dynamic priority discipline where highest priority is randomly and uniformly assigned and lower priorities following modulo k). The disciplines fall naturally into the two groups of batched and non batched. The timing model for the batched fixed priority arbiter described in Section 4.3.2 is generalised to *any* batched discipline by maintaining the definitions of D_1, D_2 and D_3 and varying the order of servicing within a batch to follow the

required discipline. Similarly, the timing model of the non batched fixed priority arbiter is extended to incorporate any non batched discipline while maintaining the definition of D_1 and D_2 described in Section 4.4.2.

The Monte-Carlo simulations presented model the time durations D_1, D_2 and D_3 (in batched disciplines) and arbitration disciplines in software. Re-request times are generated using a standard uniform random number generation function which is processed to obtain exponentially distributed times of the required mean. Due to the relative insensivity of the results to the service time distribution, only constant service times are considered in the results.

8.2.1 Proportion of Time and Mean Waiting Time Results

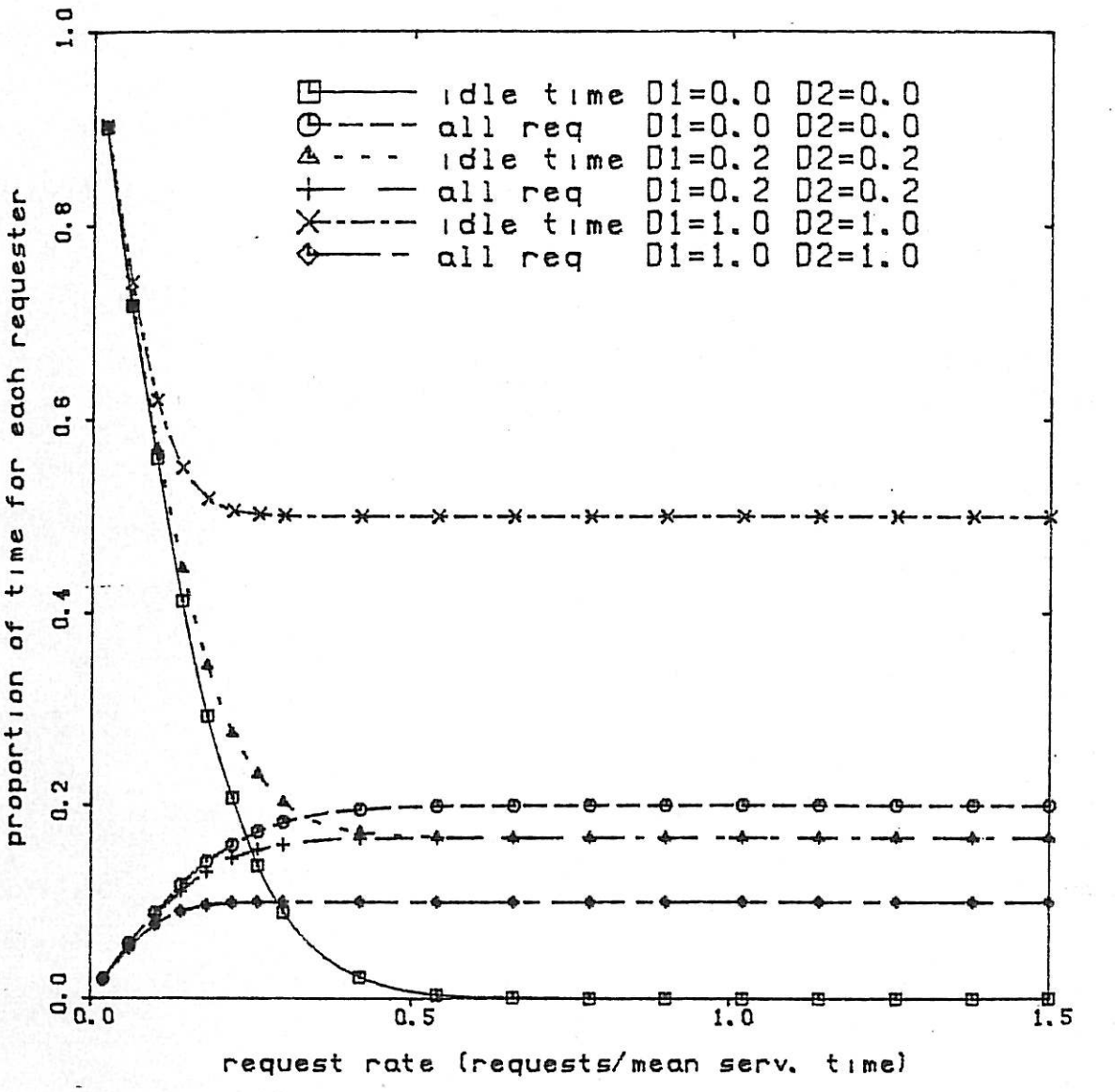
Graph 8.1 shows the proportion of time allocated to a requester and IDLE for FCFS under differing inter-service times. Graph 8.3 applies to all non batched disciplines which are symmetric to all requesters, that is, all non batched disciplines considered in this chapter except fixed priority.

The results of Graph 8.4 show the time efficiency of the arbiter when D_1 and D_2 are varied. For $D_1 = D_2 = 1.00$ less than half the available time is spent servicing requests. Graph 8.1 also gives an indication as to the request loading levels necessary to saturate the arbiter on a "throughput" basis.

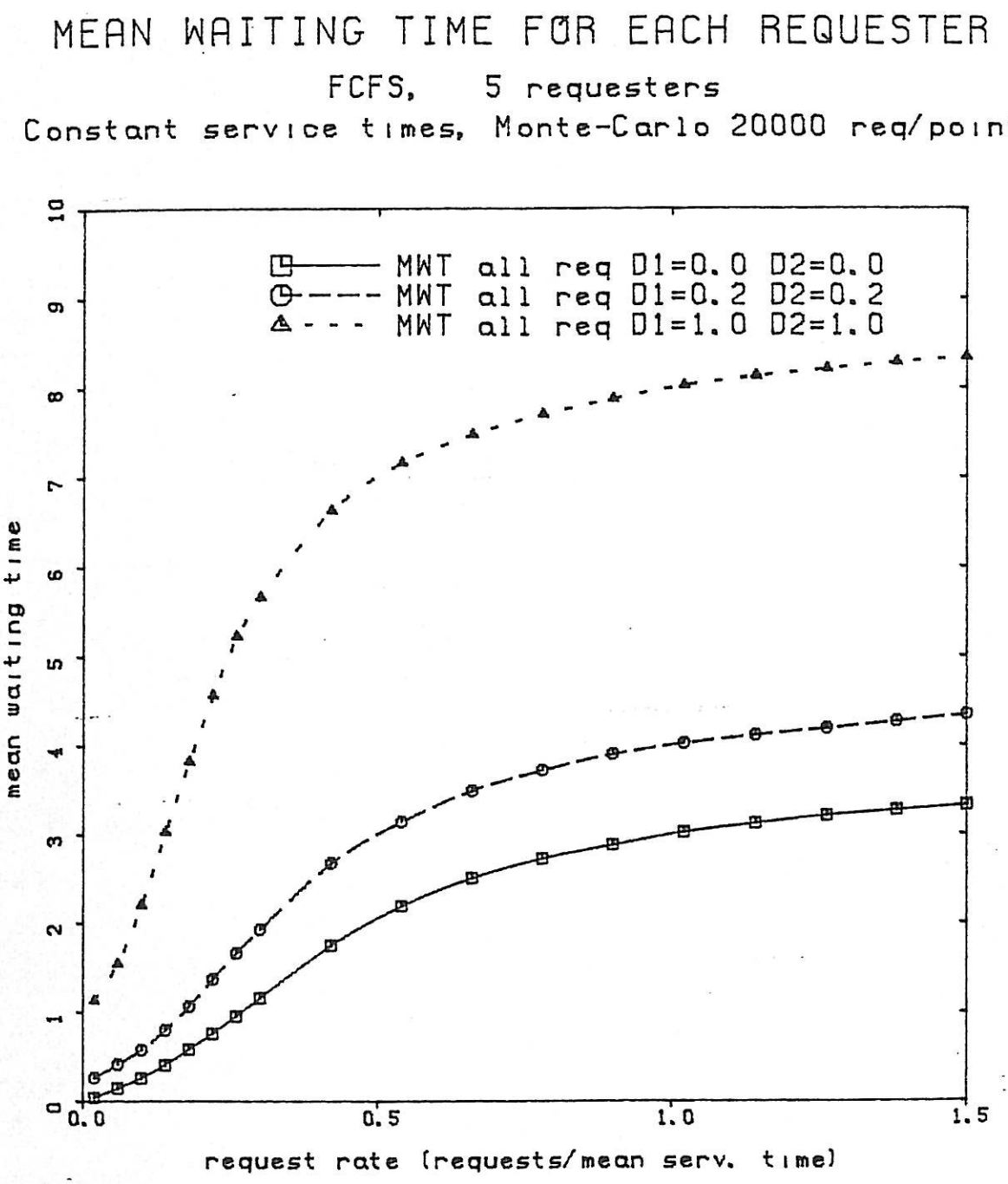
Graph 8.2 shows the corresponding mean waiting time for any requester for all symmetric non batched disciplines. Significant differences in performance can be observed for changes in the non ideal modelling parameters D_1 and D_2. Saturation in waiting times occurs at larger request rates than is the case for proportion of time. For

example, MWT starts levelling off when $D_1 = D_2 = 1$ at a request rate of approximately 0.5 requests per service time compared with a request rate of 0.2 in the proportion of time and IDLE results of Graph 8.1.

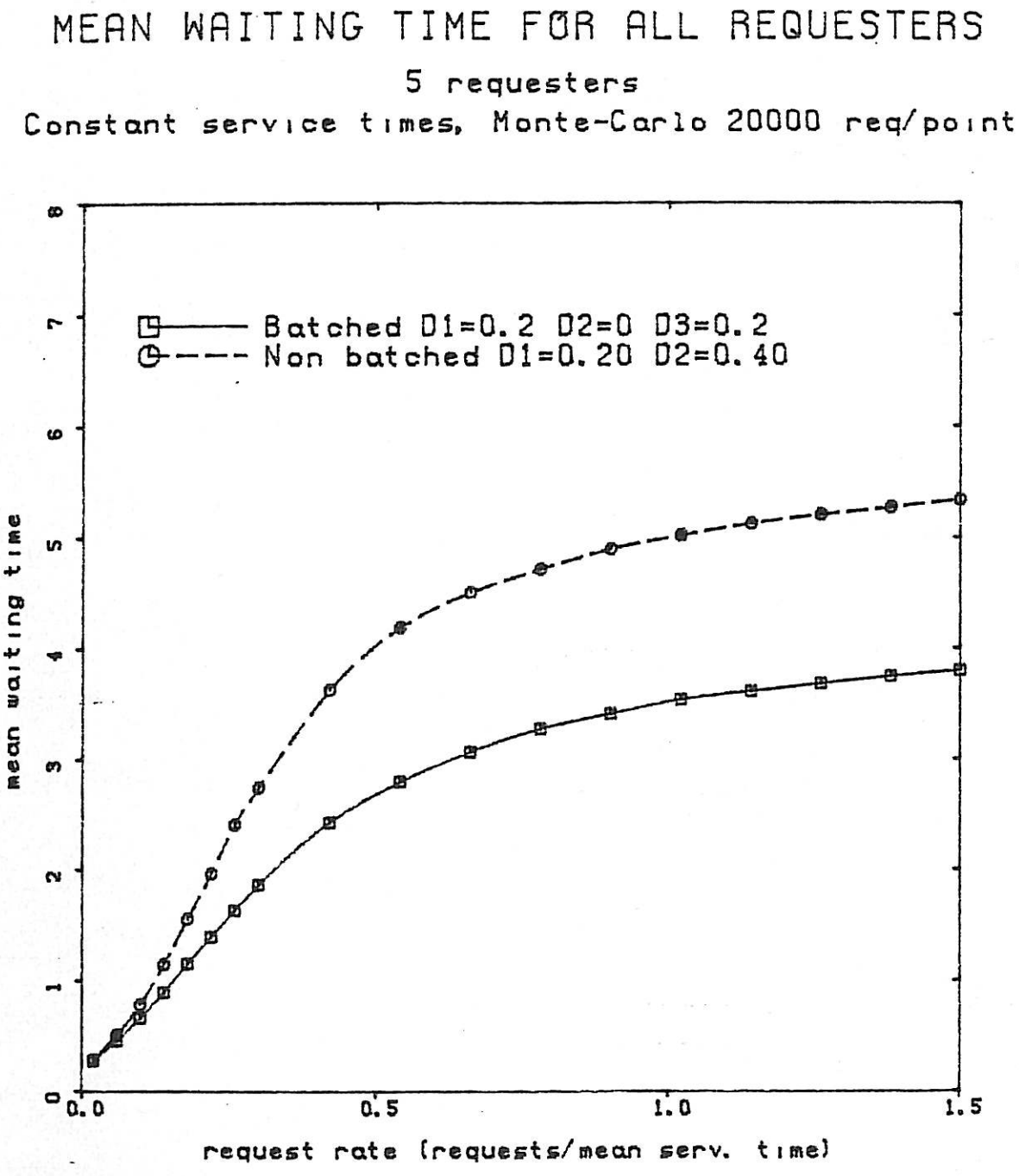
Graph 8.3 provides a comparison between batched and non batched arbiters implemented in logic with similar delay characteristics. Assuming $D_1 + D_3$ of the batched arbiter corresponds to approximately D_2 of the non batched arbiter, and D_3 of the batched arbiter corresponds to D_1 of the non batched arbiter, then Graph 8.3 provides a realistic comparison. These assumptions on D_1, D_2 and D_3 between batched and non batched arbiters can be seen to approximately hold for the daisy chained fixed priority arbiters discussed in Chapters 2 and 4. The reason non batched arbiters result in extra waiting time is attributable to "economy of scale" in servicing requests in batches of length greater than one. Very little delay occurs between servicing within a batch as apposed to consecutive non batched services.


PROPORTION OF TIME FOR EACH REQUESTER
FCFS, 5 requesters
Constant service times, Monte-Carlo 20000 req/point
idle time D1=0.0 D2=0.0
all req D1=0.0 D2=0.0
idle time D1=0.2 D2=0.2
all req D1=0.2 D2=0.2
idle time D1=1.0 D2=1.0
all req D1=1.0 D2=1.0
proportion of time for each requester
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.1


MEAN WAITING TIME FOR EACH REQUESTER
FCFS, 5 requesters
Constant service times, Monte-Carlo 20000 req/point
MWT all req D1=0.0 D2=0.0
MWT all req D1=0.2 D2=0.2
MWT all req D1=1.0 D2=1.0
mean waiting time
0
1
2
3
4
5
6
7
8
9
10
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.2


MEAN WAITING TIME FOR ALL REQUESTERS
5 requesters
Constant service times, Monte-Carlo 20000 req/point
Batched D1=0.2 D2=0 D3=0.2
Non batched D1=0.20 D2=0.40
mean waiting time
0
1
2
3
4
5
6
7
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.3

8.2.2 Standard Deviation of Waiting Time Results

As discussed in Chapter 2, if one considers the spread of waiting times to indicate a level of "fairness", the STDW can be employed as a measure of "fairness". Symmetric disciplines give identical proportion of time of mean waiting times to each requester but their STDW differs. The standard deviation is difficult to compute analytically and so Monte-Carlo techniques are employed.

Graphs 8.4 and 8.5 show STDW(h) for each requester h in batched *fixed priority* arbiters. It is interesting to observe that STDW is greater than STDW(h) for all h when the request rate exceeds 0.6 in Graph 8.4. This occurs because the arbiter begins to settle into the alternating pattern of servicing the batch 1234 followed by 1235, causing individual waiting times to be reasonably predictable and yet still a large variation between different requesters' waiting times exists.

Also, of interest are the cross overs in STDW(4) and STDW(5) at $\lambda = 0.6$, and STDW(1) and STDW(2) at $\lambda = 1.0$. This is a second order effect not explained simply. At request rates below 0.6, requester 5 can wait from zero to seven service times depending on the set of pending requests at the time. As the request rate increases, the variability decreases as the event of no requests pending becomes unlikely. The variability for requester 4 decreases less rapidly with the request rate, because the influence of requester 5's request pending is not great compared with vice versa. Requester 1 incurs greater variability in waiting times as the batch size varies with time, while requester 2 is in addition concerned with whether request 1 is pending which becomes almost a certainty above 1 request/service time.

In Graph 8.2, the non zero inter-batch times cause a greater variance in waiting times due to requesters 4 and 5 sometimes requesting

within the inter-batch times and considerably reducing their waiting times. This added variance is reflected in all STDW(h) results.

Graphs 8.6 and 8.7 show STDW(h) for the fixed priority arbiter. The results strongly reflect the priority structure of the arbiter and are similar to MWT(h) results presented in Chapter 6, with mean and standard deviation strongly coinciding.

As has been discussed in Chapter 2, FCFS minimises STDW and thus FCFS is a useful lower bound on a discipline's STDW. For this reason STDW is examined in Graph 8.8 for FCFS for varying inter-service times. The peak STDW occurs at the onset of significant contention between requests when the queue length varies considerably with time. The non ideal delays D_1 and D_2 increase STDW because of the effective increase in service time causing absolute waiting times to be magnified.

Graphs 8.9 and 8.10 compare all the disciplines with zero inter-service delay on an STDW basis. As is proved in [S.1], FCFS produces the smallest STDW of any discipline.

Of interest is the effect of applying batching to a discipline. For fixed priority, the effect of batching is substantial, reducing an unbounded increase in STDW to the more gentle rise of the batched fixed priority as shown in Graph 8.9. Batching has little effect on next robin. The example of batched and non batched next robin provides a counter-example to a possible conjecture that batching *always* reduces the STDW of an arbitrary discipline. Batched LRU has a smaller STDW than LRU as is shown in Graph 8.10. Batching tends to apply a degree of service ordering based on arrival times which more closely resembles FCFS, and hence tends to reduce STDW.

The two non derived batched disciplines, batched forward round robin and batched reverse round robin, are also compared in Graph 8.10 with the latter producing a significantly smaller STDW for request rates above

0.5. As the reader may recall from Chapter 2, batched reverse round robin increases priority modulo k after every batch. For example, if requester 2 has highest priority, then in the next batch requester 1 would be allocated highest priority. Batched forward round robin permutes priority in the opposite direction. When the arbiter is heavily loaded, a batch sequence may proceed as in Figure 8.1 in the batched reverse case, and in Figure 8.2 for the batched forward case.

HIGHEST PRIORITY:		1	5	4	3	WAITING TIMES (Unit of
SERVICE:		1 2 3 4	5 1 2 3	4 5 1 2	3 4 5	Service Times)
Req 1	↑	↑	↑		↑	1,3,3
Req 2	↑	↑	↑			2,3,3
Req 3	↑	↑		↑		3,3,3
Req 4	↑		↑	↑		4,3,3
Req 5		↑	↑	↑		2,3,3

FIGURE 8.1 Example of Batched Reverse Round Robin Under Heavy Loading.

HIGHEST PRIORITY:		1	2	3	4	WAITING TIMES (Unit of
SERVICE:		1 2 3 4	2 3 5 1	3 4 5 2	4 5 1 3	Service Times)
Req 1	↑	↑		↑		1,5,5
Req 2	↑	↑	↑		↑	2,2,5
Req 3	↑	↑	↑	↑		3,2,1,4
Req 4	↑		↑	↑		4,4,1
Req 5		↑	↑	↑		4,2,2

FIGURE 8.2 Example of Batched Forward Round Robin Under Heavy Loading.

Figures 8.1 and 8.2 show an arbitrary request sequence consistent with heavy loading. Note that in the batched reverse round robin example,

services cycle in an orderly cyclic fashion with the lowest priority requester in a batch given highest priority in the following batch. The batched forward round robin discipline disrupts the orderly cyclic service flow by assigning to the highest priority request, the lowest priority in the following batch. The large variations in waiting times give rise to a large STDW in the forward discipline. As discussed in Chapter 2, the mean waiting times will be identical for both disciplines. In [C.5] a batched forward round robin circuit is proposed without due consideration for its performance. As has been observed here, a batched *reverse* round robin discipline performs better from a service viewpoint and can be implemented as simply.

Graphs 8.11 and 8.12 show batched disciplines with non zero inter-batch times. It is interesting to observe that disciplines are affected differently when inter-batch times are increasing. At moderate to light request loading (0 to 0.7 say), increasing D_1, D_2 and D_3 results in an increase in STDW, but at heavy request loading STDW for some disciplines increases whilst others decrease.

The light to moderate request loading STDW increase can be explained by the apparent increase in request contention caused by inter-batch times diminishing available time for servicing. The heavy loading behaviour of STDW is dependent on the saturation behaviour of each discipline. For example, batched LRU and batched next robin are not significantly affected by increasing inter-batch times at heavy request loadings due to their adaptive priority allocation schemes which naturally promote a cyclic servicing order. Disciplines which cannot adapt to request patterns tend to perform worse. Batched reverse round robin increases STDW under heavy loading and large inter-batch times, because the cyclic pattern of the servicing is disrupted by the last requester serviced in a batch sometimes requesting within inter-batch times at the

end of a batch and receiving an early service in the following batch. Batched forward round robin is not as noticeably affected since it tends to naturally disrupt the cyclic service stream anyway. Batched fixed priority, as previously discussed, approaches a more cyclic service pattern as inter-batch times increase under heavy request loading.

Since direct comparisons between batched and non batched disciplines with non zero inter-service and inter-batch times is difficult, the non batched disciplines are treated separately.

Graphs 8.13 and 8.14 show such STDW results for non batched disciplines. The light and moderate request loading STDW values increase as inter-service times increase for the following reasons: The waiting time for a request during an idle period is zero, and during the service of another request is at least the remaining service times *plus* inter-service times. Since idle periods occur reasonably frequently under light and moderate request loading conditions, zero waiting times will occur a reasonable number of times. The range of possible waiting times extends from zero to those including other service times. That is, the variance is increased by the additional delay due to inter-service times.

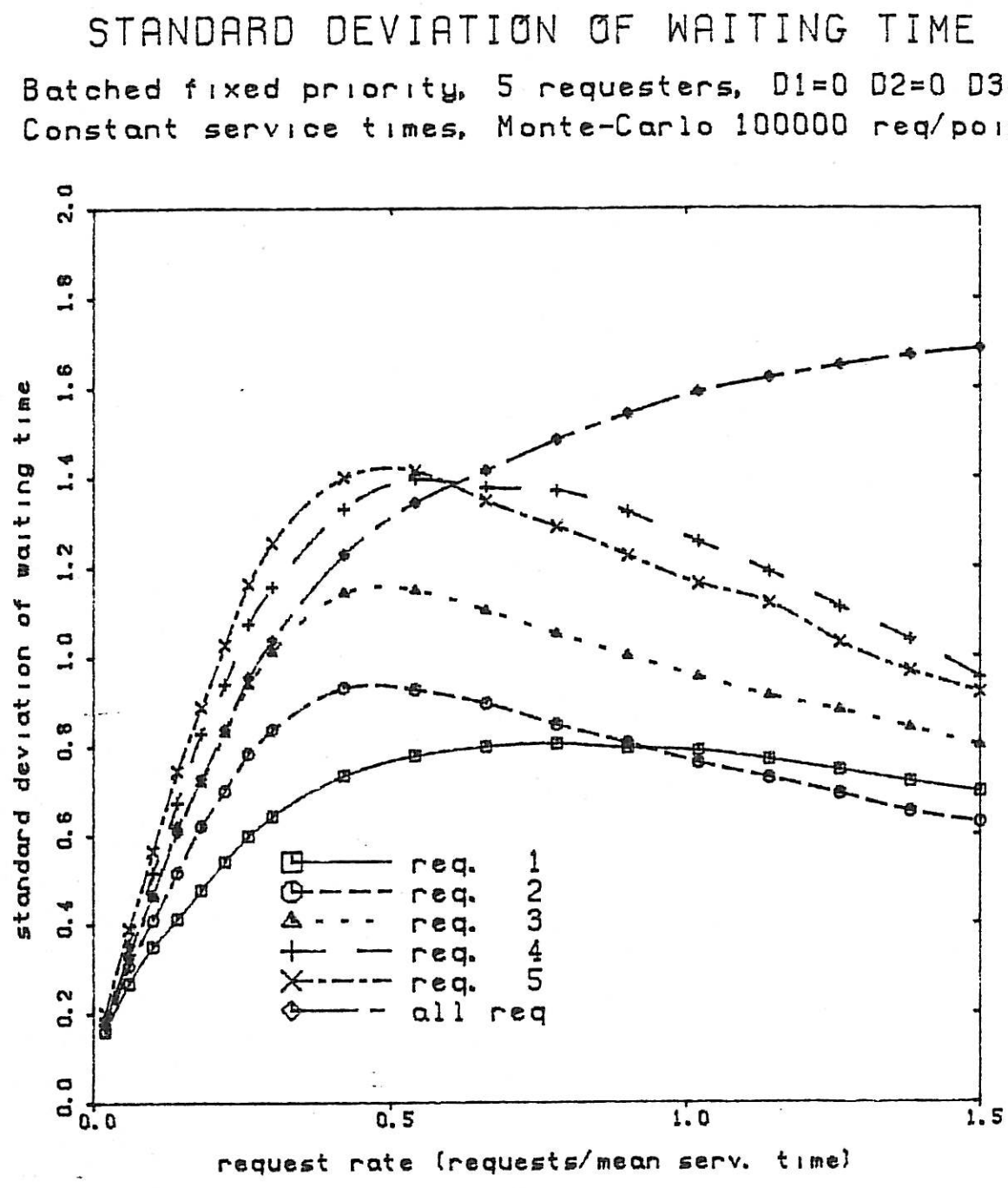
In heavy request loading conditions the effect on the spread of waiting times is less significant because the inter-service delay is incurred on almost every waiting time. The peak STDW is, thus, increased roughly in proportion to the ratio of D_2 to service time. For example, from Graphs 8.9, 8.13 and 8.14, FCFS has the following maximum for STDW:

(i) $D_1 = D_2 = 0$,maximum of 0.9;

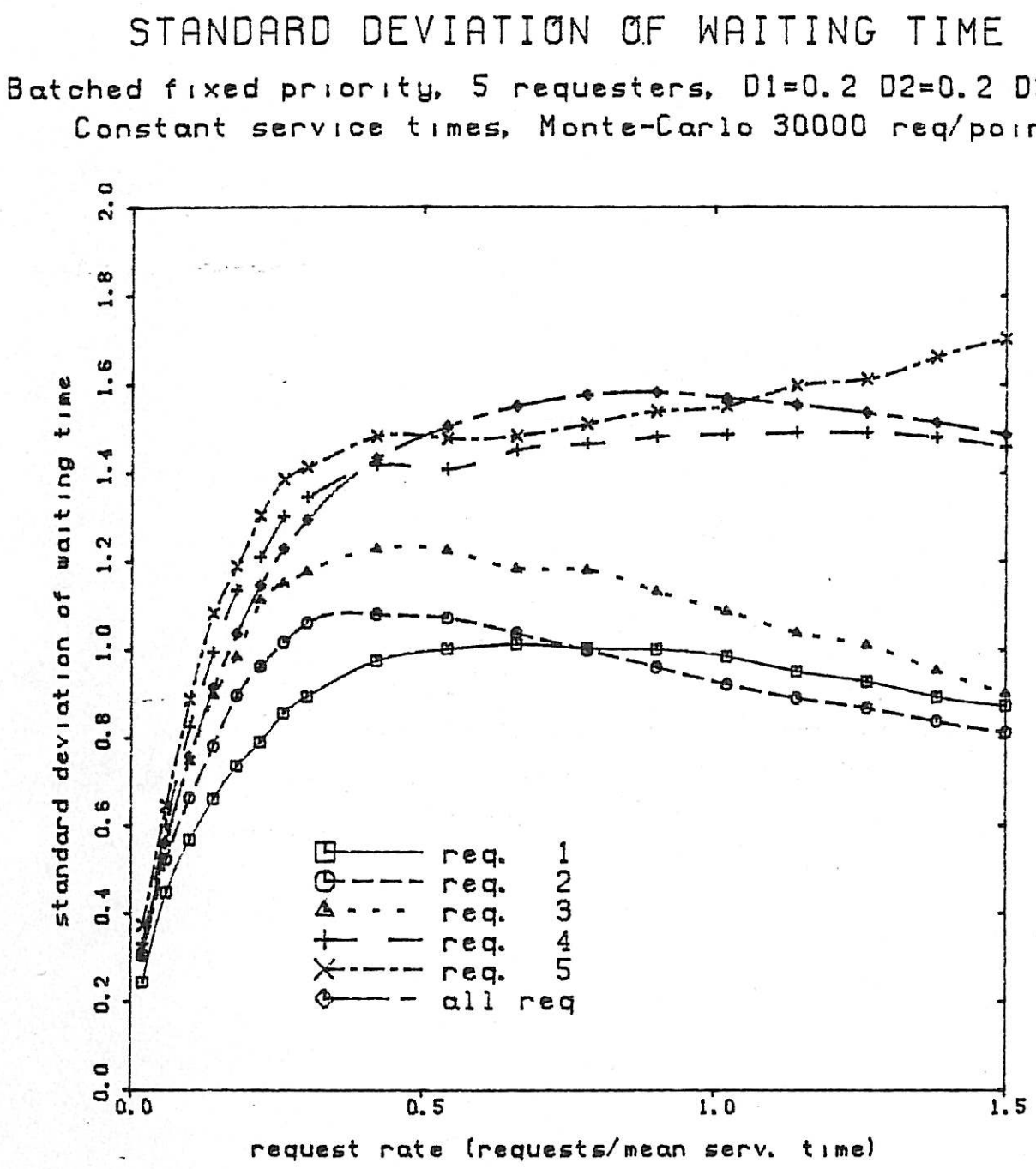
(ii) $D_1 = D_2 = 0.2$,maximum of 1.1;

(iii) $D_1 = D_2 = 1.00$,maximum of 1.85.

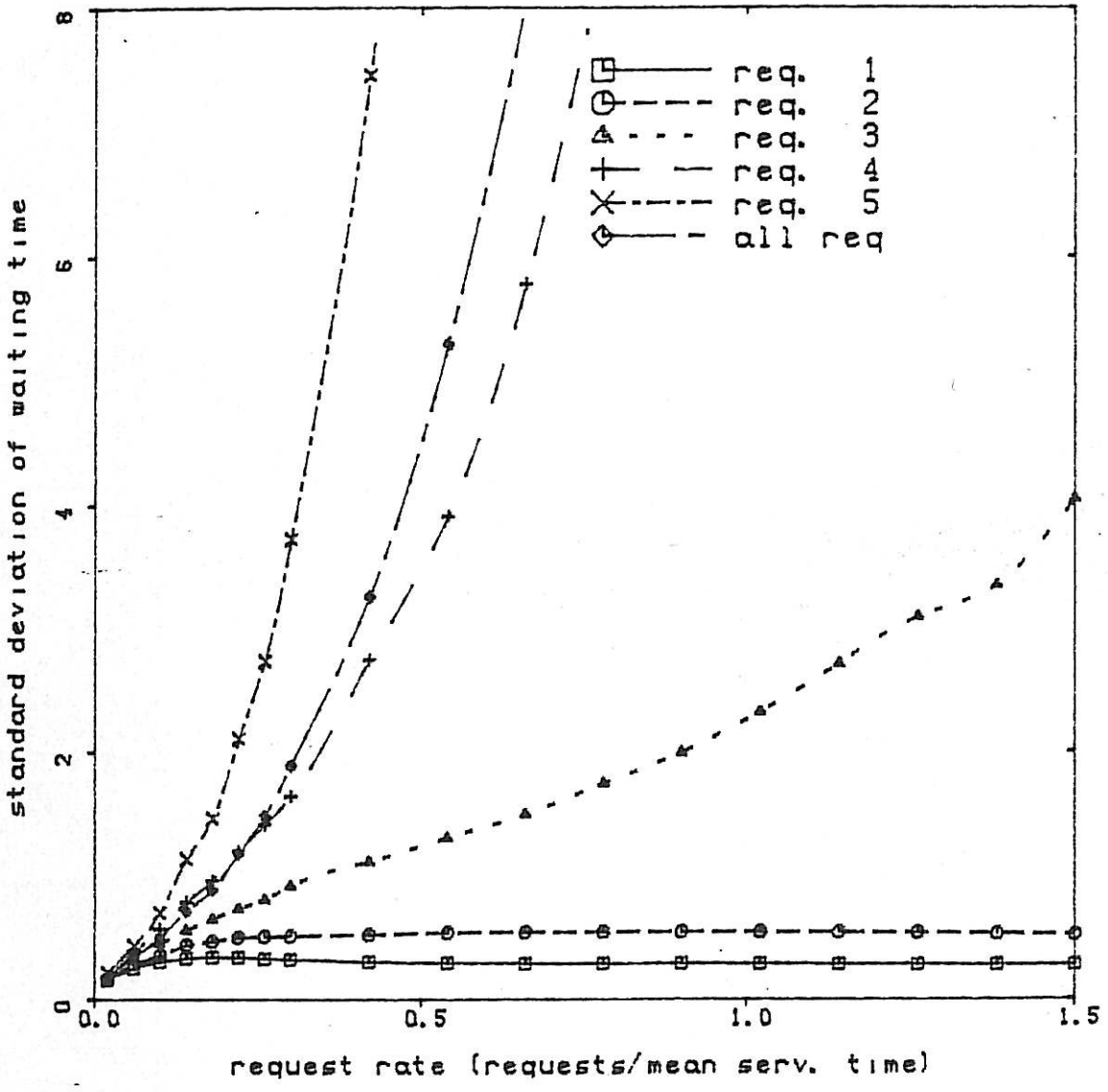
At heavy request loadings, little difference in STDW can be observed.


STANDARD DEVIATION OF WAITING TIME
Batched fixed priority, 5 requesters, D1=0 D2=0 D3=0
Constant service times, Monte-Carlo 100000 req/point
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5
all req

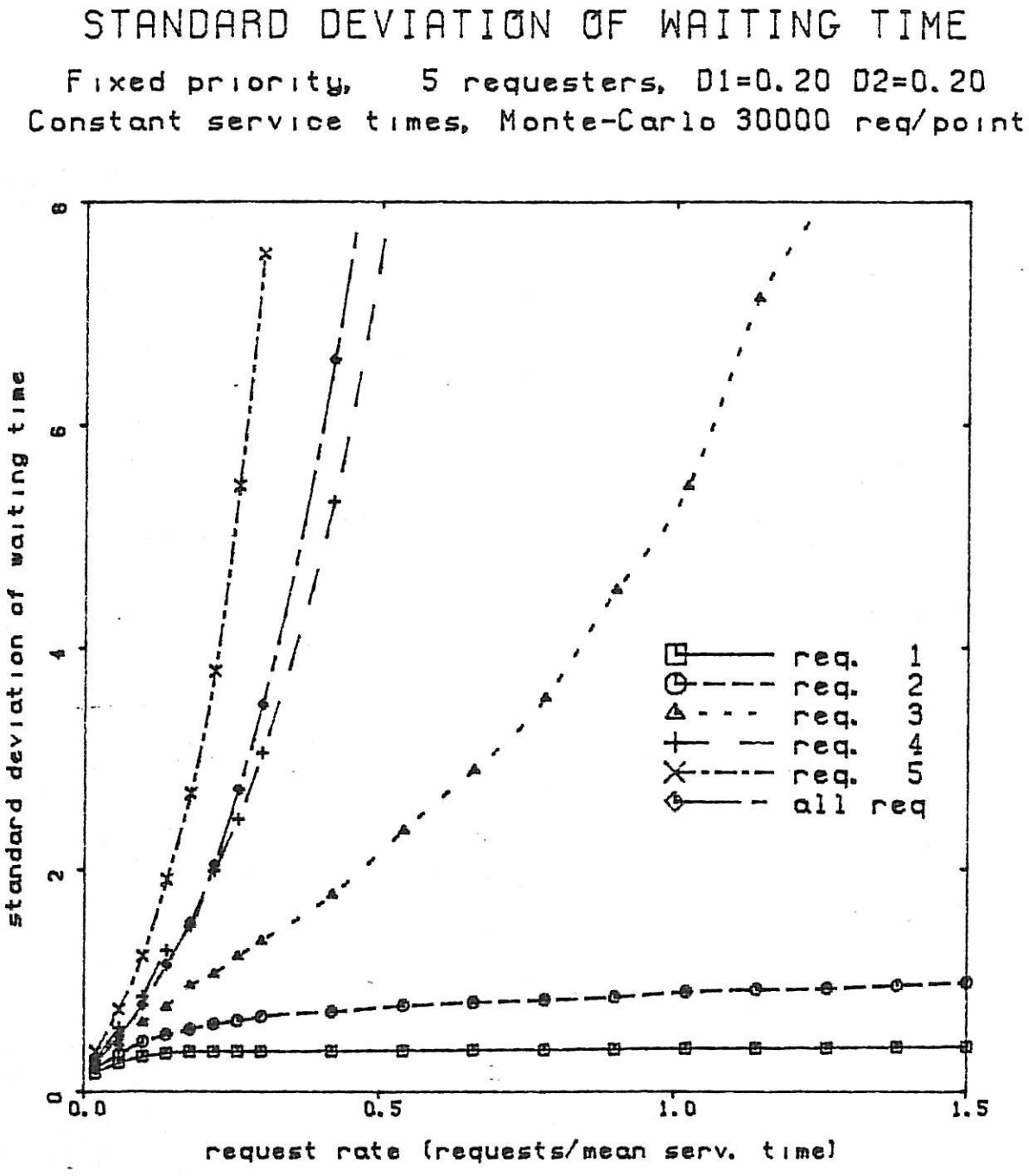
GRAPH 8.4


STANDARD DEVIATION OF WAITING TIME
Batched fixed priority, 5 requesters, D1=0.2 D2=0.2 D3=0.
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5
all req

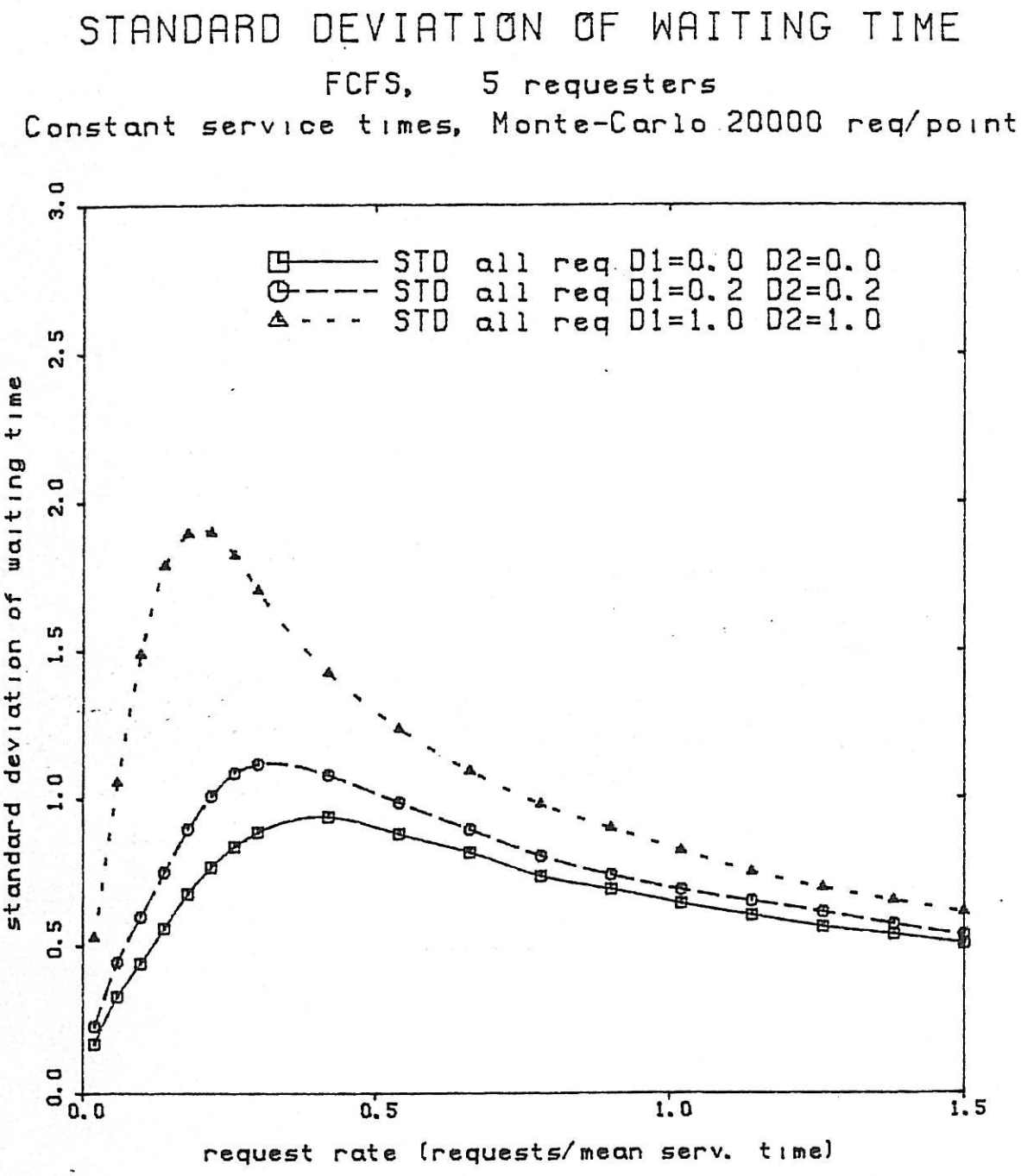
GRAPH 8.5


STANDARD DEVIATION OF WAITING TIME
Fixed priority, 5 requesters, D1=0.00 D2=0.00
Constant service times, Monte-Carlo 30000 req/point
req. 1
req. 2
req. 3
req. 4
req. 5
all req
standard deviation of waiting time
0
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

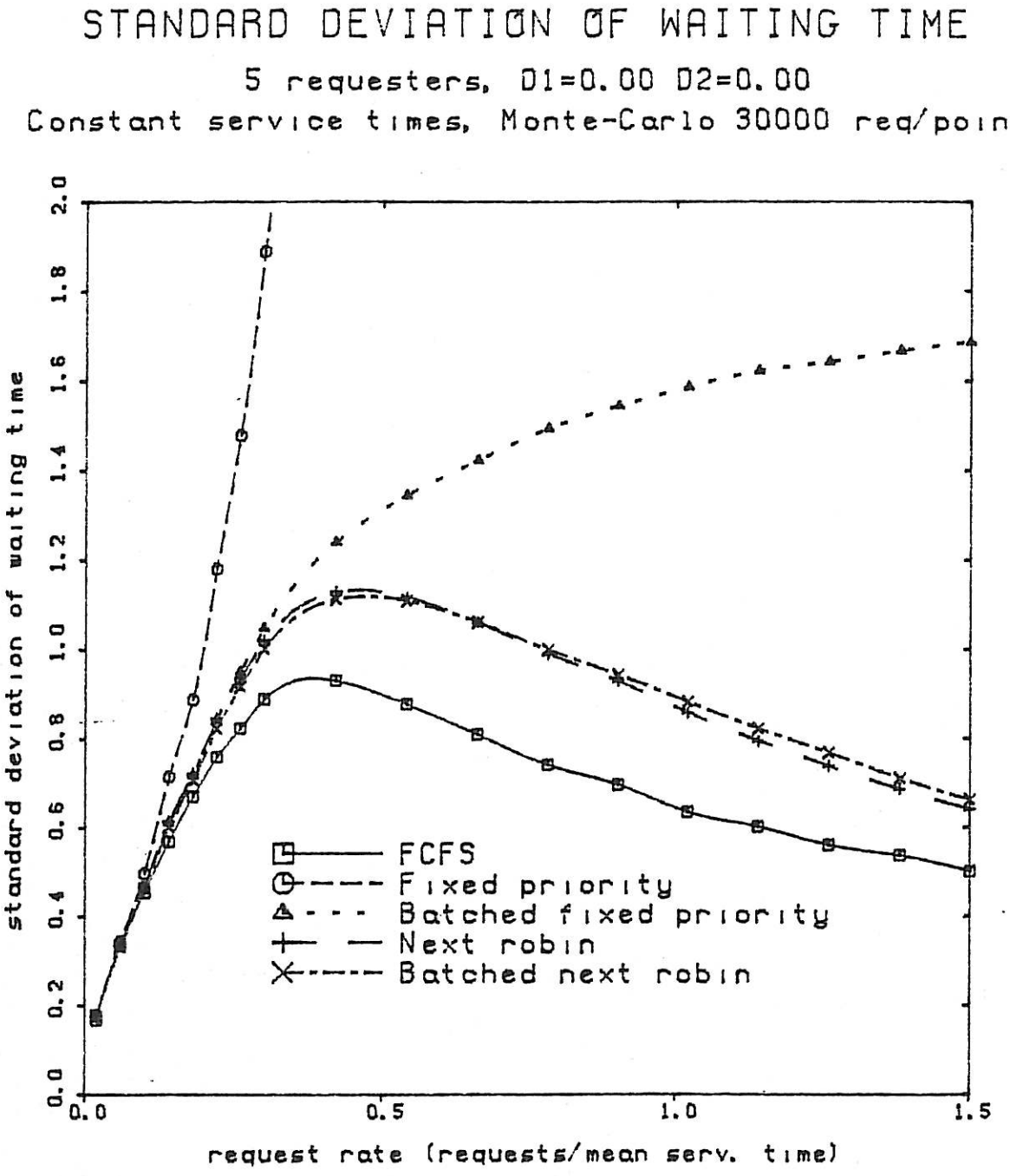
GRAPH 8.6


STANDARD DEVIATION OF WAITING TIME
Fixed priority, 5 requesters, D1=0.20 D2=0.20
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0
2
4
6
8
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
req. 1
req. 2
req. 3
req. 4
req. 5
all req

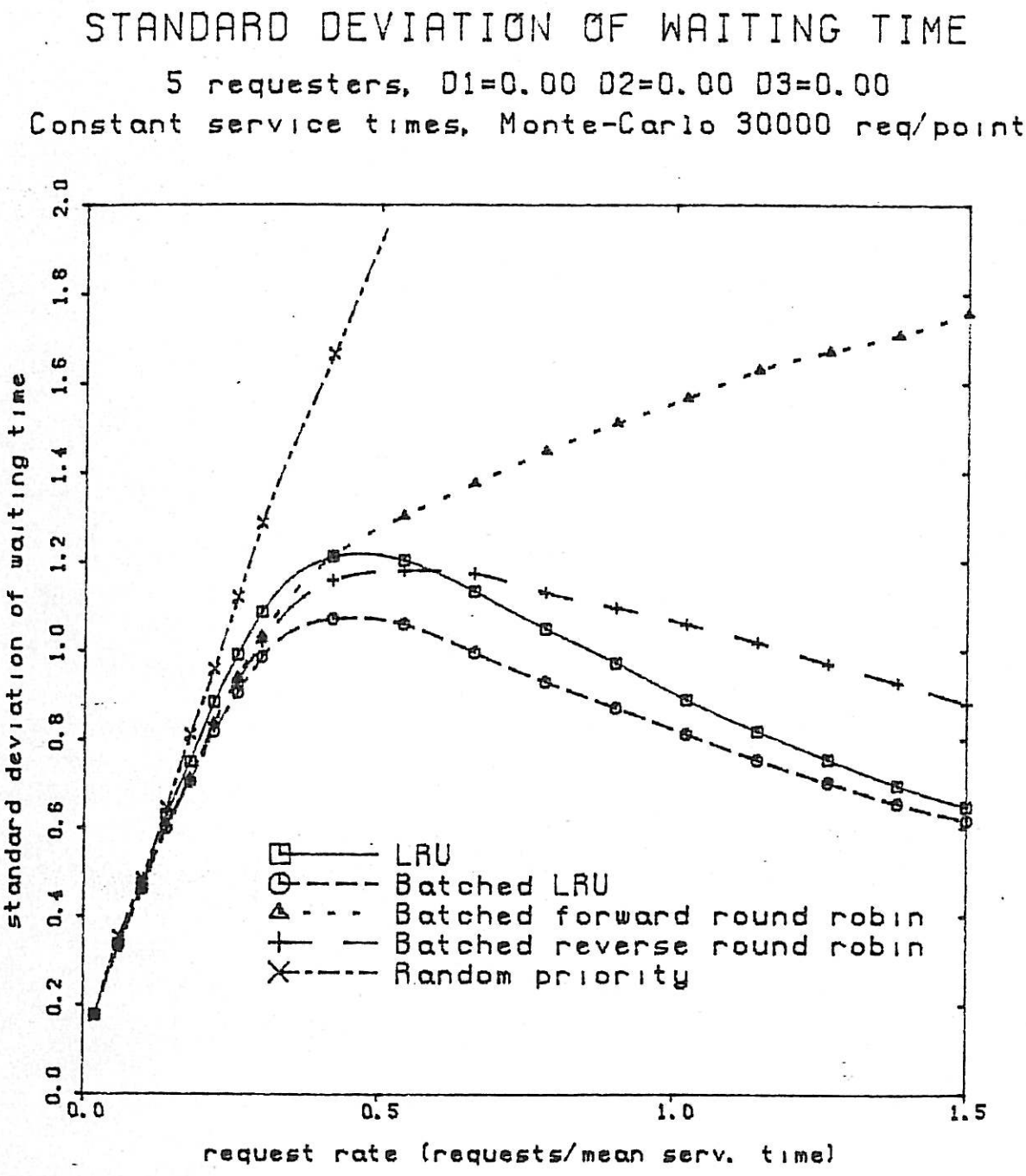
GRAPH 8.7


STANDARD DEVIATION OF WAITING TIME
FCFS, 5 requesters
Constant service times, Monte-Carlo 20000 req/point
STD all req D1=0.0 D2=0.0
STD all req D1=0.2 D2=0.2
STD all req D1=1.0 D2=1.0
standard deviation of waiting time
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

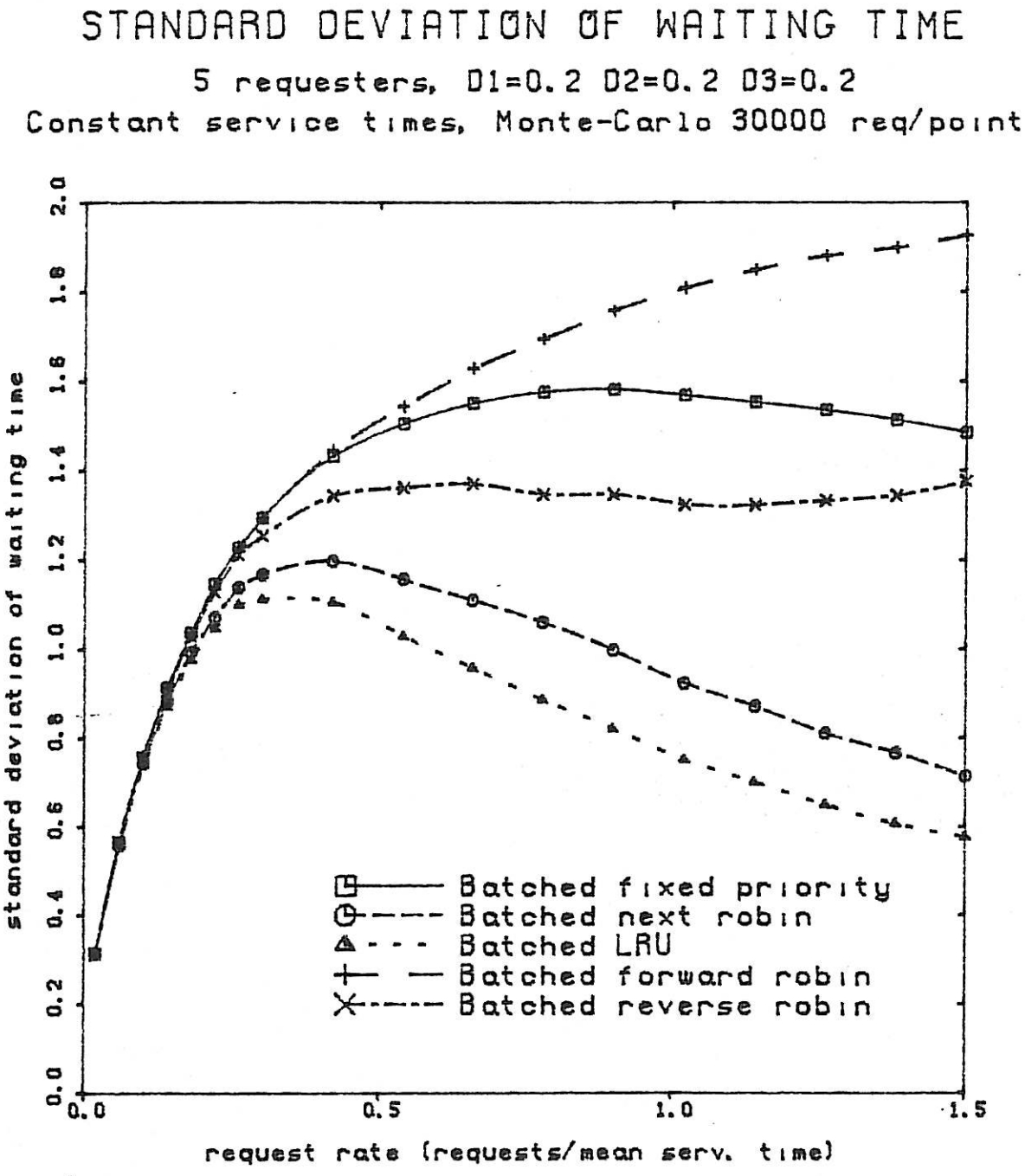
GRAPH 8.8


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=0.00 D2=0.00
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
FCFS
Fixed priority
Batched fixed priority
Next robin
Batched next robin

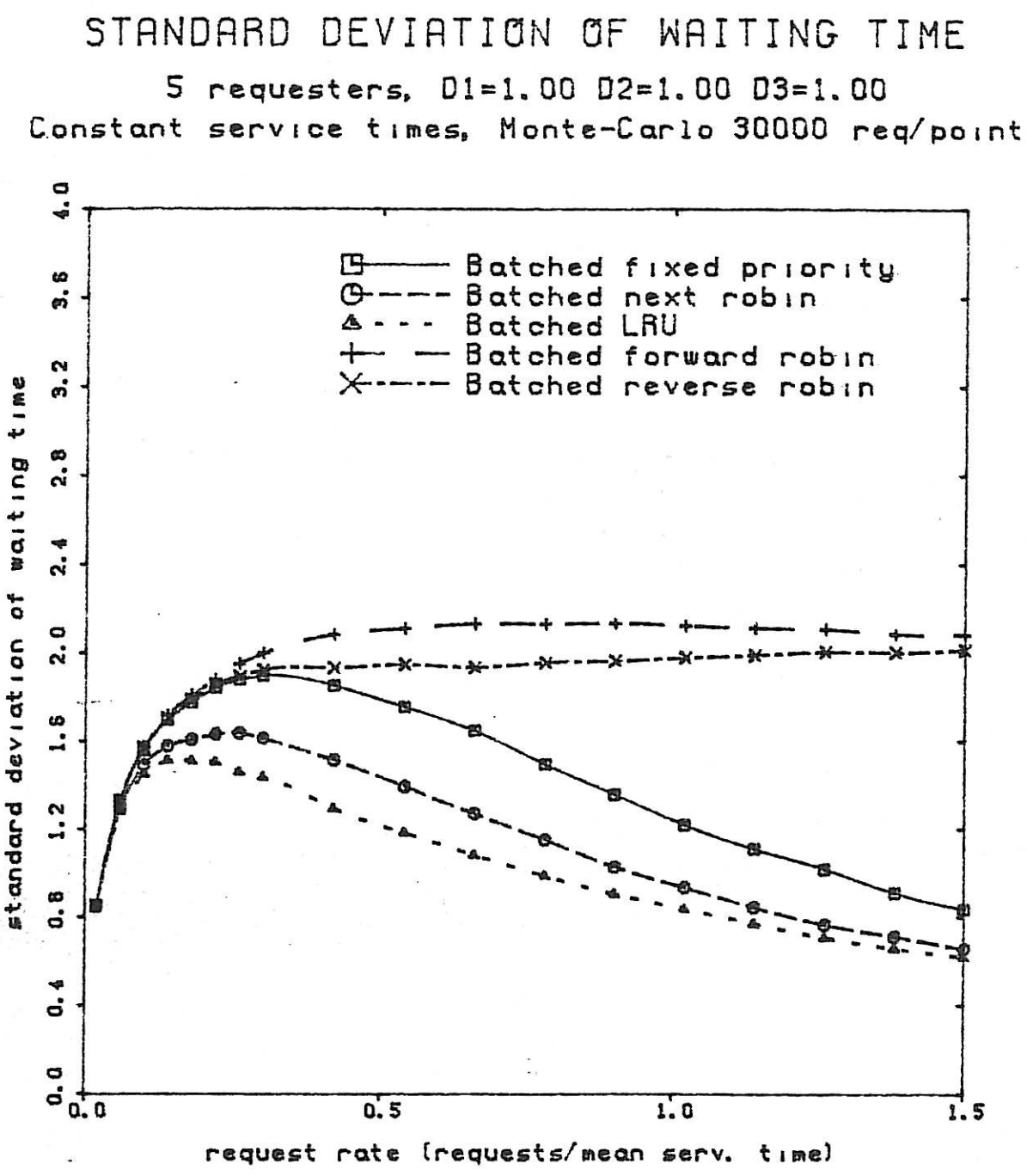
GRAPH 8.9


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=0.00 D2=0.00 D3=0.00
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
LRU
Batched LRU
Batched forward round robin
Batched reverse round robin
Random priority

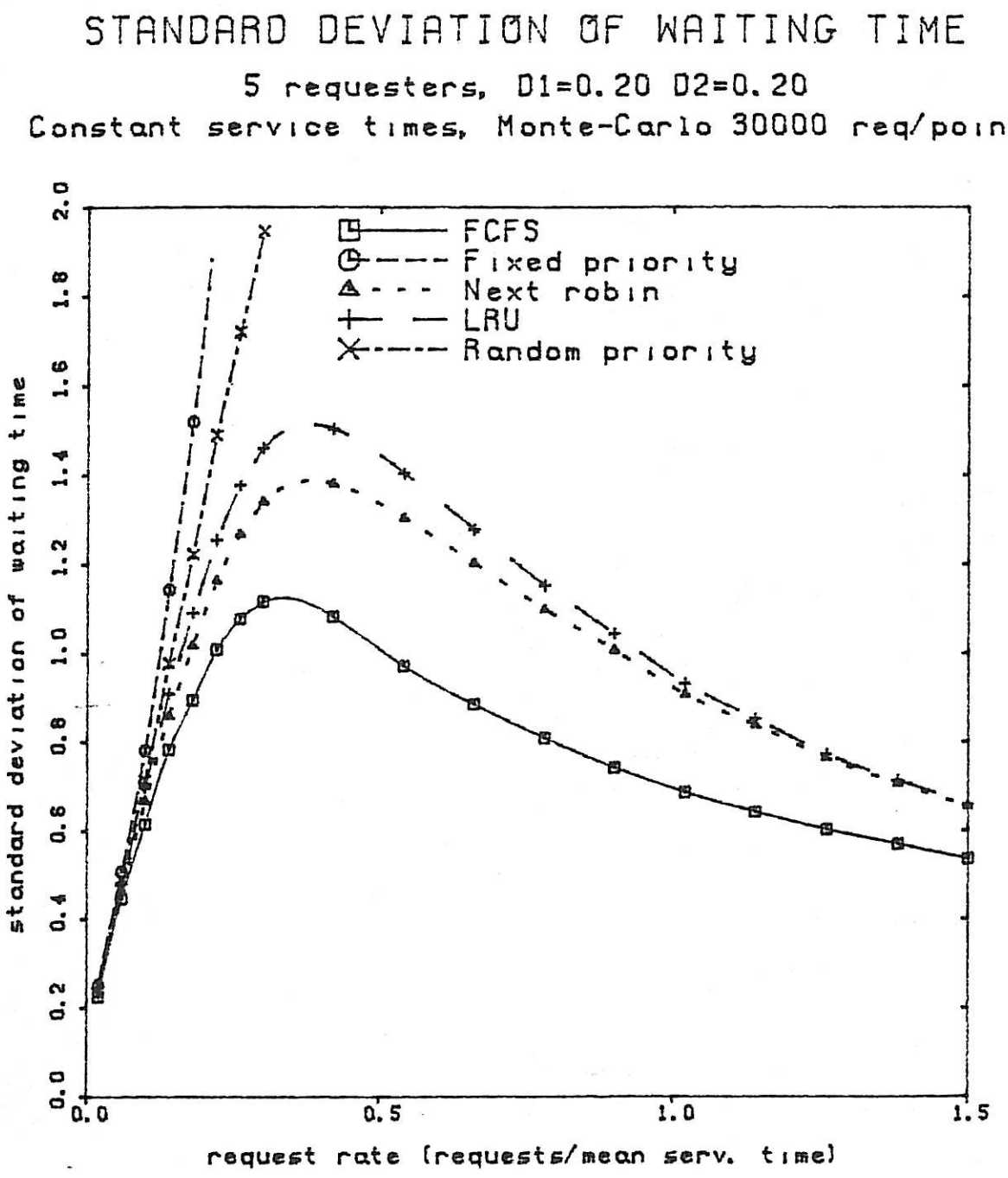
GRAPH 8.10


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=0.2 D2=0.2 D3=0.2
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
Batched fixed priority
Batched next robin
Batched LRU
Batched forward robin
Batched reverse robin

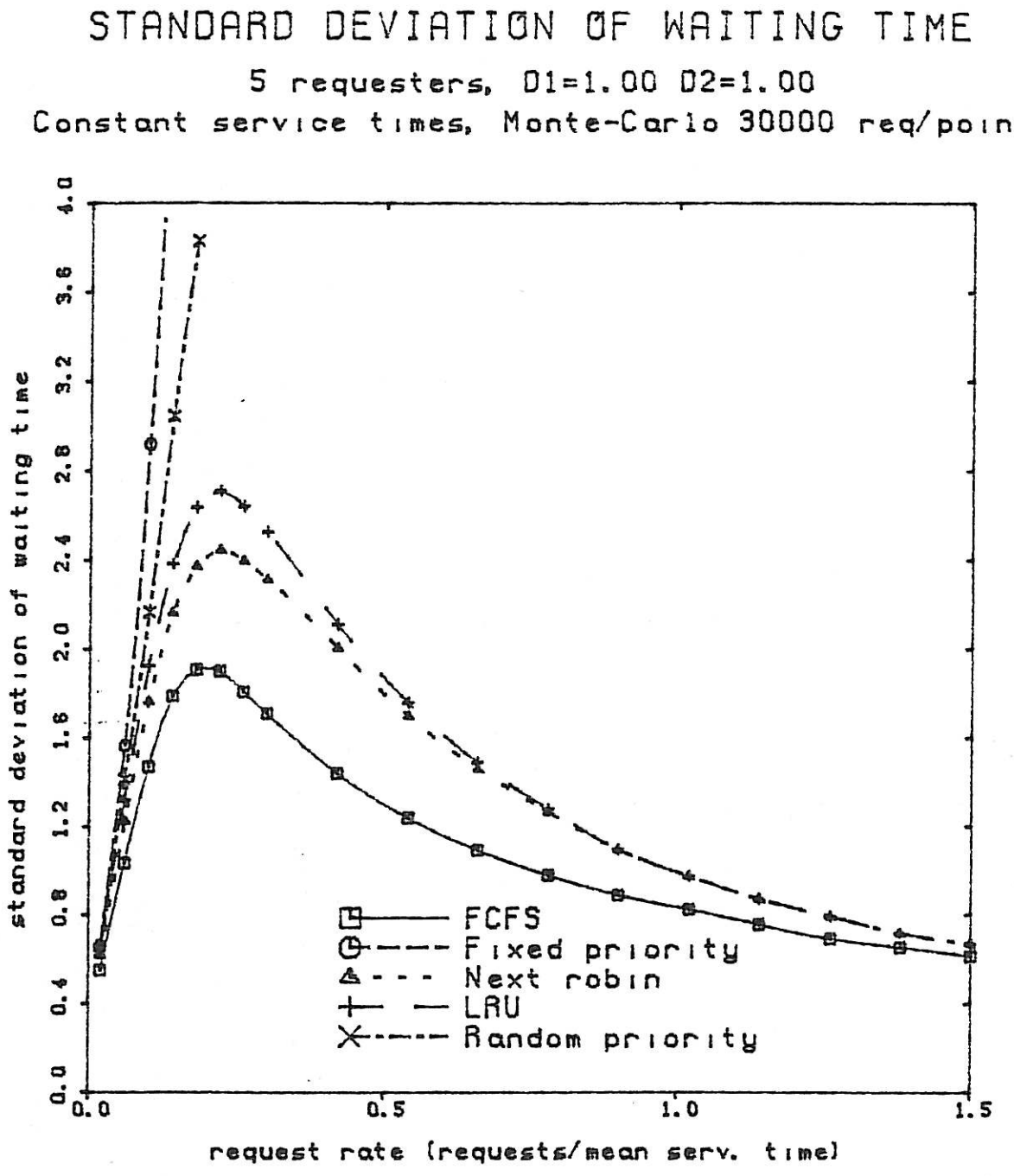
GRAPH 8.11


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=1.00 D2=1.00 D3=1.00
Constant service times, Monte-Carlo 30000 req/point
Batched fixed priority
Batched next robin
Batched LRU
Batched forward robin
Batched reverse robin
standard deviation of waiting time
0.0
0.4
0.8
1.2
1.6
2.0
2.4
2.8
3.2
3.6
4.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.12


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=0.20 D2=0.20
Constant service times, Monte-Carlo 30000 req/point
FCFS
Fixed priority
Next robin
LRU
Random priority
standard deviation of waiting time
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
2.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.13


STANDARD DEVIATION OF WAITING TIME
5 requesters, D1=1.00 D2=1.00
Constant service times, Monte-Carlo 30000 req/point
standard deviation of waiting time
0.0
0.4
0.8
1.2
1.6
2.0
2.4
2.8
3.2
3.6
4.0
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)
FCFS
Fixed priority
Next robin
LRU
Random priority

GRAPH 8.14

8.3 NORMALISED METASTABLE FAILURE RATE PERFORMANCE

In this section NMFR results generated by Monte-Carlo simulation are discussed. The simulation approach is based on evaluating the failure rate by counting the number of requesters that could cause failure by requesting within an aperture at the beginning of each state. The technique is similar to that employed in Chapters 5 and 6.

8.3.1 Determination of NMFR for each Discipline

The number of aperture events occurring per state is a function of the implementation of an arbitration discipline. For example, in Chapter 7 it has been found that a clocked arbiter may have a different metastable reliability performance to an asynchronous arbiter implementing the same discipline. Certain assumptions regarding the occurrence of aperture events are made in order to model asynchronous implementations of disciplines. In Chapter 7 results are presented that indicate superior metastability performance may be obtained with an asynchronous design compared to a clocked design. For this reason asynchronous arbiters are modelled in this section. The modelling assumptions are first stated below and then discussed.

(i) All request rates are equal.

(ii) The metastable characteristics of request storage elements in the arbiter implementation are identical.

(iii) A state of a non batched discipline, when referred to as a period of time, includes a decision point followed by the service of one of the requests pending at the decision point. In a batched discipline, the state also includes the service of the remaining requests pending at the time of the decision point.

(iv) An aperture occurs immediately following a decision point. No other apertures occur. Only the asserting of a request during an aperture can cause failure of the arbiter.

(v) An aperture event occurs when a requester

(a) has no request pending at the decision point; and

(b) would cause failure of the arbiter if it asserted a request during the aperture.

(vi) The number of aperture events associated with each decision point is dependent on the discipline.

(vii) For disciplines where the service decision is based on a linear priority ordering (i.e. fixed priority, dynamic priorities and batched versions of these), the number of aperture events is the number of requesters with priority higher than the highest priority request pending at the decision point.

(viii) FCFS is assumed to be implemented with a dynamic priority secondary arbitration discipline, such as in [S.1]. The number of aperture events at a decision point for FCFS is zero if the previous state is not the idle state. The number of aperture events after an idle period is the same as would be the case in the secondary dynamic priority discipline.

Assumption (iv) is a property which a well designed asynchronous arbiter is expected to conform to. The asynchronous examples of fixed priority and batched fixed priority studied in detail in Chapters 2 and 4 are only vulnerable to metastable failure at the beginning of a state. The disciplines studied in the NMFR results can be considered to be

variations on these circuits in that priority is allocated dynamically through some storage mechanism in the circuit. The allocation of priority is assumed to be based on stable past information, such as the previous service history, and so cannot contribute to metastable failure. An exception is FCFS which effectively allocates priority based on the arrival order of requests and consequently the mechanism itself for allocation of priority can be prone to metastable behaviour. This is discussed below.

Assumption (vii) is a generalisation of batched fixed priority and non batched fixed priority NMFR derivations described in Chapters 5 and 6. It relies on the notion that low priority requesters are masked by the circuit state. Any metastable behaviour in the associated low priority storage elements will not be seen at the outputs of the arbiter for a considerably longer period of time than the nominal shortest settling time allowed by the circuit.

Assumption (viii) and (iv) with regard to FCFS are now discussed. Consider the situation of at least one request pending before the end of a non zero state. Any further requests will be serviced at earliest *after* the next service, assuming pending requests have been registered within the arbiter circuit. The mechanism that orders requests based on their arrival times ,that is, the secondary arbitration circuit in [S.1], is vulnerable to metastable behaviour. If the arbitration for the position in the request queue occurs when requests are already pending, the time before this arbitration decision directly affects the service order is at least one service time. It is assumed here that the secondary discipline allows more time for metastable settling when the request queue is not empty since more time is available without delaying the servicing of any request. Thus, apertures need only be considered when the queue is empty.

The aperture events that are counted correspond to a request occurring during an idle period. It is assumed that the aperture position occurs a time D_1 after the first request to end an idle period as shown in Figure 8.3.

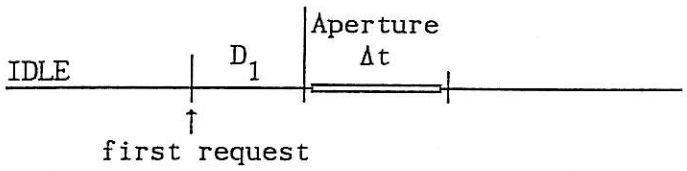


FIGURE 8.3 Aperture Position for FCFS (not to scale).

Firstly, all disciplines with a priority structure are examined, excluding FCFS. The priority of requester h is denoted pr(h) and the highest priority requester in state i is denoted hp(i), where the state is the set of pending requests at the decision point. The sum of the request rates of requesters that can cause failure by requesting within the aperture at the beginning of a state is denoted apert(i), and for priority structured disciplines, is given by

$$\text{apert}(i) \triangleq \sum_{h \in A} \lambda_h \qquad (8.1)$$

where

$$A = \left\{ h : \text{pr}(h) < \text{pr}(\text{hp}(i)) \right\}$$

Equation (8.1) is a generalisation of equations (5.49) and (6.21). Equation (8.1) applies to both batched and non batched disciplines, with decision points occurring at the start of each service for non batched disciplines and at the start of each batch at for batched disciplines.

For FCFS, the number of requesters which can cause failure by requesting within the aperture of Figure 8.3 is now considered. The secondary discipline chooses one requester of all pending requests to be serviced immediately after D_1 has elapsed. Suppose a dynamic priority secondary discipline is employed, as suggested in [S.1][1]. The requesters with higher priority than all pending requests are the only candidates to cause metastable failure by requesting within the aperture. All others are assumed to be masked out because of their lower priorities. If all request rates are equal, then following an idle period every requester is equally likely to have a request pending at the end of D_1. Since the re-request time distributions are exponential, during an idle period the request behaviour is independent of all past history. Each requester is statistically identical since they are memoryless and have the same request rates. Without loss of generality, the requesters are labelled in order of their priority at the decision point following the idle period, with the highest priority requester labelled 1. The situation is thus equivalent to fixed priority at the end of an idle period. The set of requests pending at the end of D_1 is a random sample of the k requesters, each having equal probability. For a state following an idle period the number of aperture events (i.e. the number of requesters which can request within Δt of Figure 8.3 and cause failure) is the number of requesters without requests pending and priority higher than all requests pending. If $|i|$ requests are pending and the state i follows an idle period, the mean number of aperture events is:

---

[1]Not all disciplines can be described by a dynamic priority scheme. For example, a 3 input arbiter may allocate the resource to 1 when 1,2 are pending, 2 when 2,3 are pending and 3 when 1,3 are pending.

$$\frac{\text{apert}(i)}{\lambda} = \frac{\binom{k-1}{|i|} + \binom{k-2}{|i|} + \binom{k-3}{|i|} + \ldots + \binom{|i|}{|i|}}{\binom{k}{|i|}} \qquad (8.2)$$

where $\binom{n}{m}$ is the number of ways of choosing m items from n. When state i does not follow a zero state apert(i) is defined to be zero. Equation (8.2) simplifies to $(k-1)/2$ when $|i|$ is 1 which is a good approximation when D_1 is small. Equation (8.2) also applies to FCFS arbiters, such as the example shown in Figure 2.13, where the queue ordering is determined directly by the use of $k(k-1)/2$ RS flip-flops (one for each pair of requests) which are set/reset depending on the arrival order of requests. This arbiter can be modelled with D_1 as zero. (A practical asynchronous FCFS arbiter based on Figure 2.13 would be modified to ensure the combinational logic resolved circular orderings described in Section 2.5, and also to allow for an adequate metastable settling time). Suppose $D_1 = 0$ and failure occurs when two requests occur within Δt of each other. The number of possible pairs of requests is $k(k-1)/2$. The number of possible states over which these apertures are to be counted is k, where singleton states only are counted since $D_1 = 0$. This gives $(k-1)/2$ apertures per state which agrees with (8.2) when $|i|$ is one.

Once the "apert" function is defined for all disciplines the NMFR can be obtained from equation (5.54) and found to be

$$\text{NMFR} = \frac{\sum_{i=1}^{m-1} \text{apert}(i)\, n_i}{\sum_{j=0}^{m-1} n_j t_j} \qquad (8.3)$$

where n_j is the number of states j and t_i is the mean duration of state j.

8.3.2 Discussion of NMFR Results

Equations (8.1), (8.2) and (8.3) have been applied in the Monte-Carlo simulation program and Graphs 8.15, 8.16, 8.17, 8.18, 8.19, 8.20 and 8.21 show the results.

Graphs 8.15, 8.16 and 8.17 show NMFR for all the disciplines with zero inter-service and inter-batch times. Fixed priority and random priority disciplines performed poorly at high request rates due to aperture events persisting above saturation levels of requests. In other disciplines at heavy request loadings, the arbitration disciplines tend not to give priority to recently serviced requesters which are more likely not to have pending requests. That is high priority is more likely to be given to pending requests which cannot cause failure. The priority is such that low priority tends to be assigned to recently serviced requesters. This occurs ***explicitly*** in LRU and ***indirectly*** in round robin schemes where the priorities encourage servicing to occur in a cyclic fashion.

All disciplines have approximately the same NMFR for light request loadings - a common feature of all the NMFR results. The similarity follows from the fact that little correlation exists between priority and request arrivals since the arbiter cannot utilise previous history in masking synchronisation failure as occurs at heavy loading. Little interference between arrival times of requests occurs because very little contention is present at low request rates. Thus, the request input process is approximately Poisson and consequently the past request/service history is of little use in predicting future arrivals. For these reasons at low request rates a fundamental NMFR is suspected, independent of arbitration discipline. It is shown here that for all disciplines considered in this chapter the light loading NMFR's coincide.

The arbiter alternates between idle and singleton service as shown in Figure 8.4 for the non batched arbiter model.

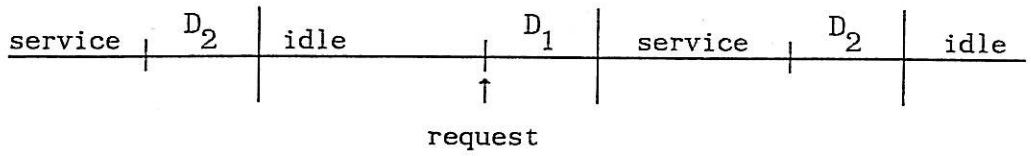


FIGURE 8.4 Low Request Rate Service Pattern for Non Batched Arbiters.

The mean idle period is $1/k\lambda$. For $\lambda << \frac{\mu}{k}$ the mean rate of singleton services is approximately

$$\frac{\sum_{|i|=1}^{m-1} n_i}{\sum_{j=0} n_j t_j} \cong \frac{1}{\frac{1}{k\lambda} + D_1 + \frac{1}{\mu} + D_2 + D_3} \qquad (8.4)$$

where $D_3 = 0$ for non batched arbiters.

As follows from the previous section, under light request loading conditions $\text{apert}(i) \simeq \left[\frac{k-1}{2}\right]\lambda$ for $|i| = 1$. It follows that

$$\text{NMFR} \cong \frac{\lambda\left[\frac{k-1}{2}\right]}{\frac{1}{k\lambda} + D_1 + \frac{1}{\mu} + D_2 + D_3} \qquad (8.5)$$

$$\cong \lambda^2 \frac{k(k-1)}{2} \quad \text{for} \quad \frac{1}{k\lambda} >> \left[\frac{1}{\mu} + D_1 + D_2 + D_3\right] \qquad (8.5a)$$

Equation (8.5a) can be seen to be a good approximation in Graphs 8.15, 8.16, 8.17, 8.18 and 8.20 for $\lambda < 0.1$ and in Graphs 8.19 and 8.21 for $\lambda < 0.05$.

Another common feature of the NMFR results is that the peak failure rate occurs at approximately the onset of saturation $\left[\lambda \cong \frac{1}{k-1}\right]$ in all disciplines excluding fixed priority and random priority. At peak failure rates, the arbiters are resolving maximum contention between requesters. At heavy request loadings the request pattern is dictated more by the order of service and the arbiter can use this information to effectively reduce the rate of aperture events.

In the ideal case of zero inter-service and inter-batch times, it is clear from Graphs 8.15 and 8.16 that all batched disciplines have a smaller NMFR than the corresponding non batched disciplines at moderate to heavy request loadings. This result can be attributed to two factors :

(i) batched disciplines have a lower rate of decision points where apertures occur, and

(ii) the number of aperture events at each decision point is likely to be smaller because the duration between decision points is longer, increasing the number of pending requests. More pending requests results in greater metastable masking and less candidates for requests during an aperture.

The order of "merit" , based on magnitude of NMFR, is FCFS, batched LRU and batched next robin (approximately equal), batched fixed priority and LRU (approximately equal), batched forward robin, batched reverse robin, random priority and worst of all, fixed priority.

The NMFR performance of batched reverse robin at heavy request rates deserves some comment. At first one may be perplexed by the apparent poor performance, after studying Figure 8.1. However, other "modes" of operation are possible at heavy request loadings as illustratated in Figure 8.3, where NMFR is large.

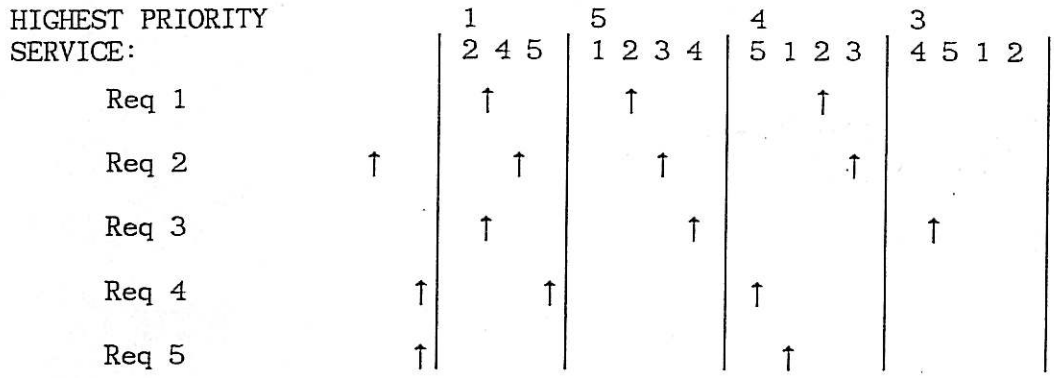
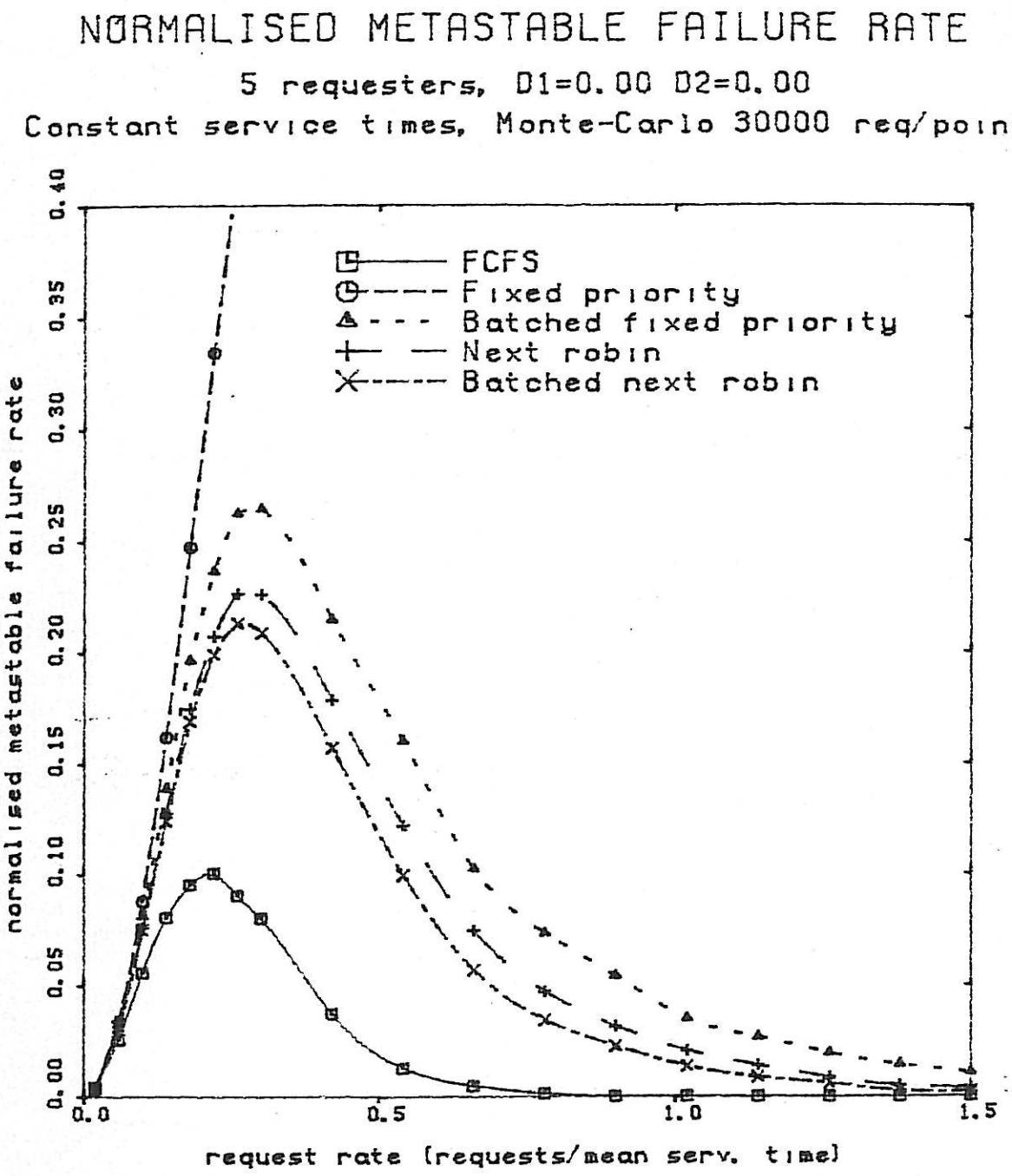
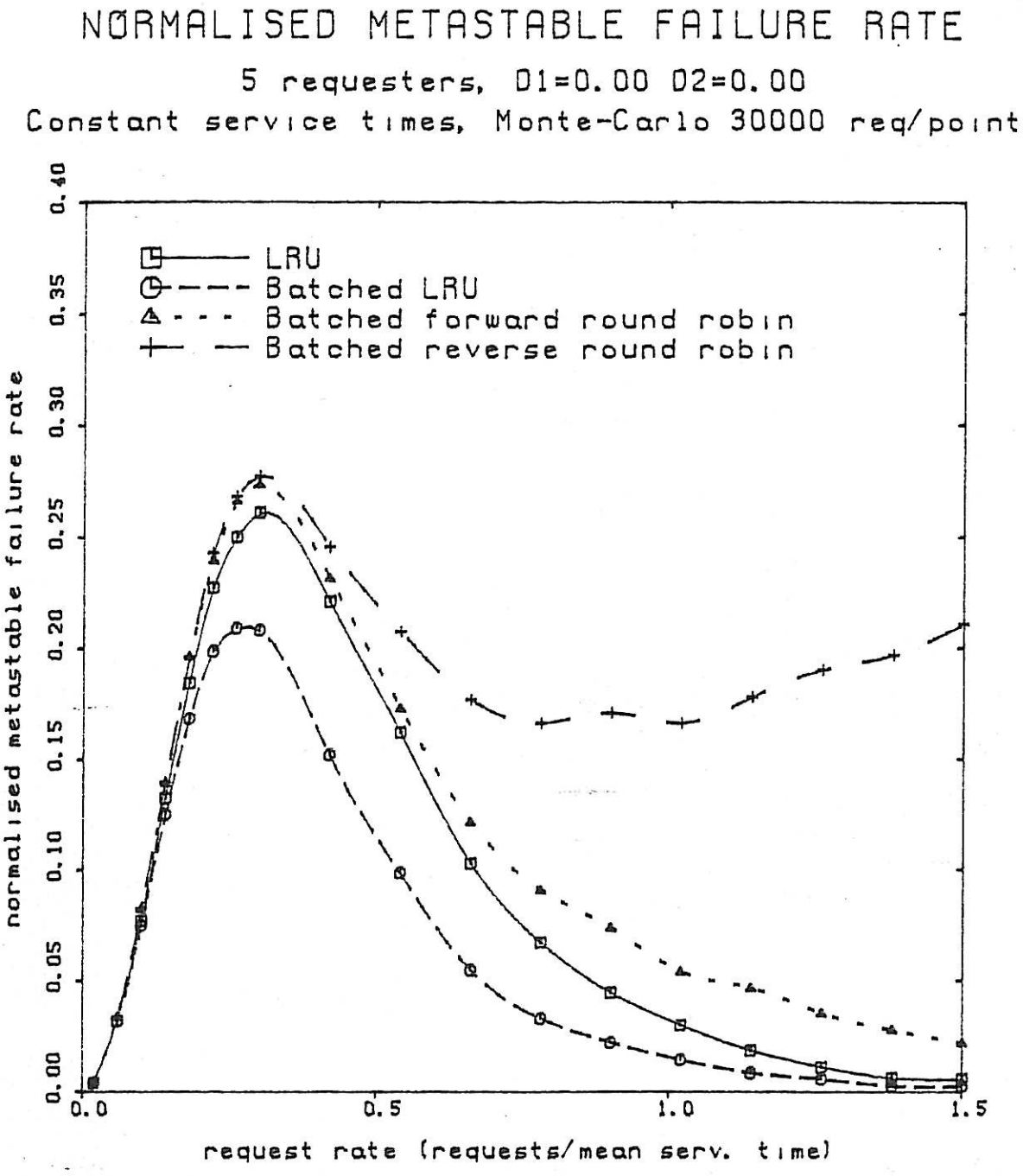


FIGURE 8.5 Alternative Low STDW Example of Batched Reverse Round Robin with Large NMFR.

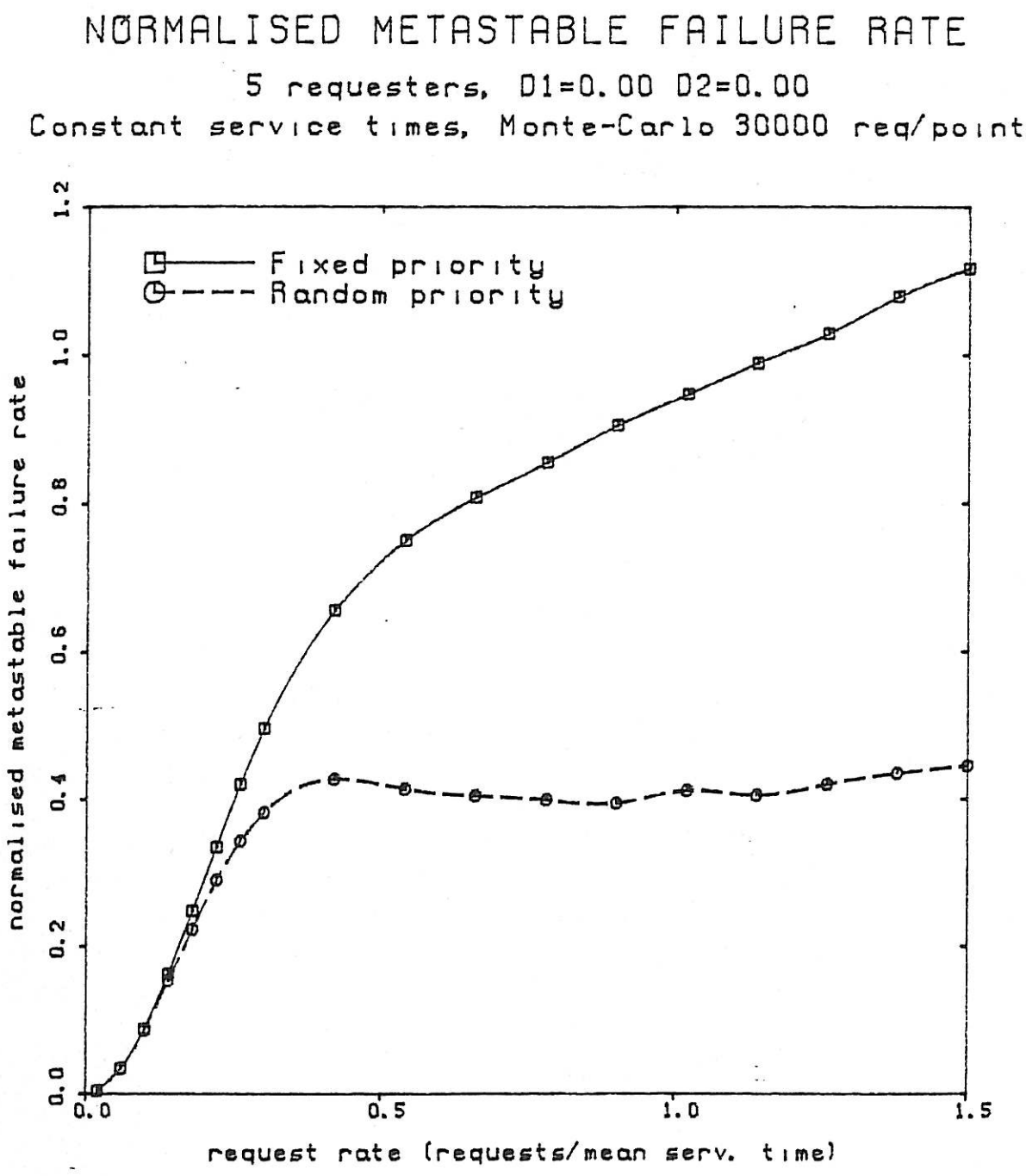
In Figure 8.5, the second highest priority requester is serviced first in each batch since the highest priority request is serviced *last* in the previous batch and cannot request before the batching point. As a result, an aperture is presented at the beginning of each batch where the high priority requester can cause failure. Because the "mode" of the behaviour is largely determined by initial request conditions under heavy request loadings, the observed fluctuation in the NMFR results occurs in Graphs 16, 18 and 19. Similar behaviour does *not* occur in batched *forward* round robin because the highest priority requester in the next batch has a second highest priority in the current batch and consequently is very likely to request before the batching point as can be seen in Figure 8.2.


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=0.00 D2=0.00
Constant service times, Monte-Carlo 30000 req/point
FCFS
Fixed priority
Batched fixed priority
Next robin
Batched next robin
normalised metastable failure rate
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

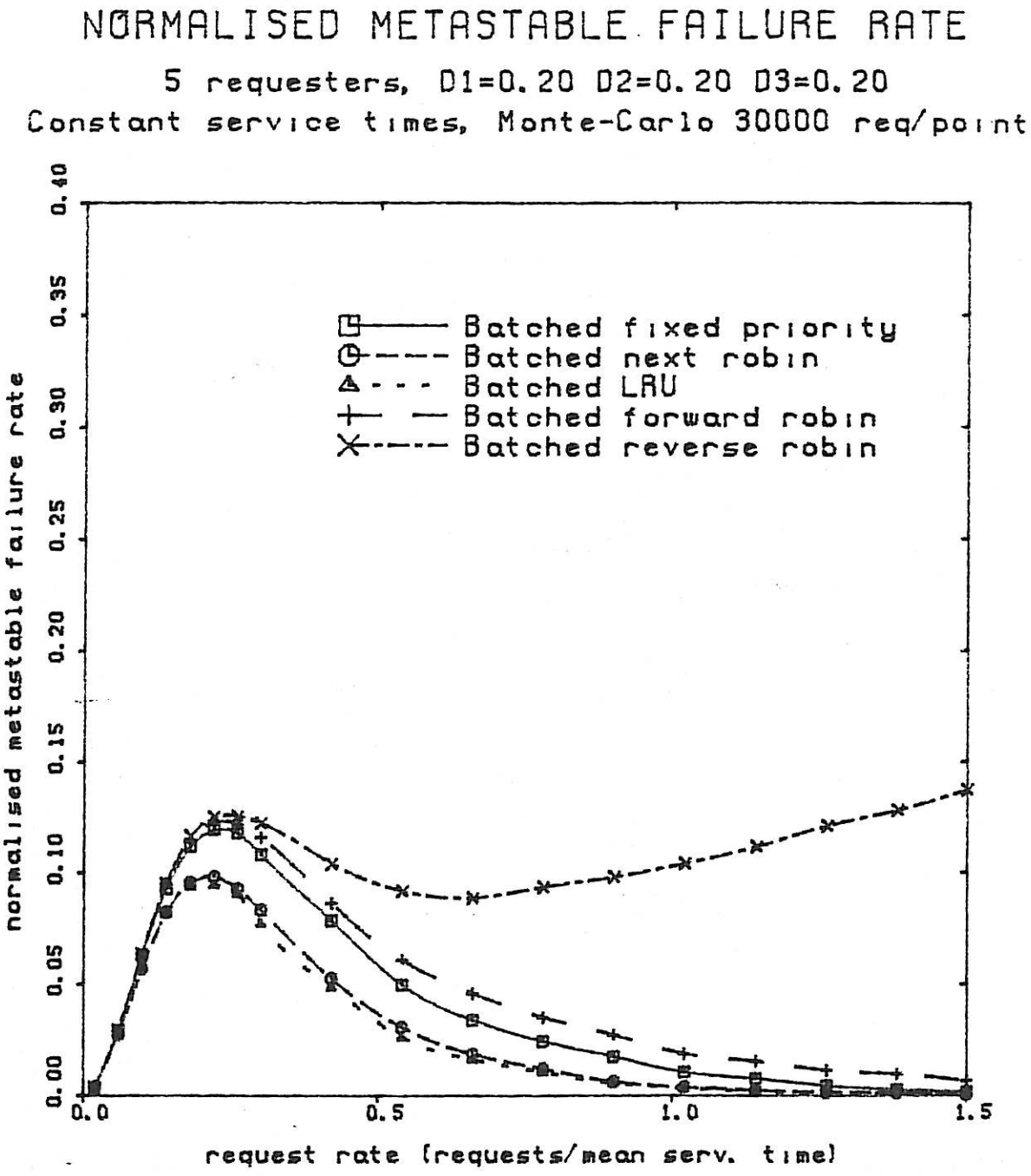
GRAPH 8.15


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=0.00 D2=0.00
Constant service times, Monte-Carlo 30000 req/point
LRU
Batched LRU
Batched forward round robin
Batched reverse round robin
normalised metastable failure rate
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

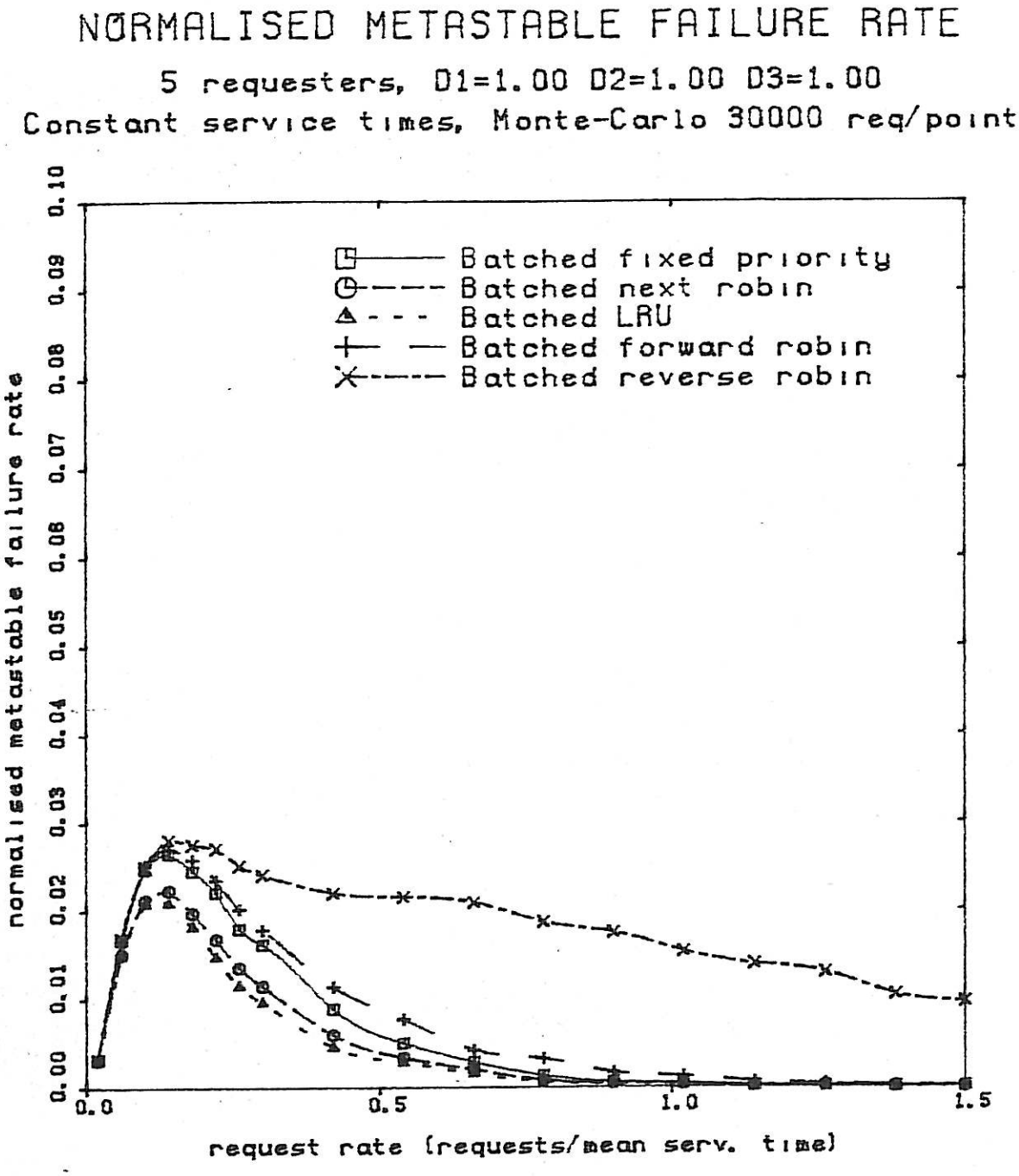
GRAPH 8.16


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=0.00 D2=0.00
Constant service times, Monte-Carlo 30000 req/point
Fixed priority
Random priority
normalised metastable failure rate
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

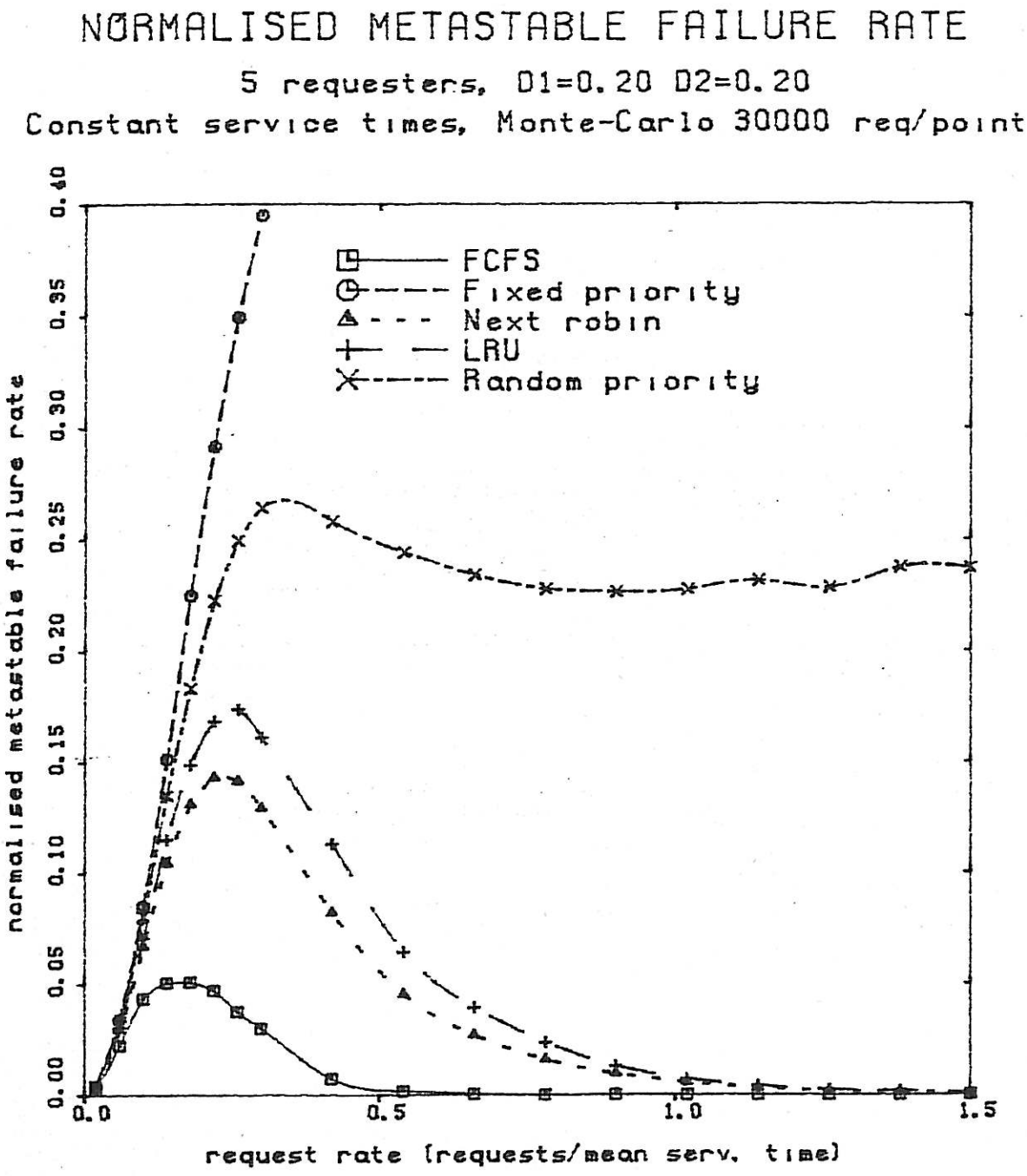
GRAPH 8.17


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=0.20 D2=0.20 D3=0.20
Constant service times, Monte-Carlo 30000 req/point
Batched fixed priority
Batched next robin
Batched LRU
Batched forward robin
Batched reverse robin
normalised metastable failure rate
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

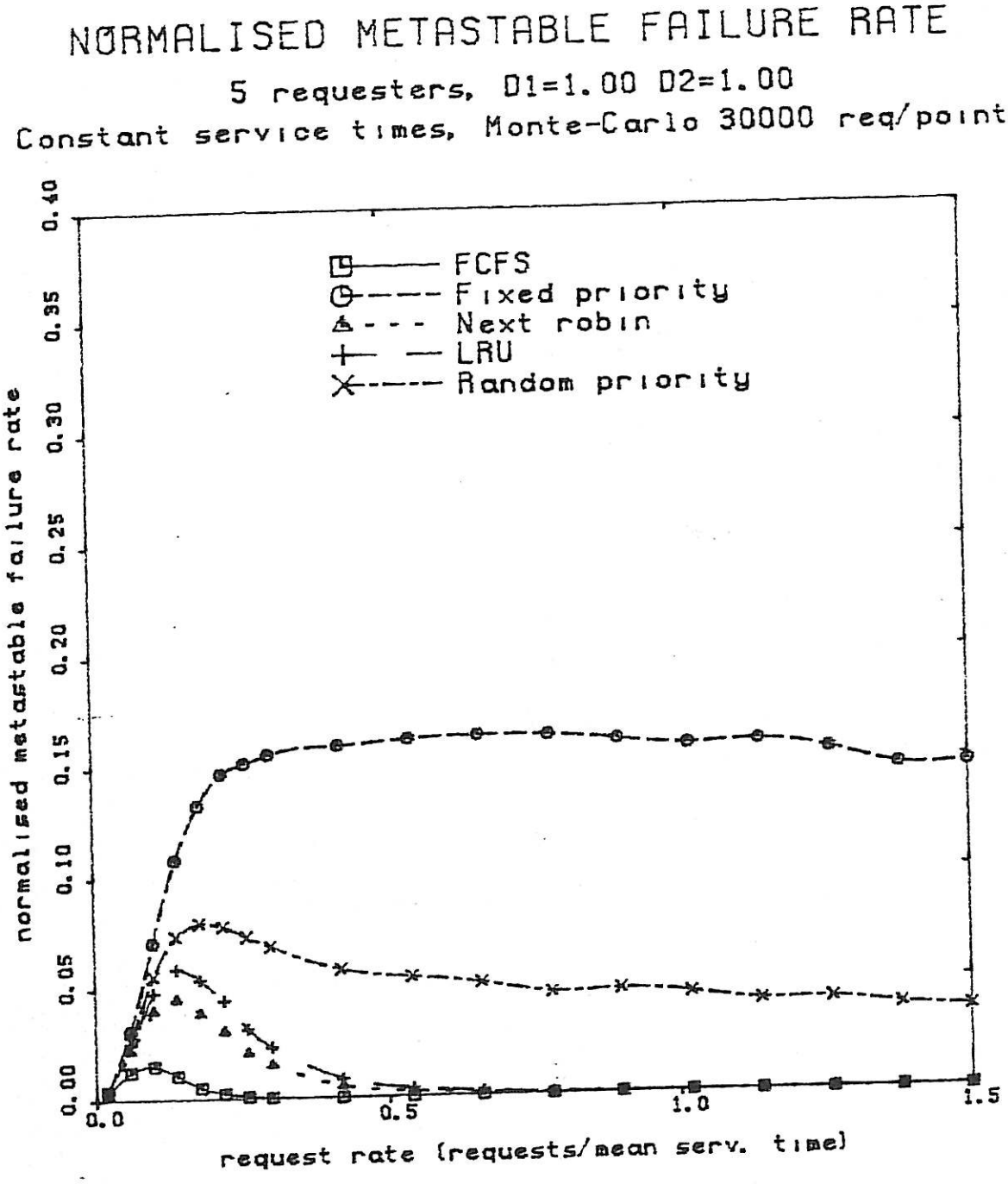
GRAPH 8.18


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=1.00 D2=1.00 D3=1.00
Constant service times, Monte-Carlo 30000 req/point
Batched fixed priority
Batched next robin
Batched LRU
Batched forward robin
Batched reverse robin
normalised metastable failure rate
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.10
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.19


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=0.20 D2=0.20
Constant service times, Monte-Carlo 30000 req/point
FCFS
Fixed priority
Next robin
LRU
Random priority
normalised metastable failure rate
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.20


NORMALISED METASTABLE FAILURE RATE
5 requesters, D1=1.00 D2=1.00
Constant service times, Monte-Carlo 30000 req/point
FCFS
Fixed priority
Next robin
LRU
Random priority
normalised metastable failure rate
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.0
0.5
1.0
1.5
request rate (requests/mean serv. time)

GRAPH 8.21

8.4 CONCLUSIONS

The performance of a wide range of disciplines has been compared when all requesters have the same characteristics. The results contain a large amount of information which is briefly summarised here.

The discipline with the best performance is FCFS, both from a metastable failure rate viewpoint and service viewpoint. This assumes the inter-service times D_1 and D_2 can be made small which could prove a problem when secondary arbitration is employed in an implementation. As discussed in Chapter 2, FCFS is one of the most complex disciplines to implement. A good compromise in complexity and performance may be the disciplines batched LRU and batch next robin, both offering good service and reliability performance.

It has been found, generally, that batched disciplines offer advantages over their primary non batched disciplines, both in terms of utilisation efficiency and performance, and their metastable failure rate characteristics. As discussed in Chapter 2 batched disciplines involve the addition of minimal extra hardware compared with the primary discipline.

Disciplines that have revealed very poor service and reliability performance are fixed priority and random priority, which unfortunately can be attractive from a hardware simplicity viewpoint. Random priority has been implemented in an approximate form [M.1] using a rapidly rotating grant signal. Such schemes have potentially very poor metastable failure rates because of both their inherent quasi-random priority and the necessity for very short metastable settling time allowances in each module to maintain a fast rotation of the grant.

It has been found that at light request rates, a fundamental NMFR exists independent of the discipline, which is proportional to the square of both the request rate and number of requesters in the arbiter. Also, for disciplines with reasonable NMFR performance at heavy request loadings, a *peak* failure rate exists corresponding to the onset of saturation of requests. The peak occurs approximately at request rates that could fully utilise the resource with no waiting times under orchestrated deterministic requesting conditions. Random non deterministic request behaviour at the same mean rate results in contention and less utilisation of the resource.

The effect of non ideal inter-service times has been found to affect both service performance and metastable failure rate. Increasing the relative inter-service delays, results in poorer utilisation and service performance. However, somewhat surprisingly, the metastable failure rate decreases with increasing inter-service delays. Purposeful introduction of inter-service delay as a means for decreasing the failure rate is only recommended if the delay directly increases the settling time given to synchronising elements.

CHAPTER 9

CONCLUSIONS AND EXTENSIONS

9.1 SUMMARY AND CONCLUSION

In this thesis, new techniques and results have been developed for the modelling and analysis of the performance of arbiters. Both service performance and metastability performance have been considered. An important contribution of the thesis is the inclusion of the modelling and analysis of the performance of arbiters with respect to metastability of devices employed within them. The metastable performance of arbiters has not been previously considered in published work on arbiters. The results on metastability performance apply to a class of digital circuits susceptible to metastable behaviour which includes arbiters as a special case. In summary, the main contributions of this thesis are as follows :

(i) A new class of arbiters referred to as batched arbiters is identified. Examples of batched arbiter implementations have been presented from which general timing models have been developed. The batching concept can be applied to any arbitration disciplines to generate a new derived batched discipline. The thesis demonstrates a simple method of implementing the batched version of a discipline by employing a wired-or common line to facilitate a lock-out mechanism on new requests. The technique is particularly suitable for distributed designs.

(ii) New models for practical arbiters have been developed. These models incorporate the effects of circuit delays and other non ideal characteristics, through the inclusion of inter-batch and

inter-service time durations. The models have been shown to apply to several practical arbiter circuits, including centralised, decentralised, asynchronous and clocked arbiters.

(iii) The service and utilisation performance of arbiters have been analysed with new analytical tools developed in the thesis. Imbedded Markov chains form the basis of the analytical techniques which permit arbitrary service time distribution and allow diverse applications to be accurately modelled. The analysis of batched and non batched fixed priority disciplines has been performed. The thesis has also indicated methods for extending this analysis to more general disciplines, such as dynamic priority, by adding further dimensions to the state definition. The analysis gives insight into behaviour which, in hindsight, can be explained in physical terms. The analysis technique has been shown to produce interesting limiting results under light and heavy request loadings. These results have been verified from computer studies of the Markov analysis and also independently via Monte-Carlo simulation. Service performance results of practical interest have been presented which include mean waiting times, which is a measure of response time, and proportion of time allocated to each requester, which is a measure of throughput. It has been observed that the properties of these measures is quite different when viewed from various "fairness" aspects. The ultimate limitation in the Markov analysis is the exponential re-request time distribution assumption. Also, the exponential increase in number of states with the number of requesters is a practical computational constraint.

(iv) Fundamental issues concerning metastability in digital systems, and arbiters in particular, have been examined. Metastable reliability of arbiters has been highlighted as an area in need of careful attention and design consideration. The thesis has contributed in the following areas of the study of metastability :

(a) The aperture model has been developed for metastable reliability evaluation. The model has been justified in theoretical terms and from published experimental results on many commonly employed flip-flops, latches and synchronising elements.

(b) Several schemes aimed at improving metastable reliability have been evaluated in this thesis. The use of pausable clock schemes is questionable due to the unreliability of metastable detectors and resulting reduced noise margins. Synchronisers employing Schmitt triggers have been shown to perform less reliably, mainly due to the critical loss in allowed settling of the circuit compared to a simple synchroniser. Synchronisers designed for fast resolution to valid logic states have been concluded to offer an improvement, providing rigorous testing and verification is performed. Metastable reliability is not improved, however, when the remaining logic is also upgraded in speed with the same fast devices, as may occur in future technologies. Should a fundamental limit in logic speed be reached, other techniques need to be considered. The most reliable scheme at present may be to employ a simple synchroniser with adequate provision for settling time.

(c) Previous results of Marino, Hurtado and Elliott [H.4, M.4] on the unavoidability of metastable behaviour have been extended in Theorem 3.1 of this thesis. The classification of the sets of possible input functions that give rise to metastability has been generalised to apply to practical digital signals. Theorem 3.1 of the thesis states that provided the set of possible input functions is connected and contains inputs which drive the system to two stable states, then metastability is unavoidable. Examples have been presented to illustrate the theorem. For example, it has been shown that a two input arbiter with asynchronous requests restricted to certain minimum rise and fall times cannot avoid metastable behaviour regardless of the physical realisation of the arbiter. These results on unavoidability of metastable behaviour have an important impact on digital design.

(d) It has been proved, using the aperture model for a basic synchroniser and reasonable assumptions on combination logic, that redundancy and masking techniques are ineffective in improving the metastable reliability of synchronisers. It is well known that it is possible to improve reliability with respect to component failure by employing the same redundancy and masking techniques. The result establishes that metastable failure and component failure *cannot* be treated in an equivalent manner in terms of their characteristics nor techniques of analysis. For example, metastable behaviour cannot be treated as a transient component failure, even though the two may appear, on first inspection, to be similar.

(v) The metastable reliability performance of arbiters has been analysed for all disciplines treated in the thesis. Both asynchronous and clocked implementations have been examined. From a circuit design viewpoint, the critical parameters that affect the metastable reliability have been identified as the allowed settling time within the circuit, and time constant of the basic synchronising element employed. It has been observed when examining resetting strategies for clocked arbiters that careful examination of all possible synchronising modes is necessary, due to the presence of subtle and obscure failure possibilities.

It has been noted that asynchronous arbiter designs can offer metastable reliability advantages over clocked designs. The clocked designs examined in Chapter 7 are susceptible to metastable failure when request inputs are reset after a service. This problem has been found to be absent from corresponding asynchronous arbiter designs presented in this thesis.

Monte-Carlo and Markov analysis techniques have been employed to examine a normalised metastable failure rate for the arbitration disciplines under certain reasonable assumptions. The results have shown that all disciplines, except the poorly performing fixed priority and random priority schemes, have a peak failure rate that occurs at the onset of request saturation of the arbiter. The peak failure rate is associated with maximum contention of requests for the resource.

At light request loadings, a fundamental normalised metastable failure rate has been derived which is identical for all disciplines examined in the thesis. This NMFR is proportional to the request rate squared and the number of requesters squared.

It has been found that the first come first served discipline has the lowest NMFR of all the disciplines discussed in the thesis. However, this discipline is difficult to implement. Other disciplines more amenable to implementation and with good NMFR performance have been found to be batched least recently used, and batched next robin.

It has been noted that batched disciplines perform better than the primary non batched counterparts on a NMFR basis.

(vi) Results comparing the service performance of all the disciplines have been presented in the thesis. The results of the standard deviation of waiting times indicate that batched diciplines usually have a lower standard deviation of waiting times compared with their primary non batched counterparts. The first come first served discipline has the lowest STDW; followed by batched LRU; LRU and next robin and batched next robin (equal); batched reverse robin; batched forward robin and batched fixed priority; random priority and; finally, fixed priority.

(vii) The effect of increasing inter-batch and inter-service delays has been shown to degrade service performance, and improve the NMFR. Increasing inter-batch and inter-service times has been recommended as a means to improve NMFR performance only when the allowed metastable settling time is directly increased.

9.2 SUGGESTIONS FOR FURTHER RESEARCH

There are several interesting extensions arising from the research carried out in the thesis :

(i) In the Markov analayis of the batched and non batched arbiters request rates are not assumed to be equal. In order to limit the large number of possible combinations of parameters in presenting the numerical results, balanced request loading has been assumed. While this assumption is representative of many systems of interest, relaxing the balanced request rate assumption may be necessary to model unbalanced requesters. The effects of biasing request rates would be of interest. For example, it is quite conceivable that fixed priority may perform better under these conditions.

(ii) In studying the NMFR of arbiters, it has been assumed that all flip-flops have identical metastable characteristics. This assumption is useful in examining a symmetric arbiter and also because it simplifies the analysis considerably. In practice, some variations will occur between flip-flop characteristics and it would be of interest to examine the sensitivity of the NMFR to such changes for various implementations. One could optimise the performance by permuting a given set of flip-flops if such results were known.

(iii) It has been suggested in this thesis that the Markov analysis can be extended to any non batched and batched disciplines with Markov state representations. This has been left for future work. Suggestions as to an appropriate state representation have been made in Chapter 4.

(iv) More complex performance measures may be able to be derived theoretically and their properties studied. For example, in the thesis standard deviation of waiting times results are generated by Monte-Carlo simulation. A theoretical derivation may reveal further results.

(v) The modelling of re-request time distributions has been restricted to exponential distributions. The exponential assumption successfully models many requesters, however, in practice the re-request distribution may be quite complex and dependent of the particular application. Although it is difficult to relax this assumption, simulation studies may be employed to estimate the effect of more accurate modelling.

(vi) The impact of arbiter performance on global system performance measures, such as completion time of a process or system response time to users, is an area for future research. Existing research, for example in [C.11, G.4, H.2, M.5], on system performance analysis do not examine the effect of differing arbitration disciplines.

(vii) Further research is suggested in the area of metastable modelling. For example, the effect of cascading flip-flops on metastable performance is not clear. Also, the consequences of undefined inputs, due to metastable behaviour of the driving devices, on various logic elements is not understood nor widely addressed in the literature. It has been reported that mysterious system crashes are possible [C.8, C.9], but few, if any, well documented case studies exist.

(viii) Scope for further research exists in the area of hardware correctness proving for asynchronous circuits. Some attempts have been made to develop automated theorem provers [S.3, S.4, W.4], but the use of these theorem provers is only feasible when extremely high reliability is required such as in life support systems or flight control computers [W.4]. The process of correctness proving is extremely tedious, and is not often employed by practical designers.

(ix) The problem of reducing metastable failure probability is an ongoing research problem. The use of metastable detectors needs further research in order to establish reliable operation under all conceivable electrical conditions, such as capacitive loading. Other means, such as the detection of timing conditions on the inputs and extending settling the time, may yield improvements.

APPENDIX A

PROOF AND COMMENTS ON THEOREM 3.1

The proof and discussion of Theorem 3.1 are based on the following theory from [M.4]: (The reader is encouraged to review the mathematical introduction in Section 3.4.1 before tackling the formal theory below.)

Theorem 1 If L is stable for input range C and $\bar{u} \in U_c$, then $A(L,\bar{u})$ is an open set.

Proof Since L is stable for C, there exists $r > 0$ such that $S_r(L) \subset A(L,C)$. Suppose $p \in A(L,\bar{u})$. Then there exists $t \in R^+$, such that $\varphi(p,\bar{u},t) \in L$. By continuity of φ, there exists $\delta > 0$ such that $|p - q| < \delta \Rightarrow |\varphi(p,\bar{u},t) - \varphi(q,\bar{u},t)| < r$. Hence, $\varphi(q,\bar{u},t) \in A(L,C)$ and so there exists $s > 0$, such that $\varphi(\varphi(q,\bar{u},t)\bar{u}_t,s) \in L$. Hence, $\varphi(q,\bar{u},t+s) \in L$ and so $q \in A(L,u)$. ▒

Theorem 2 $RID(L_0,L_1,\bar{u})$ is non empty. Furthermore, if $p \in RID(L_0,L_1,\bar{u})$, then $\varphi(p,\bar{u},t) \notin A(L_0,C) \cup A(L_1,C)$ for any $t \in R^+$.

Proof By Theorem 1, $A(L_0,\bar{u})$ and $A(L_1,\bar{u})$ are open subsets of Σ. Since L_0 and L_1 are disjoint, $A(L_0,\bar{u})$ and $A(L_1,\bar{u})$ are disjoint as well. Since Σ is connected, it cannot be the union of two non empty disjoint open sets. Hence, $RID(L_0,L_1,\bar{u})$ is non empty.

Now, suppose $\varphi(p,\bar{u},t) \in A(L_i,C)$ for some $p \in \Sigma$, $t \in R^+$, and $i = 0$ or 1. Then, since $\bar{u}_t \in U_c$, there exists $s \in R^+$, such that $\varphi(\varphi(p,\bar{u},t),\bar{u}_t,s) \in L_i$. Hence, $\varphi(p,\bar{u},t+s) \in L_i$ and so $p \notin RID(L_0,L_1,\bar{u})$.

▒

The following lemma is useful in establishing that any state trajectory that connects $A(L_0,\bar{u})$ and $A(L_1,\bar{u})$ must intersect $RID(L_0,L_1,\bar{u})$ for any $\bar{u} \in U_c$.

Lemma Suppose X is a connected metric space, Y is a metric space, G is a subset of Y, and $f:X \to Y$ is a continuous function, such that $f(X) \cap \text{int}(G) \neq \phi$ and $f(X) \cap \text{int}(G') \neq \phi$. Then, $f(X) \cap \text{bnd}(G) \neq \phi$.

Proof Suppose $f(X) \cap \text{bnd}(G) = \phi$. It will be shown that this leads to a contradiction. For any metric space Y, $Y = \text{int}(G) \cup \text{int}(G') \cup \text{bnd}(G)$. Since $f(X) \cap \text{bnd}(G) = \phi$, $f(X) \subset \text{int}(G) \cup \text{int}(G')$. Since the interior of a set is open, both int(G) and int(G') are open, and by hypothesis, each of these has a non empty intersection with $f(X)$. Hence, $f(X)$ is not connected. But, since $f(X)$ is the continuous image of a connected set, $f(X)$ is connected, a contradiction. Hence, $f(X) \cap \text{bnd}(G) \neq \phi$. ▨

Theorem 3.1 Suppose $\bar{u} \in U_c$ and $p \in A(L_o, \bar{u})$. Let $\Gamma \subset U$ be a *connected* set of input functions, $u:R^+ \to I$, with the following properties:

There exists $t_1 \in R^+$, $\hat{u}_1, \hat{u}_2 \in \Gamma$ such that

(i) $\phi(p, \hat{u}_1, t_1) \in A(L_o, \bar{u})$

(ii) $\phi(p, \hat{u}_2, t_1) \notin A(L_o, \bar{u})$

(iii) for all $u \in \Gamma$, $u_{t_1} = \bar{u}$

Also, suppose ϕ is continuous with respect to u on Γ, then, there exists $u^* \in \Gamma$ such that $\phi(p, u^*, t_1) \in RID(L_o, L_1, \bar{u})$. Furthermore, for any $T > 0$, there exists $\epsilon > 0$ such that for $0 \leq t \leq T$

$$|u^* - u| < \epsilon \Rightarrow (p, u, t_1 + t) \notin L_0 \cup L_1$$

Proof Consider the function $f: \Gamma \to \Sigma$ defined by $f(u) \triangleq \phi(p, u, t_1)$. By the hypotheses of the theorem f is continuous and $f(\hat{u}_1) \in A(L_o, \bar{u})$; and $f(\hat{u}_2) \notin A(L_o, \bar{u})$. It is now possible to apply the lemma above.

Applying this lemma with $X = \Gamma$, $Y = \Sigma$ and $G = A(L_o, \bar{u})$, there exist $u^* \in \Gamma$ such that $f(u^*) = \phi(p, u^*, t_1) \in \text{bnd}(A(L_o, \bar{u}))$. Since $A(L_o, \bar{u})$ and $A(L_1, \bar{u})$ are both open (see Theorem 1), then $\text{bnd}(A(L_o, \bar{u})) \cap A(L_o, \bar{u}) = \phi$ and $\text{bnd}(A(L_o, \bar{u}) = \phi$. Thus, $\text{bnd}(A(L_o, \bar{u})) \subset \text{RID}(L_o, L_1, \bar{u})$, giving $\phi(p, u^*, t_1) \in \text{RID}(L_o, l_1, \bar{u})$.

The remainder of the proof follows the last part of the proof of theorem 4 in [M.4] and is given here for completeness. It relies on Theorem 2 above.

For $i = 0$ or 1 there exists $r_i > 0$ such that

$$\phi\left[\phi(p, u^*, t_1), \bar{u}, T\right] \notin \bigcup_{\ell_i \in L_i} S_{r_i}(\ell_i)$$

Since ϕ is continuous with respect to initial state there exists $\delta_i > 0$ such that

$$|y - \phi(p, u^*, t_1)| < \delta_i \quad \Rightarrow \quad |\phi(y, \bar{u}, T) - \phi(\phi(p, u^*, t_1), \bar{u}, T)| < r_i$$

And since ϕ is continuous with respect to u, there exists $\epsilon_i > 0$ such that

$$|u - u^*| < \epsilon_i \Rightarrow |\phi(p, u, t_1) - \phi(p, u^*, t_1)| < \delta_i$$

Combining these gives

$$|u - u^*| < \epsilon_i \Rightarrow \phi(p, u, t_1 + t) \notin L_i \quad \text{for any} \quad t \in [0, T].$$

Letting $\epsilon = \min(\epsilon_o, \epsilon_1)$

$$|u - u^*|_< \epsilon \Rightarrow \phi(p, u, t_1 + t) \notin L_o \cup L_1 \quad \text{for } t \in [0,T]$$

Comments on Theorem 3.1

Theorem 4 in [M.4] is a special case of Theorem 3.1, where the set of input function with bounded first derivatives replaces the connected set of input functions. However, condition (i) is added in Theorem 3.1, with justification below:

Marino employs the set of input motions

$$U(b, e, t_1, \bar{u}) \triangleq \{u: R^+ \to I \,|\, u \text{ is differentiable on } R^+ \text{ with } |\dot{u}(t)| < b \text{ for all } t \in R^+, u_{t_1} = \bar{u}, \text{ and } u(0) = e\}$$

In Marino's proof of Theorem 4 [M.4], he asserts that $U_c \cap U(b, e, t_1, \bar{u}) \neq \phi$. This statement is not always true when C is not path connected. For example, if $C = [0,1] \cup [2,3]$, $e = 0$, $\bar{u}(t) = 3$ for all $t \in R^+$, $t_1 = 1$ and $b = 10$, $u \notin U_c$ for all $u \in U(b, e, t_1, \bar{u})$. This follows because since u is differentiable it must be continuous and $u(0) = 0$ and $u(1) = 3$ implies there exists t such that $u(t) \notin C$.

Moreover, Marino then goes on to deduce from $U_c \cap U(b, e, t_1, u^-) \neq \phi$ that there exists $u \in U(b, e, t_1, \bar{u})$ such that $\phi(p, u, t_1) \in L_o \subset A(L_o, \bar{u})$. This is invalid because $U(b, e, t_1, \bar{u})$ may not include $\bar{u}$, and, further t_1 may not be large enough so that $\phi(p, u, t_1) \in L_o$.

Thus, the statement "$\phi(p, u, t_1) \in A(L_o, \bar{u})$ for some $u \in U(b, e, t_1, \bar{u})$" needs to be included as a hypothesis in Marino's theorem. The significance of the theorem is not altered by this change, only the rigour restored.

APPENDIX B

DERIVATION OF EQUATION (5.9)

Equation (5.9) follows from (5.8). In order to evaluate the first probability term in (5.8) consider

$$\text{prob}\left[\begin{array}{l|l}\text{h requests first during the} & \text{at least one requester} \\ \text{time interval } [0,T] & \text{requests during } [0,T]\end{array}\right]$$

$$= \sum_{a=1}^{\left\lceil \frac{T}{\Delta t} \right\rceil} \frac{\text{prob}\left[\begin{array}{l}\text{h requests during } [(a-1)\Delta t,\ a\Delta t] \\ \text{and no other requests during } [0,\ a\Delta t]\end{array}\right]}{\text{prob}(\text{ at least one requester requests during } [0,T])}$$

$$= \lim_{\Delta t \to 0} \sum_{a=1}^{\left\lceil \frac{T}{\Delta t} \right\rceil} \left[\prod_{g=1}^{k} e^{-\lambda_g a\Delta t}\right] \left[\frac{\lambda_h \Delta t}{1 - \prod_{g=1}^{k} e^{-\lambda_g t}}\right]$$

$$= \frac{\int_0^T \lambda_h e^{-\sum_{g=1}^{k} \lambda_g t}\, dt}{1 - e^{-\sum_{g=1}^{k} \lambda_g T}}$$

$$= \frac{\lambda_h}{\sum_{g=1}^{k} \lambda_g} \qquad \text{(B.1)}$$

Note that (B.1) is independent of T, thus (B.1) applies as T approaches infinity. The remaining terms of (5.8) follow from observing that $e^{-\lambda_g D_3}$ is the probability that requester h makes no request during D_3.

APPENDIX C

Derivation of Equation (5.11)

The first probability in (5.10) on the R.H.S. is found by observing:

$$\text{prob}\left[\ \text{requesters} \in i \quad \text{request in} \quad B_{n-1} \quad \middle|\ S(n-1) = j\ \right]$$

$$= \text{prob}\left[\begin{array}{l}\text{requesters} \in i \quad \text{request in} \quad B_{n-1} \\ \text{and at least one of which before} \quad D_3\end{array}\ \middle|\ S(n-1) = j\ \right]$$

$$+ \text{prob}\left[\begin{array}{l}\text{requesters} \in i \quad \text{request during} \\ D_3 \quad \text{of} \quad B_{n-1}\end{array}\ \middle|\ S(n-1) = j\ \right] \qquad \text{(C.1)}$$

where

$$B_{n-1} = D_2 + \sum_{\ell \in i} t_s(\ell) + D_3$$

B_{n-1} corresponds to the time duration of the $(n-1)^{th}$ batch, if and only if at least one request occurs before D_3. Rearranging (C.1) and using (5.5) gives

$$\text{prob}\left[\begin{array}{l}\text{requesters} \in i \quad \text{request in} \quad B_{n-1} \quad \text{and} \\ \text{at least one of which before} \quad D_3\end{array}\ \middle|\ S(n-1) = j\ \right]$$

$$= \prod_{h \in i}\left[1 - Q(j,h)e^{-\lambda_h D_3}\right] - \prod_{h \in i}\left[Q(j,h)\left[1 - e^{-\lambda_h D_3}\right]\right] \qquad \text{(C.2)}$$

Equation (5.11) now follows from (C.2) and applying (5.5).

APPENDIX D

PROOFS OF THEOREMS 5.1 AND 5.2

Theorem 5.1 The probability transition matrix of the fixed priority batch arbiter model, P, has a unique positive limiting probability vector independent of the initial probability vector, provided $\mu_h, \lambda_h > 0$ for $h = 1, .., k$ and $D_1 + D_3 > 0$.

Proof As mentioned in conjunction with (5.17) and Figure 5.6, the proof of Theorem 5.1 reduces to choosing an intermediate state w such that $p_{iw}p_{wj} > 0$. Consider three cases for $D_1 + D_3 > 0$:

Case 1: $D_1 > 0$, $D_3 > 0$

Equations (5.6), (5.7), (5.9) and (5.11) imply that $p_{xy} > 0$ for all states x and y except $x = y = 0$. Choose $w = 1$, arbitrarily, which gives $p_{i1}p_{1j} > 0$ for all $i,j = 0, \ldots, m-1$. So by (5.17) $P^2 > 0$.

Case 2: $D_1 = 0$, $D_3 > 0$

$p_{xy} > 0$ for all states x and y with the following exceptions:

(i) $x = y = 0$.

(ii) The state x contains only one requester, h, say and the lowest priority requester in y is also h.

Exception (ii) follows because h has no time in which to request at the end of the batch with state y, since $D_1 = 0$ and no other request is pending (since the following batch, x, contains only h) hence D_3 does not occur. Exceptions (i) and (ii) are the only transitions with zero probability. This can be established by considering the following classes of transitions:

(a) $x \neq 0$ and $y = 0$. See Figure D.1

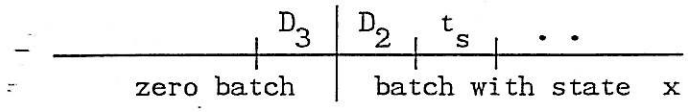


FIGURE D.1 Zero to Non Zero Batch Transition.

Since $D_3 > 0$, any requester may request after the first request during the zero batch. Thus, $p_{x0} > 0$ for all $x \neq 0$.

(b) $x \neq 0$ and $y \neq 0$.

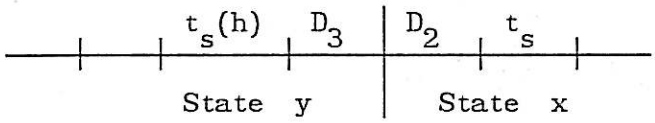


FIGURE D.2 Non Zero to Non Zero Batch Transition with $D_1 = 0$.

In Figure D.2, h is shown as the last requester in state y. Any requester, other than h, has time to request before the end of state y. Thus, state i can be any state made up of requesters other than h. If state x contains a requester besides h, ℓ say, then it is possible for h to request in order to be in state x because D_3 occurs due to ℓ requesting before the end of $t_s(h)$. Thus, transitions from y to x have non zero probabilities when h is not in x, or when h is in x and there is at least one other requester in x.

(c) $x = 0$ $y \neq 0$

All transitions from y to the zero state have non zero probabilities because it is possible for every requester not to request during a non zero batch.

Now that it has been established which transitions have non zero probabilities, the intermediate state w can be chosen as follows: For i not a singleton state choose $w = m-1$ (i.e. the full batch). For i a singleton state, choose $w = m-i$ (the full batch without the singleton requester). It follows from the above that $p_{iw}p_{wj} > 0$ and hence from (5.17) $P^2 > 0$.

Case 3 $D_1 > 0$ and $D_3 = 0$

Since D_1 is present at the end of all non zero batches, $p_{xy} > 0$ for all x and $y \neq 0$. However, for $y = 0$, x must be a singleton state in order that $p_{xy} > 0$. This is because the probability of at least two requesters requesting at exactly the same time is zero. Hence, there is a unique first request to terminate a zero batch and, thus, x cannot contain more than one requester.

Now it is shown that there exists a state w such that $p_{iw}p_{wj} > 0$: Choose $w = 1$ (singleton batch with requester 1) then $p_{i1}p_{1j} > 0$ for all i,j and hence from (5.17) $P^2 > 0$.

This completes all possible cases of $D_1 + D_3 > 0$. Since P is irreducible and primitive, the Markov chain is acyclic. Theorem 5.1 now follows from the Ergodic Theorem for homogeneous Markov chains [G.1, p. 95]:

> In a homogeneous Markov chain limiting absolute probabilities exists for arbitrary initial probabilities and are independent of them if and only if the chain is acyclic. ▨

Theorem 5.2 The probability transition matrix P_1 as defined in (5.18) has a unique positive limiting probability vector independent of the initial probability vector provided $\mu_h, > 0$ for $h = 1, \ldots, k$ and $D_1 = D_3 = 0$.

Proof The proof technique is the same as the Theorem 5.1 proof. There are two cases:

Case 1: $k > 2$

It will be shown that $P_1^3 > 0$

$$p_{ij}^{(3)} = \sum_{w=0}^{m-2} p_{iw}^{(2)} p_{wj}$$

$$= \sum_{w=0}^{m-2} \sum_{z=0}^{m-2} p_{iz} p_{zw} p_{wj}$$

$$\geq p_{iu} p_{ux} p_{xj} \qquad 0 \leq u, x \leq m-2 \qquad \text{(D.1)}$$

Hence from (D.1) it is sufficient to find states u and x so that $p_{iu} p_{ux} p_{xj} > 0$. Firstly, the transitions which have zero probability for $D_1 = D_3 = 0$ are examined:

p_{xy} is zero only in the following cases :

(i) $y = 0$ and x is not a singleton state.

(ii) x contains the lowest priority requester in y.

Case (i) follows from the argument used in Case 3 of Theorem 1. Case (ii) follows from the fact that with $D_1 = D_3 = 0$ there is zero time available at the end of a batch for the lowest priority requester of the batch to make a request before the next batch.

The states u and x, illustrated in Figure D.3, are now chosen to avoid zero probability transitions.

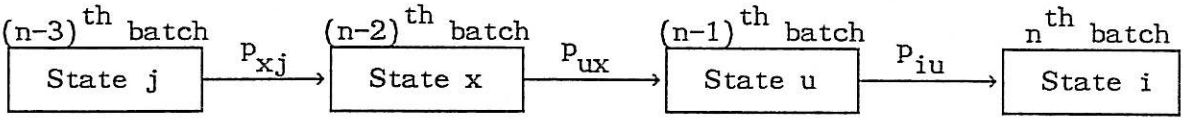


FIGURE D.3 State Transitions in Proof.

Since state i cannot contain all requesters, there is a requester, ℓ, say which is absent from i. Let u be the singleton state containing ℓ. If $j = 0$ then let x be the singleton state containing a requester different from ℓ. If $j \neq 0$ and g is the lowest priority requester in j then choose x to be the singleton state containing a requester different from g and ℓ ($k > 2$). With this choice (see Figure D.4) it can be seen that $p_{iu}p_{ux}p_{xj} > 0$.

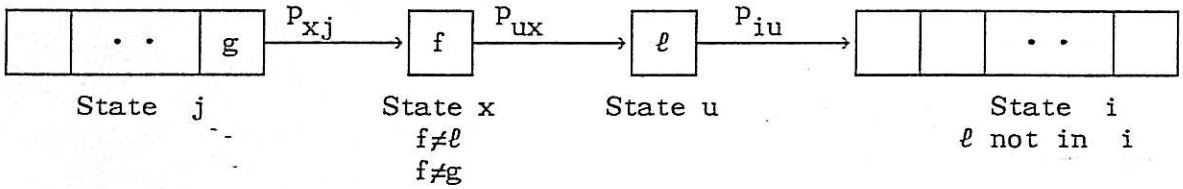


FIGURE D.4 Non Zero Batch Transitions of Proof.

Case 2: $k = 2$

It will be shown that $P_1^2 > 0$. For P_1^2 :

$$p_{ij}^{(2)} = \sum_{u=0}^{m-2} p_{iu}p_{uj}$$

$$\geq p_{iw}p_{wj} \qquad 0 \leq w \leq m-2 \tag{D.2}$$

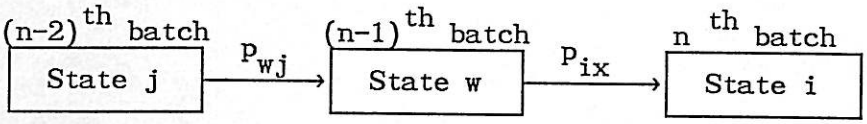


FIGURE D.5 State Transitions Illustrating Proof.

Referring to Figure D.5, since $k = 2$, w can only be chosen from 3 states (0, 1 or 2). If w is chosen to be a state different from both i and j then $p_{iw}p_{wj} > 0$ as then none of the zero probability transitions discussed in Case 1 can occur. Hence $P_1^2 > 0$ from (D.2).

The remainder of the proof is the same as the proof of Theorem 5.1.

▨

APPENDIX E

DERIVATION OF THE CONDITIONAL MEAN WAITING TIME FOR THE FIXED PRIORITY BATCHED ARBITER MODEL

In order to evaluate the waiting time $W(i,j,h)$ of (5.46), it is necessary to define a conditional mean waiting time, $cmwt(j,h)$.

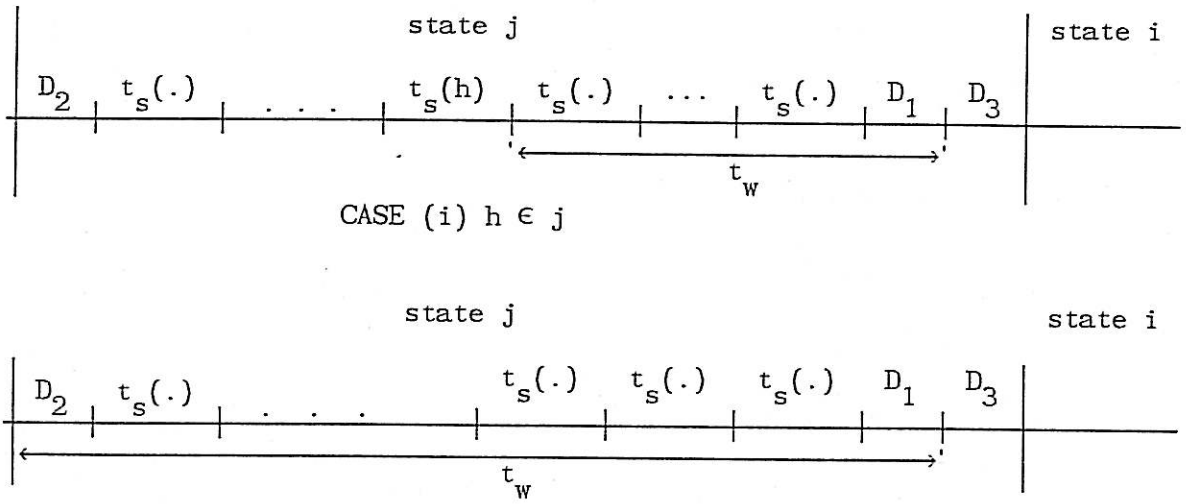


FIGURE E.1 Illustration of Definition t_w.

Firstly, t_w is defined, as shown in Figure E.1, by

$$t_w \triangleq \begin{cases} D_1 + \sum\limits_{\substack{g \in j \\ g > h}} t_s(g) & , h \in j \\ \\ D_1 + D_2 + \sum\limits_{g \in j} t_s(g) & , h \notin i \end{cases} \qquad \text{(E.1)}$$

Now the conditional mean waiting time, cmwt(j,h), is defined as the mean time from requester h requesting to the end of t_w, given that requester h requests during t_w. The time duration t_w is divided into two times: t_i, the "idle" duration when Req h = 0; and t_r, the "request" duration when Req h = 1, as shown in Figure E.2. That is

$$t_w = t_i + t_r \tag{E.2}$$

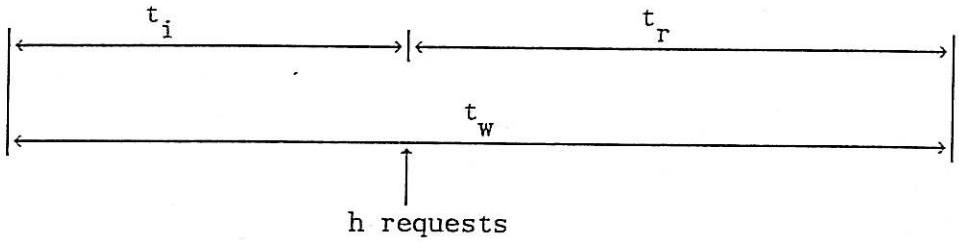


FIGURE E.2 Subdivision of t_w.

It follows that

$$\text{cmwt}(j, h) = \int_0^\infty t_r \, f_r(t_r) \, dt_r \int_0^\infty f_w(t_w) \int_0^\infty t_r \, f_{r|w}(t_r, t_w) \, dt_r \, dt_w \tag{E.3}$$

where f_r is the density function of t_r, $f_{r|w}$ is the conditional density function of t_r given t_w, and f_w is the density function of t_w. Let $f_{i|w}$ be the condition density function of t_i given t_w and, then from (E.2) and the exponential re-request time distribution

$$= \begin{cases} \dfrac{e^{-\lambda_h(t_w - t_r)}}{1 - e^{-\lambda_h t_w}} & , \ 0 \leq t_r \leq t_w \\ 0 & , \ t_r > t_w \text{ or } t_r < 0 \end{cases} \qquad \text{(E.4)}$$

Substituting (E.4) in (E.3) and integrating gives

$$\text{cmwt}(j,h) = \int_0^\infty f_w(t_w) \left\{ \frac{t_w}{1-e^{-\lambda_h t_w}} - \frac{1}{\lambda_h} \right\} dt_w \qquad \text{(E.5)}$$

For later use, the conditional request time, $\text{crt}(t_w,h)$ is defined by

$$\text{crt}(t_w,h) = \int_0^\infty t_r \, f_{r|w}(t_r,t_w) \, dt_r$$

$$= \frac{t_w}{1-e^{-\lambda_h t_w}} - \frac{1}{\lambda_h} \qquad \text{(E.6)}$$

and represents the mean time requester h has a request pending given that requester h requests during t_w.

Examples

1. For constant service times, t_w is constant and

$$\text{cmwt}(j,h) = \text{crt}(t_w,h) \qquad \text{(E.7)}$$

2. For exponentially service times, f_w is the density function of a

2. For exponentially service times, f_w is the density function of a sum of exponential random variables and a constant. If all mean service times are equal to $\frac{1}{\mu}$, f_w is given by

$$f_w(t) = \begin{cases} \frac{\mu^r (t-D)^{r-1}}{(r-1)!} e^{-\frac{t-D}{\mu}} & , \quad t \geq D \\ 0 & , \quad t < D \end{cases} \qquad \text{(E.8)}$$

where

$$D = \begin{cases} D_1 & , \quad h \in j \\ D_1 + D_2 & , \quad h \notin j \end{cases}$$

$$r = \sum_{\substack{g \in j \\ g > h}} 1$$

Equation (E.5) can be evaluated using series expansions to give

$$cmwt(j,h) = -\frac{1}{\lambda_h} + \frac{r}{\mu} \sum_{n=0}^{\infty} \frac{e^{-n\lambda_h D}}{\left[1+n\frac{\lambda_h}{\mu}\right]^{r+1}} + D \sum_{n=0}^{\infty} \frac{e^{-n\lambda_h D}}{\left[1+n\frac{\lambda_h}{\mu}\right]^{r}} \qquad \text{(E.9)}$$

It can be shown by applying standard convergent tests that (E.9) is convergent for $r \geq 0$, $\mu \geq 0$, $D \geq 0$ and $\lambda_h \geq 0$. For increasing r, (E.9) tends to $crt(D + \frac{r}{\mu}, h)$, due to f_w approaching a delta function centred on the mean $D + \frac{r}{\mu}$. In general, this is true for any service time distribution as can be seen from applying the central limit theorem of statistics.

In order that (E.5) can be applied to find $W(i,j,h)$, it is necessary to find the conditional probability, p_a, that requester h

requests before D_3 (i.e. during t_w), given that state i follows state j.

$$p_a \triangleq \text{prob}[h \text{ requests before } D_3 \mid S(n) = i,\ S(n-1) = j]$$

$$= \frac{\text{prob}[h \text{ requests before } D_3 \text{ and } S(n) = i \mid S(n-1) = j]}{\text{prob}[S(n) = i \mid S(n-1) = j]}$$

$$= \frac{1}{p_{ij}} [1 - Q(j,h)] \cdot \prod_{\substack{r \in i \\ r \neq h}} \left[1 - Q(j,r)e^{-\lambda_r D_3}\right] \prod_{r \notin i} Q(j,r)e^{-\lambda_r D_3} \qquad (E.10)$$

Then

$$W(i,j,h) = p_a\{\text{cmwt}(j,h) + D_3\} + (1 - p_a)\ \text{crt}(D_3,h) + M(i,j) \qquad (E.11)$$

where $\text{crt}(D_3,h)$ is defined in (E.6) and

$$M(i,h) \triangleq D_2 + \sum_{\substack{g \in i \\ g < h}} \frac{1}{\mu_g}, \quad j \neq 0 \qquad (E.12)$$

Note that $M(i,h)$ is the waiting after state j up until requester h is actually serviced in state i, as shown in Figure E.3.

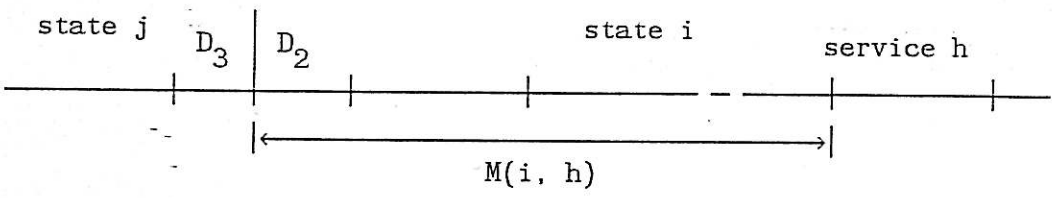


FIGURE E.3 Definition of M(i, h).

Equation (E.11) holds only for $j \neq 0$. When $j = 0$, it follows from Appendix B that

$$W(i,0,h) = \frac{D_3 \ \lambda_h}{\sum_{g \in i} \lambda_g} + crt(D_3,h) \cdot \left(1 - \frac{\lambda_h}{\sum_{g \in i} \lambda_g}\right) + M(i,h) \qquad (E.13)$$

where $\frac{\lambda_h}{\sum_{g \in i} \lambda_g}$ is the probability h requests first during an idle period, as derived in Appendix B.

The expressions above for the mean waiting time have been realised in a PASCAL program and tested against Monte-Carlo simulations. Agreement within statistical variation has been consistently obtained over a wide range of parameters D_1, D_2, D_3, λ, μ and requester h.

APPENDIX F

PROOFS OF CONVERGENCE OF EQUATIONS (5.56), (5.61) and (5.66)

This appendix provides justification for the limits obtained for light and heavy loading performance of the arbiter in equations (5.56), (5.66) and (5.66). Firstly it is shown that:

$$\lim_{n\to\infty} p(n) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \qquad \text{when all } \lambda_h > 0 \qquad \text{(F.1)}$$

From Theorems 5.1 and 5.2, the limit $\lim_{n\to\infty} p(n) = p$ of (F.1) exists, and so:

For all $\epsilon > 0$ there exists an integer $N_0 > 0$

such that $n > N_0 \Rightarrow \|p(n) - p\| < \epsilon$ (F.2)

It will be shown that for all $\epsilon' > 0$ there exists an integer N_1 such that $N > N_1 \Rightarrow \left\| \frac{1}{N} \sum_{n=0}^{n-1} p(n) - p \right\| < \epsilon'$.

Consider the definition

$$K \triangleq \max_{\text{all } n} \{\| p(n) - p \|\} \qquad \text{(F.3)}$$

K is well defined, since let $\epsilon = \| p(0) - p \|$ in (F.2), then $K = \max_{n=1..N_0} \{\| p(n) - p \|\}$ since $\| p(n) - p \| < \| p(0) - p \|$ for $n > N_0$.

Given an $\epsilon' > 0$, N_1 is chosen as folllows: Putting $\epsilon = \frac{\epsilon'}{2}$ in (F.2), there exists N_0' such that $n > N_0' \Rightarrow \| p(n) - p \| < \frac{\epsilon'}{2}$.

Now

$$\left\| \left[\frac{1}{N} \sum_{n=0}^{N-1} p(n) \right] - p \right\|$$

$$\leq \frac{1}{N} \left\| \left[\sum_{n=0}^{N_0'} p(n) \right] - (N_0' + 1)p \right\| + \frac{1}{N} \left\| \left[\sum_{n=N_0'+1}^{N-1} p(n) \right] - (N - N_0' - 1)p \right\|$$

$$\leq \frac{1}{N} \sum_{n=0}^{N_0'} \| p(n) - p \| + \frac{1}{N} \sum_{n=N_0'+1}^{N-1} \| p(n) - p \|$$

$$< \frac{N_0'+1}{N} K + \frac{N-N_0'-1}{N} \frac{\epsilon'}{2} \qquad \text{(F.4)}$$

The L.H.S. of (F.4) is $< \epsilon'$ when

$$N > \left[N_0' + 1\right]\left[\frac{2K}{\epsilon'} - 1\right] \qquad \text{(F.5)}$$

so choose

$$N_1 = [(N_0' + 1)(\frac{2K}{\epsilon'} - 1)] + 1$$

where $[x]$ is the greatest integer less than x. Hence [F.1] has been established.

Proof of $$\lim_{\lambda\to 0}\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1}p(n)=\lim_{N\to\infty}\lim_{\lambda\to 0}\frac{1}{N}\sum_{n=0}^{N-1}p(n) \qquad (F.6)$$

To establish (F.6) it suffices to show that $\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1}p(n)$ converges uniformly in λ.

It will be shown that there exists a $\delta > 0$ such that $\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N=1}p(n)$ converges uniformly for $0 < \lambda < \delta$. That is, it will be shown that for all $\epsilon > 0$ there exists an integer $N_2 > 0$ independent of λ such that

$$N > N_2 \Rightarrow \left\| \frac{1}{N}\sum_{n=0}^{N=1}p(n) - p \right\| < \epsilon \qquad (F.7)$$

This then allows the order of the limits in (F.6) to be swapped [R.4 p.668].

Consider the eigenvalues of $P_0 \triangleq \lim_{\lambda\to 0} P$, where P_0 is given in (5.25). The eigenvalues of P are denoted by $e_1(\lambda), e_2(\lambda), \ldots, e_m(\lambda)$. The characteristic equation of P_0 is

$$(e^2 - 1)e^{m-2} = 0 \qquad (F.8)$$

Hence the eigenvalues of P_0 are given by

$$e_1(0) = 1,\ e_2(0) = -1,\ e_i(0) = 0 \quad i = 3, 4, \ldots, m \qquad (F.9)$$

For $\lambda > 0$, P is irreducible and primitive and hence, since P is also

stochastic, column sums equal one, it has one eigenvalue of 1 and all others have modulus less than 1. (See Frobenius theorem [G.1, p.53], noting that eigenvectors corresponding to eigenvalues different from 1 have a zero coordinate sum, hence cannot correspond to the maximal eigenvalue. Thus 1 is the maximal eigenvalue.)

The eigenvalues of P are continuous functions of the elements of the matrix and the elements of the matrix are continuous functions of λ for $\lambda \geq 0$. Hence there exists a $\delta > 0$ such that for $0 < \lambda < \delta$ (using (F.9))

$$e_1(\lambda) = 1$$

$$\mathrm{Re}\left[e_2(\lambda)\right] < -\frac{1}{2} \quad , \quad |\, e_2(\lambda)\,| < 1 \quad \text{(since unique max =1)}$$

and

$$|\, e_i(\lambda)\,| < \frac{1}{2} \qquad \text{for} \qquad i = 3, 2, \ldots, m \tag{F.10}$$

Note that e_2 is distinct from the other eigenvalues. Consider now the eigenvalues, $f_i(\lambda)$, of $\frac{1}{N}\sum_{n=0}^{N=1} P^n$

$$f_i(\lambda) = \frac{1}{N}\sum_{n=0}^{N=1} e_i(\lambda)^n \qquad i = 1, 2, \ldots, m$$

$$= \frac{1}{N}\left[\frac{1-e_i(\lambda)^N}{1-e_i(\lambda)}\right] \qquad , e_i(\lambda) \neq 1 \quad \text{(i.e. } i \neq 1\text{)} \tag{F.11}$$

Thus from (F.10) and (F.11) with $0 < \lambda < \delta$

$$f_1(\lambda) = 1$$

$$|f_2(\lambda)| \leq \frac{1}{N} \frac{1 + |e_i(\lambda)|^N}{1 - \mathrm{Re}(e_i)} < \frac{2}{N}$$

$$|f_i(\lambda)| \leq \frac{1}{N} \frac{1 + |e_i(\lambda)|^N}{1 - |e_i(\lambda)|} < \frac{4}{N} \quad , i = 3, 4, \ldots, m \qquad (F.12)$$

There exists an invertible matrix Q for each λ such that

$$P = Q\, J\, Q^{-1} \qquad (F.13)$$

where J is the Jordan canonical form for P.

$$J = \begin{bmatrix} 1 & & & & 0 \\ & e_2(\lambda) & & & \\ & & J_1 & & \\ & 0 & & \ddots & \\ & & & & J_\ell \end{bmatrix} \qquad (F.14)$$

where $J_1, \ldots, J_\ell$ are Jordan blocks of the following form:

$$J_i = \begin{bmatrix} e & & & 0 \\ 1 & e & \ddots & \\ & 1 & \ddots & \\ 0 & & 1 & e \end{bmatrix} \quad \text{where} \quad e \in \{e_3(\lambda), e_4(\lambda), \ldots, e_m(\lambda)\} \qquad (F.15)$$

Note that, since $e_2(\lambda)$ is distinct from the others, its block is 1×1, similarly with $e_1(\lambda) = 1$. Now

$$J^n = \begin{bmatrix} 1 & & & \\ & e_2^n(\lambda) & & 0 \\ & & J_1^n & \\ & 0 & & \ddots \\ & & & J_\ell^n \end{bmatrix} \tag{F.16}$$

where

$$J_i^n = \begin{bmatrix} e^n & 0 & & & & & \\ \binom{n}{1} e^{n-1} & e^n & \ddots & & & & \\ \binom{n}{2} e^{n-2} & \binom{n}{1} e^{n-1} & & & & 0 & \\ \vdots & \vdots & \ddots & & & & \\ \binom{n}{j} e^{n-j} & \binom{n}{j-1} e^{n-j+1} & \ddots & & & & \\ \vdots & & & \ddots & & & \\ \binom{n}{n} e^0 = 1 & & & & \ddots & & \\ & & & & & \ddots & \\ 0 & & & & & e^n & 0 \\ \vdots & & & & & \ddots & \\ 0 & & & & & \binom{n}{1} e^{n-1} & e^n \end{bmatrix} \tag{F.17}$$

where

$$\binom{n}{j} = \frac{n!}{(n-j)!\,j!}$$

Using $p(n) = P^n p(0)$

$$q(N) \triangleq \frac{1}{N} \sum_{n=0}^{N-1} p(n) = \frac{1}{N} \sum_{n=0}^{N-1} P^n \, p(0)$$

$$= Q\left[\frac{1}{N}\sum_{n=0}^{N-1} J^n\right]Q^{-1}p(0) \quad \text{from (F.13)} \tag{F.18}$$

Consider the elements of

$$\frac{1}{N}\sum_{n=0}^{N-1} J^n = \begin{bmatrix} 1 & & & 0 \\ & f_2(\lambda) & & \\ & & \frac{1}{N}\sum_{n=0}^{N-1} J_1^n & \\ & & & \ddots \\ 0 & & & \frac{1}{N}\sum_{n=0}^{N-1} J_\ell^n \end{bmatrix} \tag{F.19}$$

A typical element of $\frac{1}{N}\sum_{n=0}^{N-1} J_i^n$ in (F.19) looks like

$$\frac{1}{N}\sum_{n=0}^{N-1}\binom{n}{j}e^n \tag{F.20}$$

but

$$\left|\sum_{n=j}^{N-1}\binom{n}{j}e^n\right| \le \sum_{n=j}^{N-1}\binom{n}{j}|e|^n \le \sum_{n=j}^{\infty}\binom{n}{j}\left(\frac{1}{2}\right)^n \tag{F.21}$$

The series is absolutely convergent since the limit of the ratio of consecutive terms is less than one:

$$\lim_{n\to\infty}\frac{\binom{n+1}{j}\left(\frac{1}{2}\right)^{n+1}}{\binom{n}{j}\left(\frac{1}{2}\right)^{n}} = \lim_{n\to\infty}\frac{n+1}{n-j}\cdot\frac{1}{2} = \frac{1}{2} < 1 \tag{F.22}$$

Thus for every element, h_{ij}, in (F.19), except 1, there exists $B_{ij} \ge 0$

such that

$$\left| -h_{ij} \right| \leq \frac{B_{ij}}{N} \qquad \text{(F.23)}$$

Let $b_1, b_2, \ldots, b_m$ be the normalised basis vectors (i.e. $\| b_i \| = 1$ for $i = 1, \ldots, m$) of the basis for J in (F.14) (i.e. the columns of Q), then for each λ there is a unique set of coordinates $\alpha_1, \alpha_2, \ldots, \alpha_m$ such that

$$p(0) = \alpha_1 b_1 + \alpha_2 b_2 + \ldots + \alpha_m b_m \qquad \text{(F.24)}$$

From (F.18)

$$q(N) = \alpha_1 b_1 + f_2(\lambda)\, \alpha_2 b_2 + K_3 b_3 + \ldots + K_m b_m \qquad \text{(F.25)}$$

where

$$K_i = \sum_{j=1}^{m} h_{ij}\, \alpha_j$$

Now

$$\| q(N) - \alpha_1 b_1 \| \leq \left| f_2(\lambda) \right| \left| \alpha_2 \right| \cdot \| b_2 \| + \sum_{i=3}^{m} \left| K_i \right| \| b_i \|$$

$$< \sum_{i=1}^{m} \frac{B_i'}{N} \left| \alpha_i \right| \qquad \text{(F.26)}$$

(N.B. $\| b_i \| = 1$, $i = 1 \ldots m$)

where B_i' is made up of finite sums of B_{ij}, and hence is finite and independent of λ.

Equation (F.26) shows that $q(N)$ converges to $\alpha_1 b_1$, as $N \to \infty$. Since the cooredinates $\alpha_1, \ldots, \alpha_m$ are continuous functions λ (since Q is a continuous function of λ) and $\alpha_1, \ldots, \alpha_m$ are bounded at

$\lambda = 0$ and $\lambda = \delta$, then they are bounded for $0 < \lambda < \delta$. Hence, there exists a $B' < \infty$ independent of λ so that (F.26) becomes

$$\left\| q(N) - \alpha_1 b_1 \right\| < \frac{B'}{N} \tag{F.27}$$

Hence, $q(N)$ is uniformly convergent for λ in the open interval $(0,\delta)$, since for every $\epsilon > 0$ there exists $N_2 = \left[\frac{B'}{\epsilon}\right] + 1$ which is independent of λ, such that

$$N > N_2 \quad \Rightarrow \quad \left\| q(N) - \alpha_1 b_1 \right\| < \epsilon$$

Equation (6.30) is now proved. It is sufficient to show

$$\lim_{\lambda\to\infty} p = \lim_{\lambda\to\infty} \lim_{n\to\infty} p(n) = \lim_{n\to\infty} \lim_{\lambda\to\infty} p(n) \tag{F.28}$$

It will be shown that there exists a $T > 0$ such that $\lim_{\lambda\to\infty} p(n)$ is uniformly convergent for $\lambda > T$.

The eigenvalues, $e_1(\lambda) = 1, e_2(\lambda), e_3(\lambda), \ldots, e_m(\lambda)$ of P continuously approach the eigenvalues of $P_\infty(+,0)$ and $P_\infty(X,+)$[1] which are the same for both matrices and denoted by

$$e_1(\infty) = 1, \; e_2(\infty) = 0, \; e_3(\infty) = 0, \; \ldots, \; e_m(\infty) = 0 \tag{F.29}$$

Hence, by continuity with respect to λ, there exists $T > 0$ such that $\lambda > T$ gives

---

[1] See equation (5.31) for an explanation of this notation

$$e_1(\lambda) = 1$$
$$| e_i(\lambda) | < \frac{1}{2} \quad i = 2, 3, \ldots, m \tag{F.30}$$

The elements of J^n now have a typical non zero element

$$\begin{bmatrix} n \\ j \end{bmatrix} e^n \qquad , \; | e | < \frac{1}{2} \tag{F.31}$$

Thus, using a similar argument that was used to establish (F.26), for some $B_{ij}' > 0$ independent of λ

$$\| p(n) - \alpha_1 b_1 \| < \sum_{i=2}^{m} \sum_{j=1}^{m} \left[\frac{B_{ij}' \begin{bmatrix} n \\ j \end{bmatrix}}{2^n} \right] | \alpha | \tag{F.32}$$

Now

$$\begin{bmatrix} n \\ j \end{bmatrix} < n^m \quad \text{since} \quad j \leq m$$

thus

$$\| p(n) - \alpha_1 b_1 \| < \sum_{i=2}^{m} \frac{B_{ij}'' n^m}{2^n} | \alpha_i | \tag{F.33}$$

$| \alpha_i |$ are bounded as $\lambda \to \infty$ since $\alpha_1, \ldots, \alpha_m$ are bounded for P_∞ and bounded for P with $\lambda = T$, and $\alpha_1, \ldots, \alpha_m$ are continuous functions of λ as discussed above. Hence $\alpha_1, \ldots, \alpha_m$ are bounded for $\lambda > T$ and

$$\| p(n) - \alpha_1 b_1 \| < \frac{B'' n^m}{2^n} \tag{F.34}$$

for some $B'' > 0$ independent of λ. Hence uniform convergence has been established.

[NOTE:

$$\frac{B''n^m}{2^n} \to 0 \quad \text{as} \quad n \to \infty$$

since

$$\lim_{n\to\infty} \frac{\left[\frac{B''(n+1)^m}{2^{n+1}}\right]}{\left[\frac{B''(n)^m}{2^n}\right]} = \lim_{n\to\infty} \frac{\left[\frac{n}{n+1}\right]^m}{2} = \frac{1}{2} \quad]$$

Equation (5.66) is justified by showing

$$\lim_{\lambda\to\infty} \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) = \lim_{N\to\infty} \lim_{\lambda\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} p(n) \qquad \text{(F.35)}$$

To establish (F.35) it is shown there exists a $T > 0$ such that $q(N) \triangleq \frac{1}{N} \sum_{n=0}^{N-1} p(n)$ is uniformly convergent for $\lambda > T$. The proof is similar to that used for (F.6), hence only a sketch of the proof is given.

Since $D_1 + D_3 = 0$, $P_\infty(0,0) = \lim_{\lambda\to\infty} P$ is used. From the matrix formed from (5.36), (5.37) and (5.38) and by expanding the determinant along the top row

$$\det(P_\infty(0,0) - eI) = (e^2-1)(-e)^{m-2} \qquad \text{(F.36)}$$

Equation (F.36) gives the eigenvalues, denoted $\infty_1(\infty)$, $e_2(\infty)$, ..., $e_m(\infty)$, of $P_\infty(0,0)$

$$e_1(\infty) = 1$$
$$e_2(\infty) = -1$$
$$e_i(\infty) = 0 \qquad i = 3, 4, \ldots, m \tag{F.37}$$

By continuity of the eigenvalues with respect to λ, there exists $T > 0$ such that $\lambda > T$ gives

$$e_1(\lambda) = 1$$
$$\mathrm{Re}\left[e_2(\lambda)\right] < -\frac{1}{2} \quad , \quad | e_2(\lambda) | < 1$$
$$| e_i(\lambda) | < \frac{1}{2} \quad , \quad i = 3, 4, \ldots, m \tag{F.38}$$

Using the same argument as above, it follows that

$$\| p(N) - \alpha_1 b_1 \| < \sum_{i=2}^{m} \frac{B_i'''}{N} | \alpha_i | \tag{F.39}$$

Again $| \alpha_i |$ are bounded for $\lambda > T$. Thus

$$\| q(N) - \alpha_1 b_1 \| < \frac{B}{N} \tag{F.40}$$

for some $B > 0$ independent of λ.

Equation (F.40) establishes uniform convergence.

APPENDIX G

PROOF OF THEOREMS 6.1 AND 6.2

Theorem 6.1 The probability transition matrix, P, of the fixed priority arbiter model has a unique positive limiting probability vector independent of the initial probability vector provided $\lambda_h, \mu_h > 0$, for h=1, .., k and $D_1 + D_2 > 0$.

Proof: It is sufficient to show P is irreducible and primitive by establishing that there exists an $r > 0$ such that

$$P^r > 0 \tag{G.1}$$

To prove (G.1) it suffices to find a sequence of r transitions from state j to state i, for all states i and j. (In this appendix "transition" is to be interpreted as a non zero probability transition unless stated otherwise). Two cases are considered: (1) $D_1 > 0$, (2) $D_2 > 0$.

(1) $D_1 > 0$:

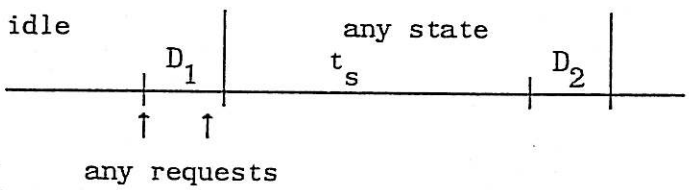


FIGURE G.1 Zero to Non Zero Transition

$D_1 > 0$ allows transitions from the idle state to any other state as shown in Figure G.1. In particular, if $D_2 = 0$ a transition to the full state can *only* occur from the idle state. It follows that a lower bound on r in order that (G.1) holds when $D_1 > 0$ and $D_2 = 0$ is k+1 (determined by observing that the full state as starting and finishing states requires the idle state as one of the intermediate states). Exactly k+1 transitions are constructed between any two states i and j by considering two sub-cases: (1.a) i non zero, (1.b) i zero.

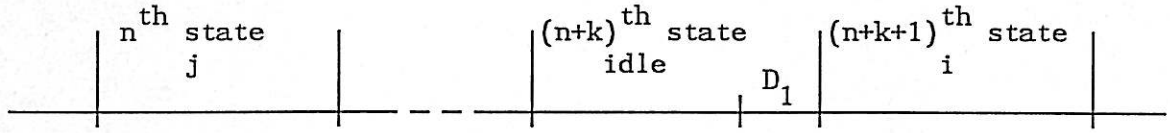


FIGURE G.2 k+1 State Transisitons j to $i \neq 0$, $D_1 \geq 0$.

(1.a) $i \neq 0$

It suffices to find k transitions from state j to the idle state, as shown in Figure G.2. After $|j| - 1$ ($\leq k - 1$) transitions where no new requests are lodged, a singleton state is reached (except when $j = 0$ in which case let any single request occur). The remaining transitions before the idle state of Figure G.2 ($k - |j| + 1$ of them for $j \neq 0$, or $k - 1$ for $j = 0$) can be constructed by a sequence of singleton states alternating between {1} and {2}, say, (starting with a different state to the previous singleton state) as shown in Figure G.3.

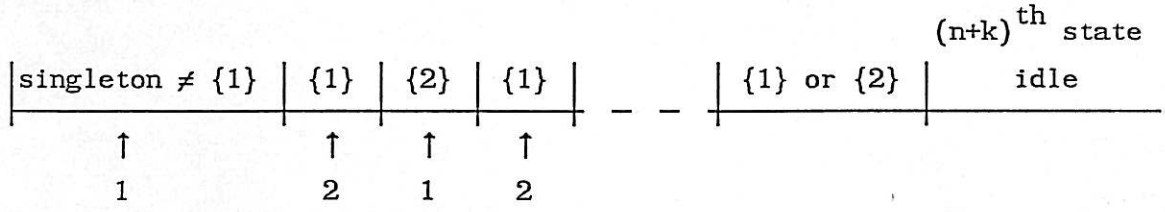


FIGURE G.3 Padding with Singleton States up to the $(n+k)^{th}$ State.

(1.b) $i = 0$

The same transitions up to the singleton $(n+k-1)^{th}$ state are employed as above. A singleton state is generated for the $(n+k)^{th}$ state by an appropriate request and then no requests occur resulting in the $(n+k+1)^{th}$ state being idle as required.

(2) $D_2 > 0$

Two subcases are again considered : (2.a) $i \neq 0$, (2.b) $i = 0$.

(2.a) $i \neq 0$

k transitions from any state j to $i \neq 0$ are constructed as follows (see Figure G.4): Requests are withheld until the state (labelled the $(n+\ell)^{th}$ state) is a subset of i (if $j = 0$, then hp(i) requests). This takes at most k-1 transitions (i.e. $\ell \leq k-1$). The requests not already pending in state i are lodged during the $(n+\ell)^{th}$ state along with a request from the requester serviced. Requester hp(i) then keeps requesting in D_2 of successive states to maintain a sequence of states equal to i until the $(n+k)^{th}$ state.

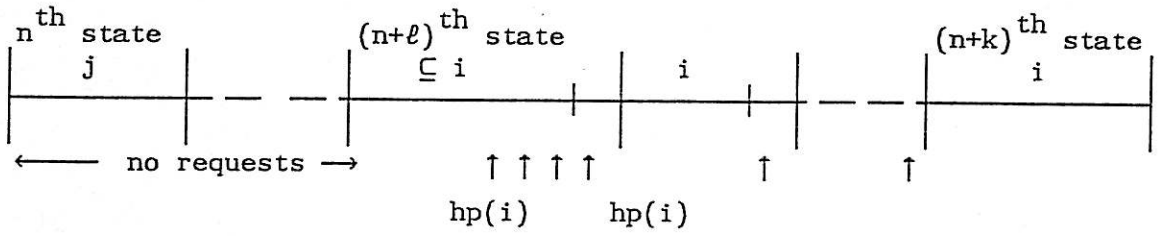


FIGURE G.4 Transitions to State i, $D_2 > 0$.

(2.b) $i = 0$

The proof is the same as (1.b) above. ▨

Theorem 6.2 For $D_1 + D_2 = 0$, the probability matrix P_1, has a unique positive limiting probability vector independent of the initial probability vector, provided $\mu_h, \lambda_h > 0$ for h=1, ..., k.

Proof: The same approach as in the previous proof is adopted, except now the full state is not included. A total of $k + 2$ transitions are constructed between any two non full states.

Firstly, k-1 transitions to the idle state are constructed.

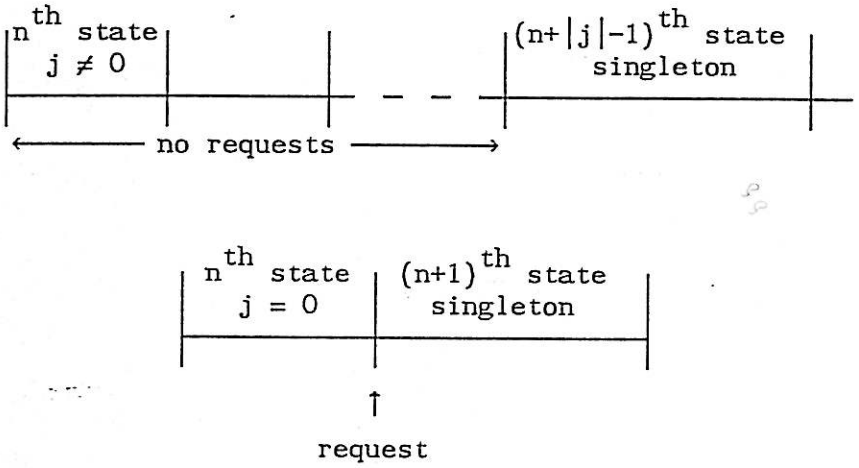


FIGURE G.5 Transitions to Singleton State, $D_1 = D_2 = 0$.

As shown in Figure G.5, $|j| - 1$ transitions can be constructed to a singleton state when $j \neq 0$. It remains to add (N.B. $|j| \leq k-1$) transitions to reach the idle state in a total of k-1 transitions. Transitions between singleton states can be arranged as shown in Figure G.6.

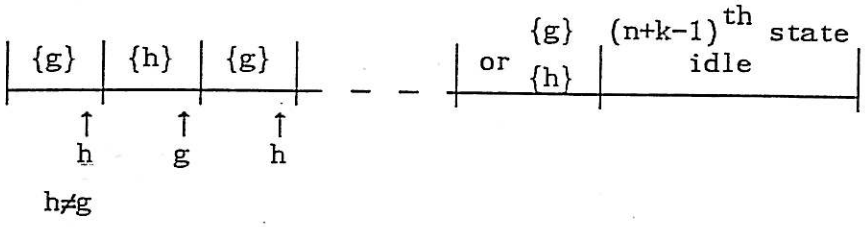


FIGURE G.6 Padding up to the $(n+k-1)^{th}$ state with Singleton States, $D_1 + D_2 = 0$.

After the singleton state an extra three transitions are constructed as shown in Figure G.7. The state {hp(full \ i)} is a singleton state which allows the final transition to i (even when i = 0). ▨

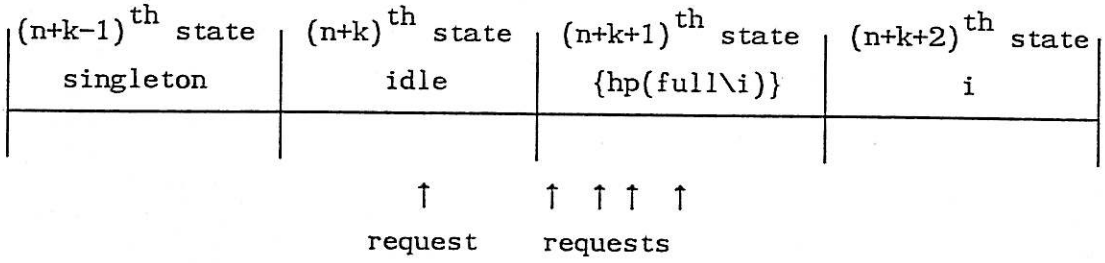


FIGURE G.7 Remaining Three Transitions, $D_1 + D_2 = 0$.

APPENDIX H

DERIVATION OF THE CONDITIONAL MEAN WAITING TIME FOR THE FIXED PRIORITY ARBITER MODEL

The derivation of MWT(h) started in Section 6.5.2 is completed in this appendix by deriving an expression for, cmwtf(j,h), the conditional mean waiting time of requester h given that requester h is serviced in state j.

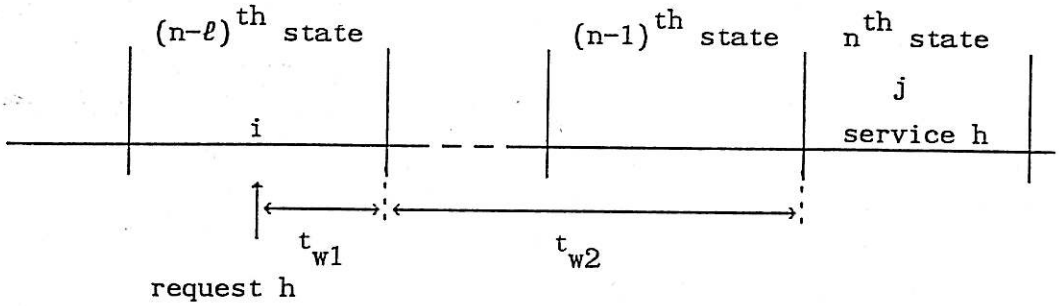


FIGURE H.1 Conditional Waiting Time.

The conditional waiting time is decomposed into two components, t_{w1} and t_{w2} as shown in Figure H.1. The mean of t_{w1}, denoted cmwtf1(i,h) is the mean time between request h and the end of the $(n-\ell)^{th}$ state i (defined as the last state before j such h $\notin$ state) given that request h occurs during this state i. The mean of t_{w2}, denoted cmwtf2(j,h,ℓ) is the mean time duration of the $(\ell-1)$ states from the $(n-\ell+1)^{th}$ state to the n^{th} state, given that requester h is serviced in the n^{th} state j (and, consequently, no state can occur during t_{w2} without request h pending). It follows that

$$\text{cmwtf}(j,h) = \sum_{\ell=1}^{\infty} \left\{ \text{cmwtf2}(j,h,\ell)\ \text{prob}\left[\begin{array}{l|l} \text{request h lodged} & \text{h serviced} \\ \text{in } (n-\ell)^{\text{th}} \text{ state} & \text{in } n^{\text{th}} \text{ state} = j \end{array} \right] \right.$$

$$\left. + \sum_{i=0}^{m-1} \text{cmwtf1}(i,h)\ \text{prob}\left[\begin{array}{l|l} \text{request h lodged in} & \text{h serviced} \\ (n-\ell)^{\text{th}} \text{ state} = i & \text{in } n^{\text{th}} \text{ state} = j \end{array} \right] \right\}$$

(H.1)

Expressions for quantities in (H.1) are derived in turn. Three cases of i need to be considered in order to evaluate cmwtf1(i,h) : (a) i = 0; (b) h = hp(i) and (c) h $\notin$ i $\neq$ 0:

(a) i = 0

$$\text{cmwft1}(0,h) = \sum_{\substack{v \text{ such that} \\ h \in v}} p_{v0} \left\{ \frac{D_1 \lambda_h}{\sum_{g \in v} \lambda_g} + \text{crt}(D_1,h) \left[1 - \frac{\lambda_h}{\sum_{g \in v} \lambda_g} \right] \right\}$$

(H.2a)

where $\text{crt}(D_1,h)$ is defined in (E.6).

(b) h = hp(i)

$$\text{cmwtf1}(i,h) = \text{crt}(D_2,h) \qquad \text{(H.2b)}$$

(c) h $\notin$ i $\neq$ 0

Let g = hp(i), then it follows from (E.3) that

$$\text{cwtf1}(i,h) = \int_{D_2}^{\infty} f_g(t)\ \text{crt}(t,\ h)\ dt$$

$$= \begin{cases} \text{crt}(D_2 + \frac{1}{\mu_g}) & \text{, constant service times} \\ \\ \text{(E.9) with } D = D_2,\ r = 1,\ \mu = \mu_g & \text{, exponential service times} \end{cases}$$

(H.2c)

The quantity $\text{cmwtf2}(j,h,\ell)$ can be expressed as

$$\text{cmwtf2}(j,h,\ell) = (\ell-1)(D_2 + \frac{1}{\mu}) \qquad \text{(H.3)}$$

when $\mu_f = \mu$ for all f. A more general expression than (H.3) is given in (H.12) below.

Define

$$q(\ell,h,i) \triangleq \text{prob}\left[\begin{array}{l|l} \text{request h lodged in} & \text{h serviced} \\ (n-\ell)^{\text{th}} \text{ state} = i & \text{in } n^{\text{th}} \text{ state} = j \end{array}\right]$$

where $h \notin i \setminus \{hp(i)\}$ (H.4)

and $$Q \triangleq [q_{ij}] \triangleq \text{prob}\left[v^{\text{th}} \text{ state} = i \mid (v+1)^{\text{th}} \text{ state} = j\right] \qquad \text{(H.5)}$$

Q is the reverse probability transition matrix and can be evaluated as follows : Let A be the event "v^{th} state = i" and B be the event "$(v+1)^{\text{th}}$ state = j", then

$$\text{prob}(A|B) \triangleq \frac{\text{prob}(A \cap B)}{\text{prob}(B)}$$

$$= \frac{\text{prob}(B|A) \cdot \text{prob}(A)}{\text{prob}(B)} \qquad \text{(H.6)}$$

Since MWT(h) is taken under limiting steady conditions (i.e. $v \to \infty$) (H.6) gives:

$$\bar{q}_{ij} = \frac{p_{ji}\, p_i}{p_j} \qquad \text{(H.7)}$$

The probability $q(\ell,h,i)$ of (H.4) can be evaluated inductively on ℓ as follows. Let $\tilde{q}(\ell,h)$ be the state vector $[q(\ell,h,i)]^T$ (where $q(1,h,i) = 0$ when $h \notin i \setminus \{hp(i)\}$)

then

$$\tilde{q}(1,h) = H\, Q\, \hat{j} \qquad \text{(H.8)}$$

where $H = [h_{ij}]$

and $$h_{ij} = \begin{cases} 0 & i \neq j \\ 0 & i = j \ , \ h \in i \setminus \{hp(i)\} \\ 1 & i = j \ , \ h \notin i \setminus \{hp(i)\} \end{cases}$$

$$\tilde{q}(2,h) = H\, Q\, \bar{H}\, Q\, \hat{j} \qquad \text{(H.9)}$$

where $\bar{H} = [\bar{h}_{ij}]$

where $$\bar{h}_{ij} = \begin{cases} 0 & , \ i \neq j \\ 1 - h_{ii} & , \ i = j \end{cases}$$

and in general

$$\tilde{q}(\ell,h) = H(Q\, \bar{H})^{\ell-1}\, Q\, \hat{j} \qquad \text{(H.10)}$$

Now

$$\text{prob}\left[\begin{array}{l|l}\text{request h lodged} & \text{request h serviced}\\ \text{in } (n-\ell)^{th} \text{ state} & \text{in } n^{th} \text{ state}\end{array}\right]$$

$$= \underset{\sim}{1}\, H\, (Q\, \bar{H})^{\ell-1}\, Q\, \hat{j} \qquad \text{(H.11)}$$

where $\underset{\sim}{1} = [1,\ 1,\ \ldots,\ 1]^T$

The above waiting time expressions have been implemented in a PASCAL program and agree with Monte-Carlo simulations over a wide range of the parameters D_1, D_2, λ, μ and requester h.

The more general expression than (H.3) with differing μ's for cmwtf2 promised, can now be stated :

$$\text{cmwtf2}(j,h,\ell) = \sum_{g=1}^{\ell-1} \left(D_2 + \frac{\underset{\sim}{\mu}^T\, (\bar{H}\, Q)\, \hat{j}}{\underset{\sim}{1}^T\, (\bar{H}\, Q)\, \hat{j}}\right) \qquad \text{(H.12)}$$

APPENDIX I

JUSTIFICATION OF EQUATIONS (6.23), (6.28) AND (6.34)

After studying Appendix F, one can determine whether the same uniform convergence results apply to other matrices by examining their eigenvalues. Not surprisingly, it turns out that the problem is almost identical to that of the batched model.

In order to establish equation (6.23), the limiting probability transition matrix as request rates tend to zero, P_0, can be taken from (6.7):

$$P_0 = \left[\begin{array}{ccccc|c} 0 & 1 & 1 & \cdots & 1 & \\ r_1 & 0 & 0 & & 0 & \\ r_2 & 0 & 0 & & 0 & \\ \cdot & & & & \cdot & \\ \cdot & & & & \cdot & \\ r_k & 0 & 0 & & 0 & \\ \hline & & & & & \\ & & 0 & & & \\ & & & & & 0 \end{array}\right] \qquad \text{(I.1)}$$

where the states in the matrix P_0 are conveniently reordered so that the group of states i, with $|i| = 0$ is followed by $|i| = 1$, $|i| = 2$ .. $|i| = k$.

The characteristic equation, $\det(P_0 - vI) = 0$ is

$$(v^2 - 1)\, v^{m-3} = 0 \qquad \text{(I.2)}$$

and so the eigenvalues are

$$v = +1,\ -1,\ 0,\ \ldots,\ 0 \tag{I.3}$$

which are identical to those in (F.9) and so (6.23) can be deduced in the same fashion.

Equation (6.28) follows from determining the eigenvalues of $P_\infty(+,0)$ which is $\lim_{\lambda\to\infty} P$ when $D_1 > 0$, $D_2 = 0$, and $P_\infty(0,0)$ which is $\lim_{\lambda\to\infty} P$ when $D_1 = D_2 = 0$. From (6.13) it follows that

$$P_\infty(+,0) = \begin{bmatrix} 0 & & & & & \\ & 0 & & & & \\ & & \ddots & & & \\ \text{don't care} & & & 0 & 1 & 0 \leftarrow \text{full} \setminus \{k-1\} \\ & \text{don't care} & & 1 & 0 & 1 \leftarrow \text{full} \setminus \{k\} \\ & & & 0 & 0 & 0 \leftarrow \text{full} \end{bmatrix} \tag{I.4}$$

where the same state ordering is employed as in (I.1) and "don't care" boxes contain 0's and 1's, and 0's are everywhere else not specified. The eigenvalues of (I.4) are now easily obtained as

$$v = 1,\ -1,\ 0,\ 0,\ \ldots,\ 0 \tag{I.5}$$

$P_\infty(0,0)$ has the same set of eigenvalues as $P_\infty(+,0)$ as can be seen by following the same process. Again, the same eigenvalues of (I.5) are treated in Appendix F (F.37) in the heavy loading limit case.

Equation (6.34) follows from determining the eigenvalues of $P_\infty(+,+)$, which is $\lim_{\lambda\to\infty} P$ when $D_1 > 0$ and $D_2 > 0$, and $P_\infty(0,+)$, which is $\lim_{\lambda\to\infty} P$ when $D_1 = 0$, $D_2 > 0$. The matrix $P_\infty(+,+)$ can be

obtained from equation (6.12):

$$\bar{P}_{\infty}(+,+) = \begin{bmatrix} 0 & 0 & .. & 0 \\ 0 & & & 0 \\ . & & & . \\ & & & . \\ 0 & & & 0 \\ 1 & 1 & .. & 1 \end{bmatrix} \qquad \text{(I.6)}$$

The matrix $P_{\infty}(+,+)$ and $P_{\infty}(0,+)$ have eigenvalues 1, 0, 0, ..,0 and (F.29) applies from Appendix F.

APPENDIX J

THE EFFECT OF NON IDEALISED TIMING MODELLING IN CLOCKED BATCHED ARBITERS

In the ideal modelling assumptions for clocked arbiters in Chapter 7, effects such as clock skew, non zero rise and fall times and non zero propagation delays were ignored. This aided in simplifying the analysis of timing and the clarity of notation. This appendix points out which assumptions are needed only for notation elegance and clarity, and those timing assumptions which cannot be easily dispensed with.

The inclusion of constant delays, independent of component or module (as opposed to differential delays between components of the same type), can be included in the definitions of D_2 , $t_a(h)$ and D_1 without great difficulty. However, the values obtained will be dependent on the batch composition in the case of D_2 and $t_a(h)$ as follows: D_2 includes a full clock period plus the propagation delay of FF3 and 2m gate delays for the daisy chain to propagate to module m. That is D_2 is now a function of the first request serviced in a batch. Similarly, values of $t_a(h)$ must account for delays in the daisy chain. The exact details depend on the resetting strategy. The timing of the dropping of the last Ack at the end of a batch will be delayed by a constant reset delay, dependent on resetting strategy, plus a gate delay. This reduces D_1 which is the time available for the last requester serviced in a batch to request before the end of batch point.

From experience with the asynchronous model in Chapter 5, the behaviour of arbiter is most sensitive to D_1 and relatively insensitive to small changes in D_2 and $t_a(h)$. The effect of increasing D_2 and $t_a(h)$ is roughly equivalent to increasing the request loading on the

arbiter, since more contention occurs due to less available servicing time. Variations in D_1, however, most critically affect the allocation of the resource to the two lowest priority requesters under heavy request loading conditions because their probability of receiving service in the batch following a service is dependent on the available time in which to service before the end of batch. By reducing D_1, as is the case when non zero delays are introduced, the two lowest priority requesters are more disadvantaged. This can be easily modelled by appropriately adjusting the value of D_1 in the transition probabilities to account for the delay incurred in resetting after a request drops. In summary, constant delays can be modelled by modifying the definitions of D_1 , D_2 and $t_a(h)$ slightly. The added complication in the model would only result in significant differences over the idealised model when delays are significant compared with the clock period.

The inclusion of differential delays between components of the same type would involve considerably more complexity, but could be incorporated. The benefits obtained from this complexity are of questionable value since the size of differential delays is expected to be smaller than the mean delay for a component.

Accounting for clock skew in the modelling causes problems because an absolute fixed time frame is assumed to exist in all modules. If clock skew were significant, timing diagrams would be needed for each module and no unique "end of batch" point would exist. In fact, the timing of the arbiter may break down altogether, since the design of a synchronous circuit depends on the clock to strictly sequence events within the circuit. Although the model could not easily account for clock skew, it is apparent that the circuit design itself relies on clock skew being small anyway and so the assumption is not unreasonable in the model itself.

APPENDIX K

DERIVATION OF TRANSITION PROBABILITIES FOR THE NO RESETTING CLOCKED BATCHED ARBITER OF CHAPTER 7

The acknowledge time for requester h , $t_a(h)$, has a discrete distribution which is dependent on whether h is serviced first in a batch. Suppose $t_s(h)$ has a probability density function f_h then define the discrete probability functions $f_h^*(m)$ and $f_h^{**}(m+\frac{1}{2})$ as follows

$$f^*_h(m) \triangleq \text{prob}\,[t_a(h) = mT_c \,|\, h \text{ not serviced first in batch}]$$

$$= \int_{(m-\frac{3}{2})T_c}^{(m-\frac{1}{2})T_c} f_h(t)dt \qquad \text{(B.1)}$$

(N.B. $f_h(t)=0$ for $t < 0$)

$$f_h^{**}(m+\tfrac{1}{2}) \triangleq \text{prob}\,[t_a(h) = (m+\tfrac{1}{2})T_c \,|\, h \text{ serviced first in batch}]$$

$$= \int_{(m-1)T_c}^{mT_c} f_h(t)dt \qquad \text{(B.2)}$$

The probability of request h not having a request pending at the end of a non zero batch with state i is denoted $Q(i,h)$ and is given by

$$Q(i,h) = \begin{cases} e^{-\lambda_h D_1} \prod_{\substack{\ell \in i \\ \ell > h}} \sum_{m=1}^{\infty} f_\ell^*(m) e^{-\lambda_h m T_c} & , h \in i \\ e^{-\lambda_h (D_1 + D_2)} \prod_{\substack{\ell \in i \\ \ell \neq g}} \sum_{m=1}^{\infty} f_\ell^*(m) e^{-\lambda_h m T_c} & \\ \quad \times \sum_{m=1}^{\infty} f_g^{**}(m)\, e^{-\lambda_h (m+\frac{1}{2}) T_c} & , h \notin i \end{cases} \tag{B.3}$$

where g is the requester serviced first in batch i.

The transition probabilities now follow:-

$$\text{prob}\,[S(n)=i \mid S(n-1)=j]$$

$$= \begin{cases} 0 & , i=j=0 \\ \dfrac{\prod_{g \in i} (1-e^{-\lambda_g T_c}) \prod_{f \notin i} e^{-\lambda_f T_c}}{1 - \prod_{h=1}^{k} e^{-\lambda_h T_c}} & , i \neq 0,\ j=0 \\ \prod_{h=1}^{k} Q(j,h) & , i=0,\ j \neq 0 \\ \prod_{h \notin i} (1 - Q(j,h)) \prod_{h \in i} Q(j,h) & , i \neq 0,\ j \neq 0 \end{cases} \tag{B.4}$$

AUTHOR'S PUBLICATIONS

[1] "Can redundancy and masking improve the performance of synchronisers?", *IEEE Trans. on Computers*, Vol. C-35, No. 7, pp. 643-646, July 1986, also Tech. Report EE8440, Dept. Elec. & Computer Engg.,University of Newcastle, N.S.W., Australia, August 1984

[2] "Metastable behaviour and digital systems reliability" to appear *IEEE Design and Test* also Tech. Report EE8441, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, September 1984.

[3] "On the unavoidability of metastable behaviour in digital systems", to appear *IEEE Trans. on Computers*, also Tech. Report EE8516, Dept. Elec. & Computer Engg.,University of Newcastle, N.S.W., Australia, December 1984.

[4] "The modelling and performance analysis of batching arbiters", *Performance 86 and ACM Sigmetrics 1986 Conference on Computer Performance Modelling, Measurement and Evaluation*, North Carolina State University, May, 1986.

[5] "Performance of Arbitration Disciplines", 2nd Australian Computer Engineering Conference, Sydney, 1986.

[6] "A class of arbiters - structures and performance analysis", Tech. Report EE8421, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, June 1984.

[7] "Decentralised asynchronous batching arbiter timing analysis and proof", Tech. Report EE8531, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, July 1985.

[8] "Decentralised clocked batching arbiter - modelling and metastable failure modes", Tech. Report EE8532, Dept. of Electrical. & Computer Engg., University of Newcastle, N.S.W., Australia, August 1985.

REFERENCES

[A.1] Adams, R.L., Castaldo, D.R. and Kurtz, G.W., "Real-time detection of latch resolution using threshold means", (US patent 3,515,998), June 1970.

[B.1] Bain Jr., W.L. and Ahuja, S.R., "Performance analysis of digital buses for multiprocessing", *Proc. 8th Int. Symp. Computer Arch.*, Minneapolis, Minn., May 1981.

[B.2] Barros, J.C. and Johson, B.W., "Equivalence of the arbiter, the synchronizer, the latch, and the inertial delay", *IEEE Trans. Computer*, Vol. C-32, pp. 603-614, July 1983.

[B.3] Bell, C. et all., Computer Engineering - A DEC View of Hardware Systems Design, Digital Press, pp. 280-286, 1978.

[B.4] Best, D. and Watson, W., "Distributed priority access to a computer unit", (US patent.3,573,856), April 1971.

[B.5] Bonkright, W. et al., "The Illiac IV System", *Proc. IEEE*, No.60, pp. 369-388, 1972.

[B.6] Bowen, B.A. and Burh, R.J.A., The logical design of multi-microprocessor systems, Prentice-Hall, Ch.4, 1980.

[C.1] Calvo, J., Acha, J.I. and Valencia, M., "Asynchronous modular arbiter", *IEEE Trans. on Computers*, Vol. C-35, pp. 67-70, January 1986.

[C.2] Cantoni, A., "A technique for interrupt distribution in a multiprocessing system", *Software & Microsystems*, Vol.1, No.6, pp. 153-159, October 1982.

[C.3] Caprani, O., Jensen, K.H. and Ougaard, U., "Microprocessor connected to a common memory", *Euromicro 77 Symposium Proceedings*, North Holland Pub. Co., pp. 175-181, 1977.

[C.4] Catt, I., "Time loss through gating of asynchronous logic signal pulses", *IEEE Trans. Electronic Computers*, Vol. EC-15, pp. 108-111, February 1966.

[C.5] Cioffi, G. and Velardi, P., "A fully distributed arbiter for multiprocessor systems", *Multiprocessing and Microprogramming* No.11, pp. 15-22, 1983.

[C.6] Chaney, T.J., "Comments on 'A note on synchronizer and interlock maloperation' ", *IEEE Trans. on Computers*, Vol.C-28, pp. 802-804, October 1979.

[C.7] Chaney, T.J., "Measured flip-flop responses to marginal triggering", *IEEE Trans.on Computers*, Vol.C-32, pp. 1207-1209, December 1983

[C.8] Chaney, T.J. and Molnar, C.E., "Anomalous behavior of synchronizer and arbiter circuits", *IEEE Trans. on Computers*, Vol.C-22, pp. 421-422, April 1973.

[C.9] Chaney T.J., Ornstein, S.M. and Littlefield W.M., "Beware of the synchronizer", *COMPCON '72*, Jack Tar Hotel, San Francisco, CA, pp. 317-319, September 1972.

[C.10] Cojan, A., Pregernig, L, Tark, J.C. and Downing, R., "The fastbus cable segment", *IEEE Trans. Nuclear Science*, Vol.N5-31, No.1, February 1984.

[C.11] Conte, G. and Corso, D., (editors), Multi-microprocessor systems for real-time applications, D. Reidel Publishing Co., Dordrecht, Holland, 1985.

[C.12] Corsini, P., "n-user asynchronous arbiter", *Electronic Letters*, Vol.II, pp. 1-2, 9th January 1975.

[C.13] Corsini, P., "Speed independent asynchronous arbiter", *Computers and Digital Techniques*, Vol.2, pp. 221-222, October 1979.

[C.14] Cortes, M. and McCluskey E.J., "Modelling power supply disturbances in digital circuits", *ISSCC'86*.

[C.15] Cortes, M., McCluskey, E.J., Wagner, K.D. and Lu, D.J., "Properties of transient errors due to power supply disturbances", *ISCAS'86*.

[C.16] Couranz, G.R. and Wann, D.F., "Theoretical and experimental behavior of synchronizers operating in the metastable region", *IEEE Trans. on Comput.*, Vol.C-24, pp. 604-616, June 1975.

[C.17] Courvoisier, M., "A component for multimicroprocessor structures: A programmable arbiter", *Proc. Fall Compcon'79, IEEE* 1979, pp. 307-311.

[D.1] Davies, D. and Wakerly, J.F, "Synchronization and matching in redundant systems", *IEEE Trans. on Computers*, Vol.C-27, pp. 531-539, June 1978.

[D.2] Digital Equipment Corporation Special Systems, PCLII - An option description, Document No.YC-C000C, November 1976.

[E.1] Elineau, G. and Wiesbeck, W., "A new J-K flip-flop for synchronizers", *IEEE Trans. on Computers*, Vol.C-26, pp. 1277-1278, December, 1977.

[E.2] Enslow, P.H., "Multiprocessor organisation - a survey", *ACM Comp Surveys*, 1977, 9 (1).

[F.1] Farber, G., 'A decentralised fair bus arbiter", *Multiprocessing and Microprogramming*, Vol.7, No.1, 1981.

[F.2] Fleischhammer, W. and Dortok D., "The anomalous behavior of flip-flops in synchronizer circuits", *IEEE Trans. on Computers*, Vol.C-28, pp. 273-276, March 1979.

[F.3] Fletcher, W.I., An Engineering Approach to Digital Design, Englewood Cliffs, N.J., Prentice-Hall, p. 484, 1980.

[F.4] Freeman, G.G., Liu, D.L., Wooley, B. and McCluskey, E.J., "Two CMOS metastable sensors", CRC Tech. Report No. 84-4, Dept. Elect. Engg. and Computer Science, Stanford University, CA., May 1986.

[G.1] Gantmacher, F.R., Matrix Theory, Vol.II, N.Y. Chelsea Publishing Co., 1974.

[G.2] Giffin, W.C., Queueing, Basic Theory and Applications, Grid Inc., Columbus, Ohio, 1978, ISBN 0.88244-133-7.

[G.3] Glasser, L.A. and Dopperpuhl, D.W., The Design and Analysis of VLSI Circuits, Addison-Wesley, pp. 360-365, 1985.

[G.4] Grasso, P.A., Dillon, T.S. and Forward, K.E., "Memory interference in multi-microprocessor systems with a time-shared bus", *IEE Proc.*, Vol.131, Pt.E, March 1984.

[G.5] Gustavson, D,B and Theus, J., "Wired-or logic on transmission lines", *IEEE Micro*, pp. 51-55, June 1983.

[H.1] Hanson, P.B., Operating System Principles, Prentice-Hall, Englewood Cliffs, N.J., 1973.

[H.2] Hoener, S. and Roehder, W., "Efficiency of a multiprocessor system with time shared busses", *Euromicro 77 Symposium Proceedings*, North Holland Pub. Co., pp. 35-42.

[H.3] Hohl, J.H., Larsen, W.R. and Schooley, L.C., "Prediction of error probabilities for integrated digital synchronizers", *IEEE Journ. Solid-state Circuits*, Vol.SC-19, pp. 236-244, April 1984.

[H.4] Hurtado, M. and Elliott, D.L., "Ambiguous behavior of logic bistable systems", in *Proc. 13th Annu. Allerton Conf. Circuit and Syst. Theory*, October 1975.

[I.1] Intel, Microsystems Components Handbook, Vol.I, Appendix B, p. 3-315, 1984.

[I.2] Intel Application Notes AP51, 8289, Bus Arbiter, 1985.

[I.3] Intel, Component Data Catalog, 8259 Programmable Interrupt Controller, pp. 6.116-6.133, 1980.

[J.1] Jaiswal, N.K., Priority Queues, Academic Press, New York and London, 1968 (Vol.50 in series Mathematics in Science and Engineering).

[K.1] Kameyama, M. and Higuichi, T., "Design of dependent-failure tolerant microcomputer system using triple-modular redundancy", *IEEE Trans. on Computers*, Vol.C-29, pp. 202-206, February 1980.

[K.2] Katsuk, B., et al., "PLURIBUS - An operational fault-tolerant multiprocessor", *Proc. IEEE*, October 1978.

[K.3] Kinniment, D.J. and Woods, J.V., "Synchronisation and arbitration circuits in digital systems", *Proc. IEE*, Vol. 123, Pt E, pp. 961-966, October 1976.

[K.4] Kleeman, L. and Cantoni, A., "A class of arbiters - structures and performance analysis", Tech. Report EE8421, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, June 1984.

[K.5] Kleeman, L. and Cantoni, A., "Can redundancy and masking improve the performance of synchronisers?", to appear 1986 *IEEE Trans. on Computers*, also Tech. Report EE8440, Dept. Elec. & Computer Engg.,University of Newcastle, N.S.W., Australia, August 1984

[K.6] Kleeman, L. and Cantoni, A., "Metastable behaviour and digital systems reliability" to appear *IEEE Design and Test* also Tech. Report EE8441, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, September 1984.

[K.7] Kleeman, L. and Cantoni, A., "On the unavoidability of metastable behaviour in digital systems", to appear *IEEE Trans. on Computers*, also Tech. Report EE8516, Dept. Elec. & Computer Engg.,University of Newcastle, N.S.W., Australia, December 1984.

[K.8] Kleeman, L. and Cantoni, A., "Decentralised asynchronous batching arbiter timing analysis and proof", Tech. Report EE8531, Dept. Elec. & Computer Engg., University of Newcastle, N.S.W., Australia, July 1985.

[K.9] Kleeman, L. and Cantoni, A., "Decentralised clocked batching arbiter - modelling and metastable failure modes", Tech. Report EE8532, Dept. of Electrical. & Computer Engg., University of Newcastle, N.S.W., Australia, August 1985.

[K.10] Kleeman, L. and Cantoni, A., "The modelling and performance analysis of batching arbiters", *Performance 86 and ACM Sigmetrics 1986 Conference on Computer Performance Modelling, Measurement and Evaluation*, North Carolina State University, May, 1986.

[K.11] Kleeman, L. and Cantoni, A., "Performance of Arbitration Disciplines", accepted 2nd Australian Computer Engineering Conference, Sydney, 1986.

[K.12] Kleinrock, L. Queueing Systems, Vol.1, John Wiley, 1976.

[K.13] Kovaleski, A.B., "High-speed bus arbiter for multiprocesssors", *IEE Proc.*, Vol.130, Pt E, No. 2, pp. 49-56, March 1983.

[K.14] Kriz, J., "A queueing analysis of symmetric multiprocessor with shared memory and buses", *IEE Proc.*, Vol.130, Pt.E, No.3, pp. 83-89 May 1983.

[L.1] Lacroix, G., Marchegay, P. and Piel, G., "Comments on 'The anomalous behavior of flip-flops in synchronizer circuits'", *IEEE Trans. on Computers*, Vol.C-31, pp. 77-78, January 1982.

[L.2] Lamdan, T., "Asynchronous timing in logic systems", *Digital Processes*, 2(1976), pp. 157-162.

[L.3] Laws, D.A. and Alfke, P., Bipolar Microprocesor Logic and Inteface Data Book, pp. 8.4 to 8.7,1983.

[L.4] Lent, B., "A variable priority arbiter for resource allocation in asynchronous multiprocessor systems", *Multiprocessing and Microprogramming*, No.9, pp. 299-307, 1982.

[L.5] Lim, W. and Cox, J.R. Jr., "Clocks and the performance of synchronizers", *Proc. IEE*, Vol.130, pp. 57-64, March 1983.

[M.1] Manner, R., "The POLYBUS : A flexible and fault tolerant multiprocessor interconnection", *Interfaces in Computing*, No.2, pp. 45-68, 1984.

[M.2] Manner. R., "Hardware task/processor scheduling in a polyprocessor environment", *IEEE Trans. on Computers*, Vol.C-33, pp. 626-636, July 1984.

[M.3] Marino, L.R., "The effect of asynchronous inputs on sequential network reliability", *IEEE Trans. on Computers*, Vol.C-26, pp. 1082-1090, November 1977.

[M.4] Marino, L.R., "General theory of metastable operation", *IEEE Trans. on Computers*, Vol.C-30, pp. 107-115, February 1981.

[M.5] Marsan, M. Ajmone and Gregoretti, F., "Memory Interference models for a multi-microprocessor system with a shared bus and a single external common memory", *Multiprocessing and Microprogramming*, No.7, pp. 124-133, 1981.

[M.6] McConnel, S.R. and Siewiorek, D.P., "Sychronization and voting", *IEEE Trans. on Computers*, Vol.C-30, pp. 161-164, February 1981.

[M.7] Mead, C. and Conway, L., Introduction to VLSI Systems, Reading, MA, Addison-Wesley, Chapter 7, 1980.

[M.8] Moore, W.R. and Haynes, N.A., "A review of synchronisation and matching in fault tolerant systems", *IEE Proc.*, Pt E, Vol.131, pp. 199-124, July 1984.

[M.9] Muhlemann, K, "Arbiters priority access conflicts and the 'glitch' problem", *Microprocessors and their Applications*, EUROMICRO North Holland Publishing Company, pp. 391-401, 1979.

[P.1] Pearce, R.C., Fielel, J.A. and Little, W.D., "Asynchronous arbiter model", *IEEE Trans. on Computers*, pp. 931-932, September 1975.

[P.2] Peatman, J.B., Digital Hardware Design, McGraw-Hill International Book Co., Chapter 5, 1980.

[P.3] Pechoucek, M., "Anomalous response times of input synchronizers", *IEEE Trans. on Computers*, Vol.C-25, pp. 133-139, February 1976.

[P.4] Plummer, W.W., "Asynchronous arbiters", *IEEE Trans.on Computers*, Vol.C-21, No.1, January 1972.

[R.1] Reese, E.A., et al., "A 4Kx8 dynamic RAM with self refresh", *IEEE Journ. Solid State Circuits*, Vol.SC-16, No.5, October 1981.

[R.2] Rosenberger, F. and Chaney, T.J., "Flip-flop resolving time test circuit", *IEEE Journ. Solid State Circuits*, Vol.SC-17, pp. 731-738, August 1982.

[R.3] Rusell, R.M., "The CRAY-I computer system", *Communications ACM*, pp. 63-72, January 1978.

[R.4] Rektorys, K., Survey of Applicable Mathematics, Cambridge, Massachusetts, The M.I.T. Press, 1969.

[S.1] Sharma, D.K. and Ahuja, S.R., "A first-come-first-serve bus allocation scheme using ticket assignments", *Bell System Technical Journal*, V.60, No.7, September 1981.

[S.3] Shostack, R.E., Schwartz, R. and Melliar-Smith, P.M., "STP: A mechanized logic for specification and verification", *Proc. 6th Conf. on Automated Deduction*, Lecture Notes in Computer Science, 138, D.W. Loveland, Ed., Springer Verlag, New York, pp. 32-49, 1982.

[S.4] Shostak, R.E., "Deciding combinations of theories", *Journal ACM*, Vol.31, pp. 1-12, January 1984.

[S.5] Simmons, G.F., Introduction to Topology and Modern Analysis, New York, McGraw-Hill, 1963.

[S.6] Stoll, P.A., "How to avoid synchronizer problems", *VSLI Design*, pp. 56-59, November 1982.

[S.7] Stuck, M.J. and Cox, J.R. Jr., "Synchronization strategies", *Proc. Caltech Conf. on VLSI*, pp. 375-393, January 1979.

[T.1] Taub, D.M., "Arbitration and control acquisition in the proposed IEEE 896 futurebus", *IEEE Micro*, pp. 28-41, August 1984.

[T.2] Theus, J., Taub, M. and Ballakrishnan, R.V., "Futurebus anticipates coming needs", *Electronics*, pp. 108-112, July 1984.

[T.3] Thurber, K.J. et al., "A systematical approach to the design of digital bussing structures", *AFIPS Conference Proc.*, Full Joint Conference, 1972, 41 part II.

[T.4] Tyner, P., iAPx432 General Data Processor Architecture Reference Manual, Intel Corp., January 1981.

[U.1] Unger, S.H., "Asynchronous sequential switching circuits with unrestricted input changes", *IEEE Trans. on Computers*, Vol.C-20, pp. 1437-1444, December 1971.

[V.1] Veendrick, H.J.M., "The behavior of flip-flops used as synchronizers and prediction of their failure rate", *IEEE J. Solid State Circuits*, Vol.LSC-15, pp. 168-176, April 1980.

[W.1] Wakerly, J.F., "Transient failure in triple modular redundancy systems with sequential modules", *IEEE Trans. on Computers*, Vol.C-24, pp. 570-573, May 1975.

[W.2] Wakerly, J.F., "Microcomputer reliability improvement using triple modular redundancy", *Proc. IEEE*, Vol.64, pp. 889-895, January 1976.

[W.3] Wakerly, J.F., "The Intel MCS-48 microcomputer family: a critique", *IEEE Computer*, pp. 22-31, February 1979.

[W.4] Wensley, J.H., et al, "SIFT: Design and analysis of a fault tolerant computer for aircraft control", *Proc. IEEE*, Vol.66, pp. 1240-1255, October 1978.

[W.5] Wulf, W.A. and Bell, C.G., "C.mmp - A multi-miniprocessor", *Proc. AFIPS Fall Joint Computer Conference*, N.J., 1972.