Design and Analysis of Control Schemes for Soft Real-Time Computation Systems

Paul D. Alexander

Department of Electrical and Electronic Engineering
The University of Adelaide

Submitted for M.Eng.Sc on 14th September 1994
Contents

Abstract vii

List Of Figures viii

Declaration ix

Acknowledgements x

Symbol Table xi

1 Introduction 1

2 Solving for System State Probability 7

  2.1 Introduction 7

  2.2 Queueing System State Theory 8

  2.3 Solving State Transition Rate Diagrams 14

    2.3.1 Nomenclature 14

    2.3.2 Restriction on STRD Due to Algorithm 16

    2.3.3 Algorithm 19

    2.3.4 Adapting the System to Algorithm Requirements 23

  2.4 Summary 26
3 Imprecise Computation System Control Schemes

3.1 Introduction ........................................... 29
3.2 Task and System Models ............................... 31
3.3 Notice Generating Algorithms ....................... 36
  3.3.1 Structure of System State Under PNG_A .......... 47
  3.3.2 Structure of System State Under PNG_N .......... 47
  3.3.3 Structure of System State Under PNG_1 .......... 48
  3.3.4 Structure of System State Under PNG_H .......... 49
3.4 System Performance Metrics ......................... 49
3.5 Analysis of Control Schemes ......................... 52
  3.5.1 State Description of System ..................... 54
  3.5.2 State Transition Rate Diagram Description of System .... 55
  3.5.3 Solving State Transition Rate Diagrams .......... 61
  3.5.4 System Performance Characterisation from STRD Solution ... 62
3.6 Theoretical Performance Evaluation Under Different Control Schemes ... 64
  3.6.1 Theoretical Evaluation of System Under PNG_A ...... 65
  3.6.2 Theoretical Evaluation of System Under PNG_N ...... 68
  3.6.3 Theoretical Evaluation of System Under PNG_1 ...... 70
  3.6.4 Theoretical Evaluation of System Under PNG_H ...... 73
3.7 Performance Evaluation ............................... 75
  3.7.1 Performance Under Variable Load ................. 76
  3.7.2 Performance Under Variable H .................... 81
3.8 Conclusions .......................................... 84
4 Controlling the System During Load Transitions

4.1 Introduction .................................................. 95
4.2 Problem Formalisation ....................................... 96
4.3 Transitional Precise Notice Generator Derivation .......... 98
  4.3.1 Swapping to $PNG_A$ via $TNG_A$ ...................... 98
  4.3.2 Swapping to $PNG_N$ via $TNG_N$ .................... 99
  4.3.3 Swapping to $PNG_1$ via $TNG_1$ ................... 100
  4.3.4 Swapping to $PNG_H$ via $TNG_H$ .................. 103
4.4 Transitional Notice Generator Example .................... 106
4.5 Summary ..................................................... 107

5 Conclusions and Future Work ............................... 109

5.1 Conclusion .................................................. 109
5.2 Future Work ................................................ 111

Bibliography .................................................. 117
Abstract

This thesis studies the management of task execution precision in dynamic soft real-time computer systems which utilise imprecise computations. The tasks arrive randomly during run-time and are to be executed with some balance between accuracy and response-time. In the task model, each task consists of two parts: a mandatory part, and an optional part. When the mandatory part of a task is computed and the optional part is discarded we call this an imprecise computation. We use the fact that a precise computation tasks longer than a imprecise computation to implement control strategies. A Notice Generation (NG) approach is utilised to decide on task execution precision. In this method, the notice associated with a task reflects the level of computation that the task will receive when the task is executed. We analyse four different NGs and show that two of these form upper and lower bounds on the performance of the other two.

To study the performance of the system controlled by the four NGs, we propose performance metrics that reflect the requirements of the system in task waiting time and task computation quality. State transition rates are used to analyse the systems using a method derived as part of this work, and performance metrics are calculated. Our results reveal that some of the NGs proposed here can keep the mean waiting time low when the load is high which contrasts with a standard system where performance degrades significantly.

With these four NGs at our disposal six methods are proposed to deal with transient overload in the system. The overload is modelled by a significant increase in the mean arrival rate of tasks to the computation system. One type of NG is used when the load was high and one other type when the load is low. The complexity here is in preserving the state of the system when changing from one NG to another. Results show that this method provides significant performance benefits over single scheme methods that ignore transient overloads.
List of Figures

1.1 Value of Task Completion in Hard and Soft Real–Time Systems . . . . . . . 4

2.1 Two Examples of the Exponential Distribution . . . . . . . . . . . . . . . 10
2.2 Two State STRD Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.3 Example of Alignment of States to Grid . . . . . . . . . . . . . . . . . . . . 15
2.4 Example of Permitted and Prohibited Branches . . . . . . . . . . . . . . . 16
2.5 The Rectangular Structure of $N$ . . . . . . . . . . . . . . . . . . . . . . . . 18
2.6 Branches Due to Diagonal $Q_j(-1)$ . . . . . . . . . . . . . . . . . . . . . . 21
2.7 Branches Due to Upper Triangular $Q_j(-1)$ . . . . . . . . . . . . . . . . 21
2.8 Two Non–Rectangular State–Transition–Rate Diagrams . . . . . . . . . . 24
2.9 Correctable Non–Rectangular State–Transition–Rate Diagram . . . . . . 24
2.10 An Example of Absent Node Matrices . . . . . . . . . . . . . . . . . . . . . 25
2.11 An Example of Branches Added to STRD . . . . . . . . . . . . . . . . . . . 26

3.1 System Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
3.2 Quality of Task Result in a Two–Level Imprecise Computation System 32
3.3 Stages of Task Existence Within the System . . . . . . . . . . . . . . . . . . 35
3.4 General Queue Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
3.5 $GSS_A$, The Generalised System Structure Under $PNG_A$ . . . . . . . . 47
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6</td>
<td>$GSS_N$, The Generalised System Structure Under $PNG_N$</td>
<td>48</td>
</tr>
<tr>
<td>3.7</td>
<td>$GSS_1$, The Generalised System Structure Under $PNG_1$</td>
<td>48</td>
</tr>
<tr>
<td>3.8</td>
<td>$GSS_H$, The Generalised System Structure Under $PNG_H$</td>
<td>49</td>
</tr>
<tr>
<td>3.9</td>
<td>State Transition Diagram for $M/M/1$ System</td>
<td>53</td>
</tr>
<tr>
<td>3.10</td>
<td>State Transition Rate Diagram of System Under $PNG_N$</td>
<td>60</td>
</tr>
<tr>
<td>3.11</td>
<td>State Transition Rate Diagram of System Under $PNG_A$</td>
<td>60</td>
</tr>
<tr>
<td>3.12</td>
<td>State Transition Rate Diagram of System Under $PNG_1$</td>
<td>60</td>
</tr>
<tr>
<td>3.13</td>
<td>State Transition Rate Diagram of System Under $PNG_H$</td>
<td>61</td>
</tr>
<tr>
<td>3.14</td>
<td>Two Stage Server Model of System Under $PNG_A$</td>
<td>65</td>
</tr>
<tr>
<td>3.15</td>
<td>$W_n$ Under Variable Load with $R = 0.4$</td>
<td>86</td>
</tr>
<tr>
<td>3.16</td>
<td>$W_n$ Under Variable Load with $R = 0.8$</td>
<td>87</td>
</tr>
<tr>
<td>3.17</td>
<td>$Q_c$ Under Variable Load with $R = 0.4$</td>
<td>88</td>
</tr>
<tr>
<td>3.18</td>
<td>$Q_c$ Under Variable Load with $R = 0.8$</td>
<td>89</td>
</tr>
<tr>
<td>3.19</td>
<td>$T_u$ Under Variable Load with $R = 0.4$</td>
<td>90</td>
</tr>
<tr>
<td>3.20</td>
<td>$T_u$ Under Variable Load with $R = 0.8$</td>
<td>91</td>
</tr>
<tr>
<td>3.21</td>
<td>$W_n$ Under Variable $H$ with $R = 0.4$</td>
<td>92</td>
</tr>
<tr>
<td>3.22</td>
<td>$Q_c$ Under Variable $H$ with $R = 0.4$</td>
<td>93</td>
</tr>
<tr>
<td>3.23</td>
<td>$T_u$ Under Variable $H$ with $R = 0.4$</td>
<td>94</td>
</tr>
<tr>
<td>4.1</td>
<td>Generalised System States</td>
<td>97</td>
</tr>
<tr>
<td>4.2</td>
<td>Offered Load versus Time for Simulation</td>
<td>107</td>
</tr>
<tr>
<td>4.3</td>
<td>Queue Length versus Time for System Under $PNG_A$</td>
<td>108</td>
</tr>
<tr>
<td>4.4</td>
<td>Queue Length versus Time for System Under $PNG_A$ and $PNG_H$</td>
<td>108</td>
</tr>
</tbody>
</table>
Declaration.

This thesis has been submitted to the Faculty of Engineering at the University of Adelaide for examination in respect of the Degree of Master of Engineering Science (by research).

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University, and to the best of the author’s knowledge and belief contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

The author hereby consents to this thesis being made available for photocopying and for loans as the University deems fitting, should the thesis be accepted for the award of the Degree.

Paul Alexander
April 3, 1995
Acknowledgements.

The Australian Research Council is gratefully acknowledged for making this work possible. I would like to thank my supervisor Dr Cheng Chew Lim for his enthusiasm and guidance throughout this work. I would also like to thank Dr. Wei Zhao of Texas A & M University for his expert assistance. A special thanks to Allison for her continued encouragement and support.

Paul Alexander
Symbol Table.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>time (continuous)</td>
</tr>
<tr>
<td>$s(t)$</td>
<td>state of system at time $t$</td>
</tr>
<tr>
<td>$P[\cdot]$</td>
<td>probability of event</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>mean state-transition-rate/arrival rate</td>
</tr>
<tr>
<td>$S$</td>
<td>set of all system states</td>
</tr>
<tr>
<td>$\mid \cdot \mid$</td>
<td>cardinality of set</td>
</tr>
<tr>
<td>$q_{A,B}$</td>
<td>transition rate between states $A$ and $B$</td>
</tr>
<tr>
<td>$Q$</td>
<td>set of all transition rates</td>
</tr>
<tr>
<td>$P(t)$</td>
<td>set of state occupancy probabilities</td>
</tr>
<tr>
<td>$\Delta t$</td>
<td>a small increment in time</td>
</tr>
<tr>
<td>$L_A$</td>
<td>total leaving rate for state $A$</td>
</tr>
<tr>
<td>$P_A(t)$</td>
<td>occupancy probability with time for state $A$</td>
</tr>
<tr>
<td>$E$</td>
<td>set of all differential-difference equations</td>
</tr>
<tr>
<td>$N$</td>
<td>set of all system states $(i,j$ state identifiers)</td>
</tr>
<tr>
<td>$n_{i,j}$</td>
<td>state label</td>
</tr>
<tr>
<td>$N$</td>
<td>2-D matrix containing all state labels</td>
</tr>
<tr>
<td>$P$</td>
<td>2-D matrix containing all state probabilities</td>
</tr>
<tr>
<td>$Q$</td>
<td>4-D matrix containing all transition rates</td>
</tr>
<tr>
<td>$Q_j$</td>
<td>set of 3 matrices describing transitions into column $j$</td>
</tr>
<tr>
<td>$Q_j(m)$</td>
<td>2-D matrix describing transitions into column $j$ from column $j + m$</td>
</tr>
<tr>
<td>$P_j$</td>
<td>steady state probability for states in column $j$</td>
</tr>
<tr>
<td>$L_j$</td>
<td>vector describing total leaving rate for states in column $j$</td>
</tr>
<tr>
<td>$[L_j]_{i}$</td>
<td>total leaving rate for state $(i,j)$</td>
</tr>
<tr>
<td>$n$</td>
<td>index of last column in $P$ (and $N$)</td>
</tr>
<tr>
<td>$T_j$</td>
<td>transfer matrix from $P_j$ to $P_n$</td>
</tr>
<tr>
<td>$A_{dirn,i}$</td>
<td>number of nodes absent in row $i$ of $N$</td>
</tr>
<tr>
<td>$N_{dirn}$</td>
<td>number of task absent in $N$ on side specified by $dirn$</td>
</tr>
<tr>
<td>$N'_{dirn}$</td>
<td>$N$ with states in all $(i,j)$ locations</td>
</tr>
<tr>
<td>$\frac{1}{\mu_m}$</td>
<td>mean mandatory part computation time</td>
</tr>
<tr>
<td>$\frac{1}{\mu_o}$</td>
<td>mean optional part computation time</td>
</tr>
<tr>
<td>$1/\mu$</td>
<td>mean precise computation time</td>
</tr>
<tr>
<td>$\rho$</td>
<td>offered load/mean number of seconds of work arriving per second</td>
</tr>
<tr>
<td>$N_s$</td>
<td>number of tasks in the computation system</td>
</tr>
<tr>
<td>$R$</td>
<td>accuracy of imprecise vs. precise computation</td>
</tr>
<tr>
<td>$E_i$</td>
<td>computation system event label</td>
</tr>
</tbody>
</table>
\( T_i(A) \) time at which task \( A \) experiences event \( E_i \)

\( QP_i \) system position \( i \), \( QP_0 \) is processor, \( QP_1 \) is head of queue

\( N_p \) number of tasks in the computation system with precise notices

\( H \) control threshold for \( N_s \)

\( PNG_i \) Precise Notice Generator \( i \)

\( ICP_i \) Imprecise Computation Predictor for system under \( PNG_i \)

\( \mathcal{G} \) set of all tasks in computation system

\( \mathcal{P} \) set of tasks with precise notices (\( \subseteq \mathcal{G} \))

\( \mathcal{I} \) set of tasks with imprecise notices (\( \subseteq (\mathcal{G} \setminus \mathcal{P}) \))

\( \mathcal{W} \) set of tasks without any form of notice (\( = (\mathcal{G} \setminus \mathcal{P}) \setminus \mathcal{I} \))

\( g \) vector of set sizes (\( = \{ |\mathcal{W}|, |\mathcal{I}|, |\mathcal{P}| \} \))

\( \emptyset \) the empty set

\( GSS_i \) Generalised System State under \( PNG_i \)

\( U \) utilisation of processor

\( N_q \) mean number of tasks in the queue

\( N_u \) mean number of tasks in the system without notices

\( W_n \) mean waiting time of tasks (normalised)

\( Q_c \) mean computation time of tasks (normalised)

\( T_u \) mean time until notice generation (normalised)

\( s \) system state vector (\( = (N_p, N_s, \text{mode}) \) or \( = (N_p^{\text{mode}}, N_s) \))

\( \text{mode} \) execution mode of processor

\( U(s) \) utilisation of processor when system is in state \( s \)

\( N_q(s) \) number of tasks in queue when system is in state \( s \)

\( N_u(s) \) number of tasks in system without notices when system is in state \( s \)

\( t_p \) random variable describing precise computation time of tasks

\( t_m \) random variable describing mandatory part computation time of tasks

\( t_o \) random variable describing optional part computation time of tasks
Chapter 1

Introduction

Real-time systems have the property that tasks are time-critical. A task is defined to be an object that requires the use of a resource for some period of time. This resource may be a communication link or a computational processor, etc. If the resource time required by different tasks is random the system cannot guarantee the completion time of any given task. When this is the case we have a soft real-time system [1]. Conversely, when the time required by tasks is deterministic the system can provide absolute guarantees about the timeliness of task completion. Specialising our consideration to computation tasks, imprecise computation has been proposed to facilitate the timeliness of task completion.

Soft real-time computation systems are characterised by the need for some balance between task result precision and task response time. Examples may include the calculation of the movement of a robotic limb on some arbitrary surface, or object recognition systems. In this work we study the execution of tasks in a soft real-time system.

The requirements of a soft real-time system can be met by allowing tasks to be executed to varying levels of precision. We call a system with this property an imprecise computation system. The variable precision feature of an imprecise computation system is implemented by structuring the tasks so that each task has a mandatory
part and an optional part. The computation of the mandatory part will yield an approximate result, and the subsequent computation of the optional part will yield an exact result. Tasks with the property that increased computation time leads to increased accuracy are termed monotone [2]. In our system we have restricted tasks to have only two possible levels of computation: imprecise (mandatory only) and precise (mandatory and optional). Obviously the precise computation of a task will require more processor time than the imprecise computation of a task. It is this property that is utilised in the implementation of our control strategies.

To meet the aims of the system, the notice generators may start to issue imprecise notices for tasks when the number of tasks in the system is greater than a certain threshold. When tasks are imprecisely computed the queue length should start to decrease, and the response time of tasks should improve. By choosing the threshold correctly the prescribed response-time and accuracy balance can be met. It is this philosophy that underlies the control schemes presented in his thesis.

Imprecise and approximate computations have received considerable attention from the real-time systems and control community in recent years. This work has revealed many interesting and promising approaches in providing scheduling flexibility and enhancing dependability in dynamic real-time control systems [3, 4, 5, 6, 7, 8]. An overview of the imprecise computation technique is presented in [9, 10]. Many scheduling algorithms for both hard and soft real-time systems have been devised [3, 11, 12, 13]. When the arrival pattern of tasks is completely determined and the attributes of task execution are also completely determined it is possible to develop powerful algorithms to schedule the system [14, 15]. Some of these algorithms create a static schedule under the assumption that the task attributes do not change in time. Such a system is said to be static. In a dynamic system where the task attributes change with time the schedulers must adapt a dynamic approach, where task execution timeliness cannot be guaranteed. A dynamic system is considered in this thesis.

Platforms upon which imprecise computation systems are built are discussed in [16, 17, 18, 19, 20, 21, 22], and real-time programming considerations appear in [23, 24].
The scheduling of real-time task on parallel distributed systems has also been consider in [25, 26, 27, 28, 29, 30]. Many have made use of approximate computations in time-constrained applications [31, 32, 33, 34, 35] where the schedulers are often allowed to preempt tasks in the queue. In our system we will disallow task preemption. When tasks arrivals are not deterministic transient overloads can occur. Methods to suppress this overload were discussed in [36]. The imprecise computation technique has also been used to provide fault tolerance in [37], and AI-based control in [38].

In our dynamic non-preemptive soft real-time system the decisions relating the tasks computation precision are made during run-time because task attributes are not known before system initialisation. The algorithms that decide on task execution precision are called notice generators. This name arises from the fact that these algorithms generate notices that reflect the computation precision that each task will receive when it reaches the processor. Each notice generated by a notice generator pertains to a specific task within the system.

This study is concerned with the design, analysis and application of four control algorithms that make decisions about task computation precision in a dynamic imprecise computer system. There are two major aims in the design of the controller for such a system where the precision (and therefore computation time) of individual tasks is allowed to vary. The first is to provide some prescribed balance of accuracy and speed over all tasks. This aim contrasts with the primary aim of hard real-time systems, where the tasks must meet their deadlines at any expense [39]. To explain, one can think of the value of the completion of the soft real-time task diminishing as time passes, whereas the value of a task result is zero after the deadline in a hard real-time system. A comparison between the value of task completion versus time is shown for hard and soft real-time systems in Figure 1.1.

In current imprecise systems, the external system may not know what result accuracy it will get from the task until its completion. It would be desirable from the external system point of view if some information about the service to be received could be provided before task execution. The second objective of the control algorithm
design aims to offer such information. To meet this aim the controller decides on the type of service that a given task will receive as soon as possible after task arrival. This decision takes the form of a notice which indicates the type of processing that a task will receive. These notices are useful not only for the external system but also for control purposes. Once a notice has been generated for a task the external system could proceed to execute some pre-processing to receive the task result.

To analyse operation and performance of the system under these schemes, the computer system is assumed to consist of a first-in first-out (FIFO) queued processor, and the system dynamics are represented by a two dimensional state-transition-rate diagram (STRD) [40]. Explicit solutions for some of the schemes proposed in this thesis are not possible due to STRD complexities. By solving the STRD numerically, the state probabilities are obtained. This can readily be achieved using an algorithm developed in conjunction with this work [41]. To assess the ability of the system to meet its aims, performance metrics, calculated from the state probabilities are used. The three features of service that are addressed are the mean waiting time of a task, the mean computation time of a task, and the mean time without notice. These metrics of performance will enable a system administrator to pick between the control schemes proposed.

In some circumstances overload control using a threshold approach only may not
be sufficient to control the overload [36]. In many practical systems the arrival rate of tasks varies, depending on the operation requirements of the external system. For example, during system start-up the load on the processor may be excessive. To combat these load surges we define techniques to swap from one control scheme to another during run-time, thus allowing a more dynamic approach to system control than the single scheme approach. By matching the control scheme to the load characteristics even the most difficult loading conditions should be controllable.

In the study of overload handling we model task arrivals by two arrival rates: a normal arrival rate, $\lambda_n$, when the external system is in its normal state, and a higher rate, $\lambda_h$, when the external system is in an abnormal operation state, such as during a system start-up process.

When the system is severely loaded the aims of the system differ from the aims outlined above.

1. If the load is normal ($\lambda_n$) the system aims to provide some prescribed balance of accuracy and response time.

2. If the load is high ($\lambda_h$) the system aims to keep response time bounded by reducing computational accuracy.

The first aim reflects the needs of a soft real-time system which is characterised by implicit time constraint on the task response time, with accuracy versus execution-time tradeoffs as an important system feature [6]. We use intra-scheme control to meet the system aims during periods of normal load, and we use inter-scheme control to manage the high load case by switching to different schemes. In this way we provide standard service during normal load periods while still managing excessive load periods by switching to a special alternative control scheme. The complexity involved in making this scheme switch during run-time is significant. Methods are derived that allow the switching between any of the four schemes described in this thesis.

The rest of this thesis is organised as follows. In Chapter 2 we introduce the
theoretical machinery required in this thesis, including a new algorithm used to solve state-transition-rate diagrams. Chapter 3 defines and analyses the performance of four schemes individually. The application of the four schemes to a system in a dynamic way is studied in Chapter 4. A summary of the work in this thesis is then given in Chapter 5.
Chapter 2

Solving for System State Probability

2.1 Introduction

In this chapter we study the state probability solution of a system. The state of a system is defined by a particle in motion which may occupy one of a discrete set of positions at any given time. We allow this particle to change positions at any point in time. The position of the particle at time $t$ is interpreted as the state of the system at time $t$. Insight into the behaviour of the system can be obtained if the state probabilities can be obtained. This solution requires details such as the state-transitions that are possible, and the probability of these transitions. The transient behaviour of the system can be analysed if the state probabilities can be found as a function of time. This type of analysis requires an initial position of the system to be defined. The solution will then describe the probability that the particle will occupy a particular state for all time for all states. Sometimes in system design it is important to understand the steady state performance of the system. This type of analysis is independent of the initial condition of the system. The state probabilities returned by this type of analysis are constants. Specifically they are the probability of finding the system in a particular state if you
wait for an infinite amount of time after starting the system.

In order to solve for the state probabilities (steady state, or transient) we need to assume the system can be modelled as a Markov process [40] where the history of the system is embodied by the current state. There is a rich mathematical framework to work with once a system is defined to be a Markov process. This background is examined in Section 2.2.

The system considered in this thesis is sufficiently complicated as to exceed the capabilities of well known explicit results. In Section 2.3 we describe methods to find the steady state probabilities of a Markov process numerically.

### 2.2 Queueing System State Theory

Let us define the state of a continuous time system at time $t$ to be $s(t)$. In general, $s(t)$ can take on any value for any discrete alphabet, e.g., colours, digital temperatures, populations, etc. In analysing systems it is of interest to obtain the probability that the system is in some particular state (say $A$) at some particular time (say $t$). Specifically we would like to find $P[s(t) = A]$, e.g., $P[s(t) = \text{green}]$.

The state of the system changes according to some function. This function may have as its arguments some external process and the history of the system in general. When the history of the system is fully described by the state of the system at the time of the state transition we have a Markov process defined as

**DEFINITION 2.2.1** The process $s(t)$ is Markov if

$$P[s(t) = A_n|s(t_1) = A_1, s(t_2) = A_2, \ldots s(t_{n-1}) = A_{n-1}] = P[s(t_n) = A_n|s(t_{n-1}) = A_{n-1}]$$

where the sequence $\{t_i\}$ can take any value with $t_1 < t_2 < \ldots t_n$. The Markov process is therefore in some sense memoryless.
In this thesis we will consider only Markov processes. The critical feature of Markov processes is that tight restrictions are now placed on the distributions describing the time that the system spends in a particular state before moving to some other state. We call the time spent in one state before moving to another the \textit{time-until-transition}. These restrictions follow directly from Definition 2.2.1. The following Theorem defines this restriction.

\textbf{THEOREM 2.2.1} In a continuous time Markov process the time-until-transition must be exponentially distributed for every possible transition because the exponential distribution is the only memoryless continuous time distribution.

The proof of Theorem 2.2.1 can be found in [40]. The exponential distribution has the form

\[ a(t) = \lambda e^{-\lambda t} \]  

\text{(2.1)}

where \( \hat{\lambda} = \frac{1}{\lambda} \) is the mean of the distribution. The interpretation of the mean value \( \hat{\lambda} \) is the mean time-until-transition before a particular state-transition. We can associate \( \lambda \) with the mean state-transition-rate, which follows from the fact the reciprocal of time is rate. Consider Figure 2.1 where the exponential probability density functions are plotted for \( \lambda = 1, \frac{1}{2} \) as \( c(t) \) and \( b(t) \) respectively.

Let \( b(t) \) be the pdf of the time-until-transition for the transition from state \( A \) to state \( B \). Similarly, let \( c(t) \) be the pdf of the time-until-transition for the transition from state \( A \) to state \( C \). We can now ask the question, how long does the system stay in state \( A \) on average, and also what is the probability that the system goes to state \( B \) rather than state \( C \). To solve these problems we require the memoryless property of the exponential distribution. At any time the pdfs for the time-until-transitions will be as shown in Figure 2.1 due to this memoryless property. The solutions to the above questions can now be found by considering the two pdfs. Now that we have established the importance of the exponential distribution we move on to consider the solution of a system where the time-until-transition is exponentially distributed for all possible transitions.
Figure 2.1: Two Examples of the Exponential Distribution

We define all possible states of the system to be elements of the set $\mathcal{S}$, so $s(t) \in \mathcal{S}$, $\forall t$, where $|\mathcal{S}|$ is the cardinality of $\mathcal{S}$. We also define the time–until–transition to be exponentially distributed for all possible state transitions. The exponential distribution describing the time that the system stays in state $A$ before moving to state $B$ is assumed to have mean $1/q_{A,B}$, $\forall \{A, B\} \in \mathcal{S}, A \neq B$. $q_{A,B}$ is then interpreted as the transition rate between from state $A$ to state $B$. We put all of these transition rates into the set $\mathcal{Q}$. To solve the system we must calculate $\mathcal{P}(t) = \{P_A(t), \forall A \in \mathcal{S}\}$ which is the set of all state probabilities as a function of time. The system dynamics are therefore defined by $\mathcal{Q}$ with the solution being $\mathcal{P}(t)$. We now ask the question: If we know $\mathcal{Q}$ and $\mathcal{P}(t)$ what is the probability that the system is in state $A$ at time $t + \Delta t$? The solution to this problem depends on the following Lemma.

**Lemma 2.2.1** The probability of a particular state–transition $A \rightarrow B$ in a some time $\Delta t$ is $q_{A,B}\Delta t$ in the limit as $\Delta t \rightarrow 0$.

Lemma 2.2.1 allows us to derive the probability that the system leaves a particular state,

**Lemma 2.2.2** The probability that the system leaves state $A$ in some time interval
We define a quantity called the *total leaving rate* for a particular state to be

$$L_A = \sum_{\substack{B \in S \\text{ s.t. } B \neq A}} q_{A,B}$$

and therefore we can rewrite the probability that the system leaves state $A$ in some time interval $\Delta t$ as $L_A \Delta t$.

There are only two ways that the system could be in state $A$ at time $t + \Delta t$. These are

**Event 1** $s(t) = A$ and the system does not move from state $A$ in the next $\Delta t$ seconds, or

**Event 2** $s(t) = B, B \neq A$ and the system moves from state $B$ to state $A$ in the next $\Delta t$ seconds.

Note that we assume that $\Delta t$ is small enough to disallow any other transitions. The sum of the probabilities of these two events is $P_A(t + \Delta t)$. The probability of Event 1 is

$$P[\text{Event 1}] = P_A(t) (1 - L_A \Delta t)$$

and the probability of Event 2 is

$$P[\text{Event 2}] = \Delta t \sum_{\substack{B \in S \\text{ s.t. } B \neq A}} P_B(t) q_{B,A}$$

Adding these two probabilities yields

$$P_A(t + \Delta t) = P_A(t) (1 - L_A \Delta t) + \Delta t \sum_{\substack{B \in S \\text{ s.t. } B \neq A}} P_B(t) q_{B,A}$$

$$= P_A(t) - P_A(t) L_A \Delta t + \Delta t \sum_{\substack{B \in S \\text{ s.t. } B \neq A}} P_B(t) q_{B,A}$$
Now we rearrange this equation to yield the time derivative of $P_A(t)$. This is done by subtracting $P_A(t)$ from both sides then dividing through by $\Delta t$ to get

$$\frac{P_A(t + \Delta t) - P_A(t)}{\Delta t} = \sum_{B \in S, B \neq A} P_B(t)q_{B,A} - P_A(t)L_A$$  \tag{2.3}$$

Now by taking the limit as $\Delta t \to 0$ the left-hand side of Equation (2.3) represents the derivative of $P_A(t)$ with respect to $t$. So

$$\frac{dP_A(t)}{dt} = \sum_{B \in S, B \neq A} P_B(t)q_{B,A} - P_A(t)L_A$$  \tag{2.4}$$

All that remains is to solve this differential–difference equation for $P_A(t)$. We can write equivalent equations for every other state in $S$ by considering events 1 & 2 for each state, thus generating a set of equations $E$ which contains $|S|$ equations. We therefore have as many equations as the are states in $S$, where each is of the form shown in Equation (2.4). The solution to this equation set will be discussed in the next Section.

The generation of Equation (2.4) was quite involved. We now present a method to derive the equations set efficiently. By the construction of a state–transition–rate diagram (STRD) we can obtain the differential–difference equation set by writing cut equations around each of the states in the diagram. The diagram is drawn by associating with each node in the drawing a state in $S$. Once this is done we can draw directional branches between nodes that reflect possible state–transitions. For every element of $Q$ there will be a branch in the STRD. Moreover, associated with each branch will be the transition–rate as determined by $q_{A,B}$, where the branch is directed from state $A$ to state $B$. An example where $S$ contains two states is shown in Figure 2.2.

![Two State STRD Example](image-url)
In this example \( Q = \{ q_{A,B} = 0.8, q_{B,A} = 0.6 \} \). We can see that \( Q \) completely specifies the STRD, structure is indicated by the subscripts of \( q \) and value is specified by the value of \( q \). A cut equation is a equation that equates terms that have probability flow as their units. Let us clarify the concept of probability flow. By interpreting \( P_A(t) \) to be the quantity of some entity at position \( A \) we can see that \( P_A(t)q_{A,B} \) is nothing more than the flow of entity from position \( A \) to position \( B \). By equating entity with probability and position with state we get the concept of probability flow between states rather than entity flow between positions. A cut equation equates the flow of probability into a state minus the flow of probability out of the state with the rate of change of probability in the state. Again we can think in terms of entities and positions to clarify the conservation equation. We call these equations cut equations because the net flow across the cut around the state into the state must be equal to the rate at which the probability is increasing in the state. Let us now specifically derive each of the three parts of a general cut equation. The first part we consider is the rate of change of probability in state \( A \) which is simply the time derivative of \( P_A(t) \). Now for the flow of probability out of state \( A \). This quantity is simply the value of the probability in state \( A \) times the rate at which probability flows out of state \( A \). Similarly the flow of probability into state \( A \) is the sum of all flows into state \( A \) where each flow is given by the value of the probability in state \( B \) times the rate as which probability flows from state \( B \) into state \( A \). Now by constructing this cut equation for state \( A \) we get

\[
\frac{dP_A(t)}{dt} = \sum_{\substack{B \in S \\ B \neq A}} P_B(t)q_{B,A} - P_A(t)L_A
\]

which is exactly the same equation that we derived via the transition probability approach in Equation (2.4).

In order to obtain the equation set \( (\mathcal{E}) \) for a system that can be modelled as a Markov process we construct a state–transition–rate diagram and write a cut equation for each state in the diagram. We end up with as many equations as there are states in the STRD. We have \( |S| \) equations in \( |S| \) unknowns. If this set of equations is independent then we can find the solution. However, \( \mathcal{E} \) is not an independent set,
fortunately any subset of $\mathcal{E}$ of size $|\mathcal{E}| - 1$ is independent \[40\]. In other words by throwing away one equation from $\mathcal{E}$ we have a independent set of equations to solve for $|\mathcal{S}|$ unknowns. We are one equation short! There is however one equation that we have not considered yet. The conservation of probability equation

$$\sum_{A \in \mathcal{S}} P_A(t) = 1, \forall t \quad (2.5)$$

is an equation in the $|\mathcal{S}|$ unknowns, thus we have $|\mathcal{S}|$ equations constituting an independent set $\mathcal{E}'$.

So far we have considered the system dynamics over the time interval $(0, \infty)$. If we assume the system is homogeneous the the system may have a steady state solution in the limit as $t \to \infty$. The homogeneity constraint requires that the transition–rates do not vary with time. Provided that a steady state solution exists we can find it by setting the time derivative of all state probabilities to zero. Specifically, the steady state solution is found by setting

$$\frac{dP_A(t)}{dt} = 0 \quad (2.6)$$

for all equations in $\mathcal{E}$. Now we are left with difference equations rather that differential–difference equations. The methodology used to actually solve $\mathcal{E}$ in the steady state form is considered in the next section.

### 2.3 Solving State Transition Rate Diagrams

In this section we present an algorithm that finds $\mathcal{P}$ in the steady state. The algorithm places some structure in $\mathcal{S}$ and restricts $\mathcal{Q}$. First we address the structure of $\mathcal{S}$.

#### 2.3.1 Nomenclature

Previously $\mathcal{S}$ consisted of a set of state labels, where the labels purpose was to uniquely identify each state in $\mathcal{S}$. We now map $\mathcal{S}$ into the set $\mathcal{N}$ where the general element of
2.3. Solving State Transition Rate Diagrams

\( \mathcal{N} \) is \( n_{i,j} \), thus forcing the state label to consist of an integer 2-tuple, where each state is now uniquely identified by a unique pair of integers. For example, if \( \mathcal{S} \) was the set \{red, green, blue\} then \( \mathcal{N} \) could be \( \{n_{0,1}, n_{2,2}, n_{1,0}\} \). The cardinality of \( \mathcal{N} \) is the same as the cardinality of \( \mathcal{S} \). The other parameters of the system are similarly mapped. The transition-rate set now has entries of the form \( q_{n_{i,j},n_{k,l}} \) defining the transition-rate from state \( n_{i,j} \) to state \( n_{k,l} \). The steady state probability set now contains elements of the form \( P_{n_{i,j}} \) where the 2-tuple \((i, j)\) is the state identifier. We can simplify this nomenclature by realising that because every \((i, j)\) is unique we can define a 2-dimensional matrix \( P \) that contains the value of \( P_{n_{i,j}} \) in the element corresponding to the \( j \)th element in row \( i \). So \( P_{i,j} = P_{n_{i,j}} \), where it is apparent that we have transformed the state name into an element index. We can perform a similar transformation on \( N \) where \( N_{i,j} \) refers to the state with identifier \( n_{i,j} \). \( Q \) can also be transformed. We define \( Q_{i,j,k,l} \) to be the transition-rate from \( N_{i,j} \) to \( N_{k,l} \). The nodes of the STRD are now aligned to a 2 dimensional grid, where each node has a similar two dimensional grid representing the \(|\mathcal{S}| - 1\) possible branches terminating at the node. One possible \( N \) is shown in Figure 2.3

![Figure 2.3: Example of Alignment of States to Grid](image)

In summary we have transformed to a 2-dimensional grid as shown in Table 2.1.
### Table 2.1: Transformation of General System Model to Grid Aligned System Model

<table>
<thead>
<tr>
<th>Set</th>
<th>General Label</th>
<th>Grid Aligned Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>All states</td>
<td>$S$</td>
<td>$N$</td>
</tr>
<tr>
<td>All branches</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>

With this structural detail established we know consider the restrictions due to the algorithm.

#### 2.3.2 Restriction on STRD Due to Algorithm

The algorithm places restrictions on the scope of state-transitions. State transitions that span more than one column of the $N$ matrix are not permitted. For example consider Figure 2.4

![Figure 2.4: Example of Permitted and Prohibited Branches](image)

The branches marked with the cross are not permitted but all other branches are. This STRD is not actually solvable by the algorithm because there are some other restrictions that must be satisfied that this STRD violates. A branch emanating from
a node in column \( j \) must terminate at either column \( j - 1, j, \) or \( j + 1 \). This restriction simplifies \( Q \) because we need not be so general as to allow an arbitrary state to connect to some other arbitrary state. The first simplification is that we define \( Q_j \) to be a set of three matrices that describe all transitions from columns \( j - 1, j, \) and \( j + 1 \) into each node in column \( j \). Specifically the following three matrices describe all branches terminating in column \( j \) of \( N \).

\[
Q_j(-1) \quad \text{describes branches originating in column } j - 1, \\
Q_j(0) \quad \text{describes branches originating in column } j, \text{ and} \\
Q_j(1) \quad \text{describes branches originating in column } j + 1
\]  

Row \( i \) of \( Q_j(-1) \) describes the transitions from column \( j - 1 \) into node \( N_{i,j} \), similarly row \( i \) of \( Q_j(0) \) and \( Q_j(1) \) describe the transitions from columns \( j \) and \( j + 1 \) into node \( N_{i,j} \) respectively. The entries in \( Q_j \) are state-transition-rates. Consider an example from Figure 2.4: If the transition-rate from \( N_{1,4} \) to \( N_{3,3} \) was 0.6, then \([Q_3(1)]_{3,1} = 0.6\). This information is not stored in any other \( Q \) matrix. The flow of probability from \( N_{1,4} \) to \( N_{3,3} \) is \( 0.6 \times P_{1,4} \) due to this branch.

The algorithm to find \( P \) from \( Q \) operates on rectangular \( N \). We assume \( N \) has \( m \) row and \( n \) columns. As we show later the algorithm can easily be adapted to solve non-rectangular \( N \), but for the description of the algorithm we assume \( N \) to be \( m \times n \) with nodes at every position within the rectangle. The general structure of \( N \) is now shown in Figure 2.5.

When \( N \) has the dimensions shown in Figure 2.5 there are \( 3n - 2 \) matrices to describe the STRD. This follows from the fact that column 1 only requires \( \{Q_1(0), Q_1(1)\} \) and column \( n \) only requires \( \{Q_n(-1), Q_n(0)\} \) to describe branches terminating in their respective columns, and all of the other \( n - 2 \) columns require 3 matrices because they have columns of nodes on both sides.

We now describe a structure to determine the flow of probability out of a state. We begin with a lemma.

**Lemma 2.3.1** Any node in a non-trivial \( N \) will have at least one emanating branch.
Figure 2.5: The Rectangular Structure of $N$

**PROOF 2.3.1** If this is not the case the $P$ has a trivial solution because once the system gets into a state with no branch leaving that state the system will remain there forever.

This Lemma has the implication that $L_A \neq 0$, $\forall A \in N$. The structure used to describe flow out of states in column $j$ is $L_j$ where $L_j$ is a vector of size $m$ containing elements $[L_j]_i = L_A$ where $A$ is a state identifier and the 2-tuple $(i,j)$ identifies the state $A$. $L_A$ was defined is Equation (2.2).

The information contained in $L_j$ is derivable from the set $\{Q_j(-1), Q_j(0), Q_j(1)\}$ and is therefore redundant. We define $L_j$ to simplify the description and execution of the solution technique. Let us also define the steady state probability vector $P_j$ to be a column vector consisting of all steady state probabilities in column $j$ of $N$. $P_j$ is therefore indexed from 1 to $m$, with the $i$th entry corresponding to $P_{i,j}$. 
2.3.3 Algorithm

Now we have put the parameters of the problem into a useful form we proceed to describe the solution method. Let us write a steady state cut equation for each of the \( m \) nodes in column \( j \). The resulting system of \( m \) equations is

\[
L_j P_j = Q_j(-1)P_{j-1} + Q_j(0)P_j + Q_j(1)P_{j+1}
\]  

where the left hand side of Equation (2.10) is a vector of probability flows out of column \( j \), and the right hand side is the vector of probability flows into nodes in column \( j \). We now subtract \( Q_j(0)P_j \) from both sides of Equation (2.10) to get

\[
(L_j - Q_j(0)) P_j = Q_j(-1)P_{j-1} + Q_j(1)P_{j+1}
\]  

This matrix Equation assumes that columns to the left and right of column \( j \) exists. When \( j = 1 \), \( Q_j(-1) \) is not defined, and similarly when \( j = n \), \( Q_j(1) \) is not defined. The starting point for the algorithm is the vector of \( m \) cut equations for column \( n \), which is derived from Equation (2.11) by setting \( Q(1) = 0 \).

\[
(L_n - Q_n(0)) P_n = Q_n(-1)P_{n-1}
\]  

Equation (2.12) expresses the vector \( P_{n-1} \) in terms of the vector \( P_n \) via a matrix function. This function can be found easily if \( Q_n(-1) \) is invertible. Let us assume the invertibility of \( Q_n(-1) \) for now and proceed. We can write \( P_{n-1} \) in terms of \( P_n \) by rearranging Equation (2.12) as follows

\[
P_{n-1} = [Q_n(-1)]^{-1} (L_n - Q_n(0)) P_n
\]  

Now let us write the vector cut equation for column \( n - 1 \) of \( N \). From Equation (2.11) we get

\[
(L_{n-1} - Q_{n-1}(0)) P_{n-1} = Q_{n-1}(-1)P_{n-2} + Q_{n-1}(1)P_n
\]

We now rearrange to get \( P_{n-2} \) by itself on the left hand side, now assuming the invertibility of \( Q_{n-1}(-1) \)

\[
P_{n-2} = [Q_{n-1}(-1)]^{-1} [(L_{n-1} - Q_{n-1}(0)) P_{n-1} - Q_{n-1}(1)P_n]
\]
We can now substitute for $P_{n-1}$ using Equation (2.13), yielding $P_{n-2}$ as a function of $P_n$.

\[
P_{n-2} = \left[Q_{n-1}(-1)\right]^{-1} \cdot \left[\left(L_{n-1} - Q_{n-1}(0)\right) \left[Q_n(-1]\right]^{-1} (L_n - Q_n(0)) P_n\right] - Q_{n-1}(1)P_n
\]

We can continue this process until we reach column 1, where we must consider boundary conditions again. We must assume the invertibility of $Q_j(-1)$ at each step. By generalising we can write $P_j$ in terms of $P_n$ via a transfer matrix $T_j$ that pre-multiplies $P_n$ to give $P_j$. Specifically we define $T_j$ such that

\[
P_j = T_j P_n
\]

The $T_j$'s are generated using the following algorithm.

**ALGORITHM 2.3.1 (T)** To find $P$ in terms of $P_n$ we find $T_j$ via

the following algorithm.

\[
T_n = \text{the } m \times m \text{ Identity matrix}
\]

\[
T_{n-1} = [Q_n(-1)]^{-1} (L_n - Q_n(0))
\]

For $j = n - 1$ to 2 step -1

\[
T_{j-1} = [Q_j(-1)]^{-1} \left(L_j T_j - Q_j(0)T_j - Q_j(1)T_{j+1}\right)
\]

By generating these $T_j$ we have expressions for all of the $P_j$ vectors in terms of the vector $P_n$.

The invertibility of $Q_j(-1)$, $2 \leq j \leq n$ is assumed in the construction of $T$. We now show some examples of STRD structure that yield invertible $Q_j(-1)$. The diagonal matrix with nonzero entries is always invertible. When $Q_j(-1)$ has this type of structure the branches are connections between $N_{i,j-1}$ and $N_{i,j}$, $\forall i$. An example of this type of connection is shown in Figure 2.6. Note that there are not constraints on $Q_j(0)$ or $Q_j(1)$ in terms of invertibility. In most queueing systems this type of diagonal $Q_j(-1)$ will occur.

Another type of matrix that is guaranteed to be invertible is the triangular matrix with nonzero diagonal. In this case the branches terminating at $N_{i,j}$ come from the set
2.3. Solving State Transition Rate Diagrams

In general the invertibility of $Q_j(-1)$ is not seen as a restriction on the applicability of this algorithm.
We now return to the solution method. When the cut equation around column 2 is written we get \( P_1 \) in terms of \( P_n \) as follows

\[
P_1 = T_1 P_n \tag{2.15}
\]

However we still have not cut around states in column 1. When we write the vector cut equation for column 1 we have no \( Q_1(-1) \) because the is no column of \( N \) to the left of column 1. However, we still get \( P_1 \) in terms of \( P_n \) as follows

\[
L_1 P_1 = Q_1(0) P_1 + Q_1(1) P_2
\]

\[
\Rightarrow P_1 = [L_1 - Q_1(0)]^{-1} Q_1(1) P_2 = [L_1 - Q_1(0)]^{-1} Q_1(1) T_2 P_n \tag{2.16}
\]

which is not the same set of equations obtained in (2.15) because

\[
T_1 \neq [L_1 - Q_1(0)]^{-1} Q_1(1) T_2 \tag{2.17}
\]

The most obvious difference between the left hand side and right hand side of Equation (2.17) is that \( T_1 \) does not contain \( L_1 \) which is a matrix completely unrelated to any matrix in \( T_1 \). We conclude that these two Equations are independent equations in the same two variables. In other words we should be able to solve for \( P_1 \) and \( P_n \) through the use of Equations (2.16) and (2.15). We combine these two equations to get

\[
T_1 P_n - [L_1 - Q_1(0)]^{-1} T_2 P_n = 0
\]

\[
(T_1 - [L_1 - Q_1(0)]^{-1} T_2) P_n = 0 \tag{2.18}
\]

Unfortunately this Equation has trivial solution \( P_n = 0 \). What can we do? In Equation (2.18) we have written \( m \) equations in terms of \( m \) unknowns. The fact that saves us is that one of these equations is redundant [40] as explained in Section 2.2. Let us remove an arbitrary equation by defining a matrix \( U \) as follows

\[
U = [T_1 - [L_1 - Q_1(0)]^{-1} T_2] \text{ with one arbitrary row removed.}
\]

Then from Equation (2.18) the equation set is now

\[
UP_n = 0
\]
where \( \mathbf{0} \) is now of dimensionality \( m - 1 \). So we really have \( m - 1 \) in \( m \) unknowns. But now we have insufficient equations for the \( m \) unknowns. There is still one independent equation we have not utilised. Since the system can only be in one state at any given time, the sum of all the state probabilities must be one.

\[
\sum_{j=1}^{n} \sum_{i=1}^{m} P_{i,j} = 1 \tag{2.19}
\]

Every \( \mathbf{P}_j \) can be written in terms of \( \mathbf{P}_n \), so Equation (2.19) can be expressed as

\[
\sum_{j=1}^{n} \sum_{i=1}^{m} [T_j \mathbf{P}_n]_i = 1 \\
= \mathbf{S}^T \mathbf{P}_n
\]

where \( \mathbf{S} \) is a column vector of size \( m \) with the \( k \)th element being

\[
S_k = \sum_{i=1}^{m} \sum_{j=1}^{n} T_{i,j} \]

Now we have an extra equation in the \( m \) unknowns of \( \mathbf{P}_n \), giving us \( m \) independent equations in \( m \) unknowns. The solution follows simply by the row-wise concatenation of \( \mathbf{U} \) with \( \mathbf{S}^T \). The problem is now to solve the following matrix equation,

\[
\begin{bmatrix} \mathbf{U} & \mathbf{S}^T \end{bmatrix} \mathbf{P}_n = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}
\]

where \( \mathbf{U} \) is \((m - 1) \times m\), \( \mathbf{S}^T \) is \(1 \times m\), and \( \mathbf{0} \) is \((m - 1) \times 1\). \( \mathbf{P}_n \) is found as

\[
\mathbf{P}_n = \begin{bmatrix} \mathbf{U} \\ \mathbf{S}^T \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}
\]

Now that we have \( \mathbf{P}_n \) all of the other probabilities follow via Equation (2.14).

The algorithm detailed here required that the states in \( \mathbf{S} \) be aligned to a grid \( \mathbf{N} \) that is rectangular. The assumption is that the rectangle has a state at every location. In the general system this may not be the case. We now discuss the solution of irregular \( \mathbf{N} \).

### 2.3.4 Adapting the System to Algorithm Requirements

In this section we describe a method to adapt irregular state-transition-rate diagrams (STRDs) to the algorithm detailed in Section 2.3.3. In describing irregular STRDs we
will consider the rows and columns of \( N \). In an irregular \( N \) at least one of nodes in \( N \) does not exist in the system. Note that the perimeter of \( N \) may have a rectangular shape but there may be a node missing in the middle of \( N \). Two examples of irregular \( N \) are shown in Figure 2.8

![Non-Rectangular State-Transition-Rate Diagrams](image)

Figure 2.8: Two Non–Rectangular State–Transition–Rate Diagrams

We have dropped the state identifiers because structure is important here and not state identification. The specific irregularity that we can correct is the case when \( N \) is equivalent to a rectangle with truncated rows. By this we mean some arbitrary number of nodes are removed from the left side and right side of every row of a rectangle to yield our correctable \( N \). An example of such a correctable \( N \) is given in Figure 2.9.

![Correctable Non-Rectangular State Transition Rate Diagram](image)

Figure 2.9: Correctable Non–Rectangular State Transition Rate Diagram

Mathematically we can define a structure of a correctable \( N \) by quantifying the number of absent nodes at each end of every row in \( N \). Let the number of nodes absent
on the left side of row \(i\) be \(A_{\text{left},i}\) and the number of nodes absent on the right side of row \(i\) be \(A_{\text{right},i}\). So the number of nodes in any row of \(N\) is therefore \(n - A_{\text{left},i} - A_{\text{right},i}\), where these nodes form a continuous sub-row. For example in Figure 2.9 \(A_{\text{left},5} = 2\) and \(A_{\text{right},5} = 1\), so the number of nodes in row 5 must be \(8 - 2 - 1 = 5\). We construct two matrices that when added together for the complement of \(N\). One specifies absent nodes on the left of \(N\) called \(N_{\text{left}}\) and the other specifies absent nodes on the right of \(N\) called \(N_{\text{right}}\). Examples of these two matrices for the \(N\) of Figure 2.9 are shown in Figure 2.10.

\[
\begin{array}{c}
  \text{Figure 2.10: An Example of Absent Node Matrices} \\
\end{array}
\]

\[N_{\text{left}} + N + N_{\text{right}}\] forms a full rectangular matrix of nodes called \(N'\). It is on this matrix that the algorithm will operate.

Now that we have formalised the correctable problem, how do we correct it? We have formed a full rectangle of nodes called \(N'\). Now the question arises, how to connect these new nodes \((N_{\text{left}}, N_{\text{right}})\) to the original nodes \((N)\). The problem with these absent nodes is that they result in singularities [42] in the \(Q\) matrices. This could be overcome by reducing the problem size to the number of nodes in each column dynamically as the algorithm moves from column \(n\) to column 1. Rather than do this we artificially create non-singular \(Q\) matrices by placing dummy branches in the STRD. Specifically, we connect adjacent nodes in a particular row of \(N_{\text{left}}\) with a forward-backward pair of branches of the same rate. This has the effect of placing entries in row \(i\) of the \(Q\) matrices where there would have been an all zero row before, thus removing the singularity. We perform a similar operation of the \(Q\).
matrices applicable to the right side of \( N \). Finally where row \( i \) of \( N \) and row \( i \) of \( N_{\text{left}} \) or \( N_{\text{right}} \) meet we also place a pair of equi-rate branches, thus replicating the probability of the original node in the nodes of \( N_{\text{left}} \) and \( N_{\text{right}} \). The added branches in the STRD can be seen in Figure 2.11

![Figure 2.11: An Example of Branches Added to STRD](image)

We have replicated the edge probabilities of \( N \) in the nodes of \( N_{\text{left}} \) and \( N_{\text{right}} \). At the same time all of \( Q \) matrices will now be non-singular. Now the algorithm can operate on the modified \( Q \) matrices with the structure given be \( N' \). The only modification left is to make sure that the sum of state probabilities are normalised to 1. The nodes in \( \{ N_{\text{left}}, N_{\text{right}} \} \) are not actual system states so their probabilities must be excluded in Equation (2.5).

A user guide to software that implements the algorithm detailed here can be found in [41].

### 2.4 Summary

In this section we have discussed the theoretical bases for the analysis in this thesis. A framework for the representation of systems by state-transition-rate diagrams (STRDs) was given. We then discussed a method that can be utilised to solve for the steady state probabilities of the system. Some restrictions on possible state-transitions were necessary in order to guarantee the algorithm’s operation. These restrictions were
shown to be not severe in terms of queueing system dynamics. The structure of the STRD had to have certain geometrical properties. We defined a method whereby a class of STRDs that did not conform to this geometry could also be solved by the algorithm.
Chapter 3

Imprecise Computation System
Control Schemes

3.1 Introduction

In this Chapter we introduce and analyse four schemes for deciding on task execution precision in a soft real-time imprecise computation system. Each of these schemes has been designed to offer a different class of performance. The controllable object is task execution precision where the controller dynamically decides on the execution precision of each task in the system. The control mechanism consists of the generation of task execution precision notices that indicate the quality of the task result once the task is executed.

The system consists of a single processor serving a FIFO queue. The imprecise computation system is modelled by tasks with two computational parts. The mandatory part must be executed and the optional part may be executed after the mandatory part to improve result accuracy.

The schemes that decide on task execution precision are termed Precise Notice Generators (PNGs). Tasks are issued with precise notices sometime after arrival if
the PNG decides that the task should be executed precisely. If a task arrives at
the processor with a precise notice then it is executed precisely, if no precise notice
accompanies the task it is executed imprecisely.

The PNGs described in this Chapter aim to provide some balance between task
execution precision and task response-time. From a system point of view it is advanta-
geous to provide the external system with the type of execution that a task will receive
before the task begins execution. The precise notices are generated to implement the
control, and so nothing can be done to speed up precise notice generation. However,
imprecise computation results if a task arrives at the process with no notice from the
PNG. It is possible to predict this occurrence and therefore supply imprecise notices
to the external system before the task reaches the processor. Imprecise notices are
predicted by schemes called imprecise computation predictors (ICPs), that are based
on the type of PNG in operation, and that predict the imprecise computation of tasks.

The analysis technique utilised here is new. In the past explicit solution to state-
transition-rate diagrams describing the system under a particular control scheme have
been derived [12]. Other work related to queueing theory can be found in [43, 44,
45]. The work of Lim is applicable to static control schemes where the control is
not a function of the state of the system. After constructing the state–transition–rate
diagram for the system under all of the four PNGs proposed in this Chapter, we use the
algorithm derived in Chapter 2 to solve for the steady state probabilities of the system.
Having found these probabilities we then calculate metrics of system performance that
allow us to compare the characteristics of the four PNGs.

The rest of this chapter is organised as follows. In Section 3.2 we define the models
used in this analysis. The notice generators, both precise and imprecise, are described
in Section 3.3. The metrics used to analyse the performance of the system under each
of these notice generators are presented in Section 3.4. The system is defined in terms
of a state variable and state–transition–rate diagrams are presented in Section 3.5).
Theoretical performance bounds are calculated in Section 3.6 and numerical results
are evaluated in Section 3.7. A summary of the four PNGs presented in the chapter is
then provided in Section 3.8.

3.2 Task and System Models

The system is characterised as a Markov server consisting of a FIFO queue and a single processor. Tasks are filed in the queue pending execution in the processor. We assume tasks arrive at the system according to a Poisson process [40] with a mean arrival rate $\lambda$, and are scheduled on a first come first serve basis. We assume that more than one arrival in such a small time interval is impossible. The schematic diagram of the system is shown in Figure 3.1. The external system is the source of tasks and the sink of task results.

![Figure 3.1: System Structure](image)

The tasks consist of two separate sections: a mandatory part that must be completed, and an optional part that may be completed. The completion of the mandatory part will yield an approximate but sufficiently accurate result, called an imprecise result. Following the completion of the mandatory part the optional part of the task may be executed, which uses the results from the imprecise computation to produce a precise result. Thus there are two levels of task computation. An example of task result quality versus time curve is shown in Figure 3.2.

As can be seen in the Figure partial completion of either part is not possible. It is the function of the notice generator to decide on the level of computation that each task will receive. The notice generator is part of the controller shown in Figure 3.1.
The time taken to compute the mandatory part of the task is assumed to have a mean value of $1/\mu_m$. Similarly, the optional computation time is assumed to have mean $1/\mu_o$. The computation time of the mandatory part is assume to be independent of the computation time of the optional part. It follows that the precise computation time has mean

$$1/\mu \triangleq 1/\mu_m + 1/\mu_o.$$ 

$1/\mu$ can be interpreted as the average number of seconds required by precise task execution. We have defined these execution times in this reciprocal manner for reasons that will become apparent in the sequel. The time unit seconds is chosen without loss of generality to be the time base. The mean arrival rate $\lambda$ can be interpreted as the average number of arrivals per second. We now define $\rho$ as the average number of seconds of processor work arriving per second as

$$\rho = \lambda(1/\mu_m + 1/\mu_o)$$

(3.1)

If $\rho > 1$ then there is more work being offered than can be computed precisely. We can relate queue length to offered load in the following Lemma.

**LEMMA 3.2.1** *Assuming the system executes all tasks precisely and that $\rho > 1$ then the average queue length will be infinite.*
3.2. Task and System Models

PROOF 3.2.1 The average number of tasks in a system $\bar{N}_s$ can be computed via the Pollaczek-Khinchin mean-value formula [40] as

$$\bar{N}_s|_{\rho<1} = \rho + \rho^2 \frac{(1 + C_b^2)}{2(1 - \rho)}$$

where $C_b^2$ is a function of the service time distribution only that is always positive. Clearly

$$\lim_{\rho \to 1} \bar{N}_s = \infty$$

Now if we increase $\rho$ beyond 1 the number of tasks in the system must still be infinite.

An imprecise system can reduce the effective load by computing only the mandatory parts of tasks.

To give an indication of the degree of computation accuracy in an imprecise system, we define

$$R = \frac{1/\mu_m}{1/\mu_m + 1/\mu_o}$$

(3.2)

to be the ratio between the imprecise computation time and the precise computation time. $R$ is therefore an indication of the difference in accuracy between an imprecise result and the precise result.

Due to the imprecise computation ability of the system the offered load can be different from the processor load. The offered load is $\rho$, but the processor load is determined by the type of execution that each task receives. If all tasks received precise computation then the processor load is equal to $\rho$. However, if some tasks are executed imprecisely then the processor load is less than $\rho$. The minimum load presented to the processor occurs when all of the tasks are executed imprecisely. In this case the load on the processor is $R\rho$.

Figure 3.3 shows how a task progresses through the system. Major events and the time at which particular events occur for a task are labelled. It can be seen that several events occur instantaneously. An example of a tasks progress through the system would be as follows:
1. Assume task A arrives at the system. This is registered as event $E_1$ at a system level. The time at which task A arrived is $T_1(A)$. For any two tasks their $T_1$ cannot be equal because we disallow bulk arrivals.

2. Task A waits in the queue for some time $T_3(A) - T_1(A)$.

3. Task A leaves the queue. This is registered as event $E_3$.

4. At the same time that the task leaves the queue, it enters the processor and starts mandatory part computation. These events $E_3, E_4$ and $E_5$ all occur at $T_3(A)$.

5. The tasks mandatory part is computed for some time $T_4(A) - T_3(A)$.

6. The task leaves the processor because it has been decided that this task should be imprecisely computed. The events $E_6$ and $E_9$ both occur instantaneously at $T_4(A)$.

We note that the average value of $T_4(A) - T_3(A)$ over all tasks $A$ is equal to $1/\mu_m$ and similarly, the average of $T_5(A) - T_4(A)$ over all tasks $A$ is equal to $1/\mu_o$.

The system controller uses the events shown in Figure (3.3) to implement its control. When more than one event occurs at the same time the events are placed in a FIFO buffer called the Event Monitor Buffer.

A notice generator is responsible for deciding the computation accuracy of each task. It must generate a precision notice for each task before the task enters the processor. This occurs at $T_2(A)$ (not shown in Figure 3.3 where $T_1(A) \leq T_2(A) \leq T_3(A)$). The event corresponding to precise notice generation for a task is $E_2$ and it is not shown in Figure 3.3 because its position in the Figure may vary from task to task. Once this notice has been generated the external system can make the required preparations to receive the task result. When the task enters the processor the notice associated with the task is read and the appropriate part(s) of the task are executed. No alteration to task execution precision is possible after the task leaves the queue. Nor can task execution be stopped once it begins execution. Once the task enters the
3.2. Task and System Models

Figure 3.3: Stages of Task Existence Within the System

processor its future execution is determined solely by the notice it had upon entering the processor. This type of system is known as non-pre-emptive.
3.3 Notice Generating Algorithms

There are two major objectives in our system design. The first is to provide fast and accurate service to the tasks that arrive at the system. The difficulty here is to offer a good computational accuracy that does not take too long to achieve over all tasks. The second aim is to decide on the type of computation that each task will receive as quickly as possible after task arrival while not compromising the control strategy. Note that once a notice has been issued, the precision of that task is decided.

The first objective can be achieved by varying the ratio between the number of tasks computed precisely and the number computed imprecisely. This ratio will vary the average computation time of tasks. The quality of the task result increases with time but the value of the task result diminishes with time. An increase in both quality and value of a task clearly cannot be achieved simultaneously. It is the action of the notice generating algorithms that determine the balance between these two quantities. Whilst maintaining a prescribed balance to meet the first objective, the second aim of the system can be achieved by generating the notices as soon as possible. We will now discuss some detailed issues on how to meet these aims.

Four different notice generators will be considered. Each of these notice generators is completely defined by a rule stating whether or not a task should be executed precisely. Thus we call them Precise Notice Generating algorithms (PNG). If it is decided that a task should be executed precisely then a precise notice is issued for that task. The system that reads task notices and generally controls the movement of tasks throughout the system is called the Task Flow Manager (TFM). When the task arrives at the processor the notice accompanying the task is read by the TFM. If the task has a precise notice then the task is executed precisely, otherwise the task is issued with an imprecise notice and executed imprecisely. The pseudo code for the Task Flow Manager is shown in Algorithm 3.3.1, where $QP_i$ refers to system position $i$. The task at the head of the queue is in $QP_1$ and the task in the processor is in $QP_0$ as shown in Figure 3.4.
3.3. Notice Generating Algorithms

**ALGORITHM 3.3.1 (TFM)** The flow of tasks in the general imprecise computation system controlled as follows

\[
\begin{align*}
\text{IF } E_1 \\
\quad \text{move task into } QP_N, \\
\text{IF } E_9 \text{ OR } E_{10} \\
\quad \text{FOR } i = 1 \text{ to } N - 1 \\
\quad \quad \text{move task in } QP_i \text{ to } QP_{i-1} \\
\text{IF } E_4 \\
\quad \text{compute mandatory part} \\
\text{IF } E_6 \\
\quad \text{IF task has precise notice} \\
\quad \quad \text{compute optional part} \\
\text{ELSE} \\
\quad \text{remove task from processor} \\
\text{IF } E_8 \\
\quad \text{remove task from processor}
\end{align*}
\]

![Diagram of General Queue Structure](image)

Figure 3.4: General Queue Structure

First, consider a very simple precise notice generator that *never* issues precise notices, thus causing all tasks to be execute imprecisely. We call this \(PNG_N\). It has the benefits of simplicity and deterministic notice generation (which satisfies the second aim), but may unnecessarily cause tasks to be computed imprecisely when the system loading is low.
Another very simple method is to issue precise notices to all new arrivals. This precise notice generator is denoted as $PNG_A$. The destiny of tasks under this scheme is also completely determined, thus the external system knows exactly what type of task result to prepare for. Under $PNG_A$ all tasks are computed precisely so the queue length may grow when there are slight increases in load. Thus increasing the response time of tasks.

The control action of $PNG_N$ and $PNG_A$ is independent of the system state, they may not be able to respond in some desired manner to changes in external system dynamics. We have designed two more PNGs that use the system state to decide on the type of notice to generate for a given task. The state variables are:

$N_s$: number of tasks in the system, and

$N_p$: number of tasks in the system with precise notices.

In practice these variables will be readily available. $N_s$ is used as a measure of response time, and $N_p$ to keep track of the number of tasks that have had precise notices generated for them.

The state variables are maintained by the State Variable Manager (SVM). The control executed by the SVM is as follows
ALGORITHM 3.3.2 (SVM) The two state variables in the general imprecise computation system are updated as follows

\[
\begin{align*}
IF & E_1 \\
N_s & = N_s + 1 \\
IF & E_2 \\
N_p & = N_p + 1 \\
IF & E_9 \\
N_s & = N_s - 1 \\
IF & E_{10} \\
N_p & = N_p - 1 \\
N_s & = N_s - 1
\end{align*}
\]

The PNGs use a parameter \( H \) which is a threshold for \( N_s \). They use \( N_s \) and \( H \) to decide on the type of control that should be executed. If \( N_s \leq H \) then the PNG uses one type of control strategy due to the apparent low load; if \( N_s > H \) the PNG uses another type to regulate the load at the processor.

We denote the two notice generators that use state information as \( PNG_1 \) and \( PNG_H \). \( PNG_1 \) will issue a precise notice to task \( A \) if \( N_s \leq H \) at \( T_3(A) \). \( PNG_H \) is similar to \( PNG_1 \) but it will issue a precise notice to a task if \( N_s \leq H \) any time before the task enters the processor, i.e., \( (T_1(A), T_3(A)) \). The difference between \( PNG_1 \) and \( PNG_H \) is the timing of the task computation level decision. \( PNG_1 \) makes its computation level decision for task \( A \) at \( T_3(A) \), whereas \( PNG_H \) makes its decision anywhere in \( (T_1(A), T_3(A)) \). Note, that if the decision to compute some task \( A \) precisely is not made before \( T_3(A) \) then the task is computed imprecisely.

The overall structure of system control is defined by a sequence of processes that takes place when an event occurs, where an event is as defined in Figure 3.3. Once an event is signalled by the Event Monitor the following occurs in order:

1. The task flow manager executes, then
2. The precise notice generating algorithm executes, then

3. The state variable manager executes.

For example, consider the completion of a task, the event monitor will signal $E_9$ or $E_{10}$ depending on the task execution precision, then the task flow manager will move a new task into the processor (if the queue is not empty), this task will not begin execution yet because $E_4$ has not been signalled by the event monitor. The PNG will now execute. Finally the state variables are updated, in this case $N_s$ will be decreased by one, and $N_p$ will decrease by one if the task was precisely computed. The PNG may have issued precise notices, these operation will be registered as event $E_2$ by the event monitor, which will result in $N_p$ being altered when the SVM executes again due to the occurrence of $E_2$.

All four of the PNGs are now described in pseudo code.

**ALGORITHM 3.3.3 ($PNG_N$)** The precise notice generator $PNG_N$ is defined as follows

\[
\text{do nothing.}
\]

**ALGORITHM 3.3.4 ($PNG_A$)** The precise notice generator $PNG_A$ is defined as follows

\[
\text{IF } E_1 \\
\text{issue a precise notice for task in } QP_N.
\]
3.3. Notice Generating Algorithms

**Algorithm 3.3.5 (PNG₁)** The precise notice generator PNG₁ is defined as follows

\[
\text{IF } E₁ \\
\text{IF } N_s \equiv 1 \\
\text{issue a precise notice for task in } QP_{N_s}. \\
\text{IF } E₉ \text{ OR } E_{₁₀} \\
\text{IF } N_s \equiv H \\
\text{issue a precise notice for task in } QP₁.
\]

**Algorithm 3.3.6 (PNG₇)** The precise notice generator PNG₇ is defined as follows

\[
\text{IF } E₁ \\
\text{IF } N_s \leq H \\
\text{issue a precise notice for task in } QP_{N_s}. \\
\text{IF } E₉ \text{ OR } E_{₁₀} \\
\text{IF } N_s \equiv H \\
\text{issue a precise notice for tasks in } \{QP₁,...,QP_{N_s}\}.
\]

We have now completely defined how each task will be executed for each type of PNG due to the fact that if it is not executed precisely it will be executed imprecisely. These four notice generators issue precise notices, thus notifying the external system of the fact that a particular task will be executed precisely. Unfortunately imprecise notices are issued as the task enters the processor, thus forcing the external system to wait until $T₃$ for imprecise notices. Precise notices, on the other hand are generated anywhere between $T₁$ and $T₃$. We see that there is an imbalance between the time until precise notice generation and the time until imprecise notice generation. Fortunately we can correct this imbalance by predicting imprecise computations due to the particular PNG in operation.

We construct four imprecise computation predictors (ICP) that forecast the generation of imprecise notices thus reducing the time that the external system must wait
for imprecise notices. We define an ICP for each of the four PNGs. The two simple PNGs have correspondingly simple ICPs. Consider \( PNG_N \), which results in all tasks being executed imprecisely. Clearly \( ICP_N \) simply generates an imprecise notice for every task when it arrives because the PNG will never cause a task to be precisely computed. Conversely for \( PNG_A \), no task will ever be computed imprecisely so \( ICP_A \) never issues imprecise notices. The ICPs for these two simple PNGs are

**ALGORITHM 3.3.7 (ICP\(_N\))** The imprecise computation predictor when \( PNG_N \) is operating is defined as follows

\[
\text{IF } E_1 \\
\text{issue an imprecise notice for task in } QP_{N}.
\]

**ALGORITHM 3.3.8 (ICP\(_A\))** The imprecise computation predictor when \( PNG_A \) is operating is defined as follows

\[
\text{do nothing.}
\]

The imprecise notice generators for \( PNG_1 \) and \( PNG_H \) are more interesting than \( ICP_N \) and \( ICP_A \).

Let us consider the system operating under \( PNG_1 \). The ICP for this scenario is \( ICP_1 \). To derive this ICP we begin with the following lemma

**LEMMA 3.3.1** When \( PNG_1 \) is operating a task will be executed imprecisely if when the task leaves the queue there are at least \( H \) tasks in the system.

**PROOF 3.3.1** The proof follows from the converse of the \( PNG_1 \) rule. \( \square \)

So we have the conditions under which a task will be computed imprecisely under \( PNG_1 \). Now we ask, how can we predict these imprecise computations before they occur? From Lemma (3.3.1) we see that we need to predict that \( N_s \) will be at least \( H \) when the task leaves the queue. The following lemma provides us with this tool.
LEMMA 3.3.2 $N_s$ will be at least $H$ for task $A$ when it leaves the queue if $N_b(A,t) \geq H$ for any $t \in (T_1(A), T_3(A))$, where $N_b(A,t)$ is defined to be the number of tasks queued behind task $A$ at time $t$.

PROOF 3.3.2 Because the queue is FIFO once tasks are queued behind a particular task they will remain there. Therefore once there are at least $H$ tasks queued behind some task $A$ at some time $t \in (T_1(A), T_3(A))$ there will be at least $H$ tasks queued behind task $A$ at time $t' \geq t$.

The method to predict imprecise computations under $PNG_1$ is to find tasks that satisfy the condition of Lemma (3.3.2). We are therefore looking for tasks that have at least $H - 1$ tasks queued behind them. To ensure that imprecise notices are issued as soon as possible we must check for this condition every time there is a new arrival. This new arrival may have caused $B_b(A,T)$ to become more than $H - 1$ for some task $A$ that has no precise notice. The imprecise computation predictor for a system under $PNG_1$ follows.

ALGORITHM 3.3.9 (ICP) The imprecise computation predictor when $PNG_1$ is operating is defined as follows

IF $E_1$

IF $N_s > H$

issue an imprecise notice for task in $QP_{N_s - H}$

Let us consider the system operating under $PNG_H$. The ICP for this scenario is $ICP_H$. To derive this ICP we begin with the following lemma

LEMMA 3.3.3 When $PNG_H$ is operating a task will be executed imprecisely if for all of its queued life there are at least $H$ tasks in the system.

PROOF 3.3.3 Consider some arbitrary task $A$. To prove this Lemma the system must have at least $H$ tasks in it before task $A$ arrives. Let $N^a_s$ be the number of tasks
in the system immediately that have precise notices immediately before the task arrives. So \( N^*_s > H \) tasks will be computed before task \( A \) is computed, \( N^*_p \leq N^*_s \) of which will be precisely computed. The important point is that if \( N_s \) remains greater than \( H \) while these \( N^*_s \) task are computed there will be no tasks with precise notices left in the system. This is due to two points, 1) all tasks with precise notices that were in the system have been computed, and 2) no precise notices are issued because \( N_s \) has remained greater than \( H \). Now because no tasks have precise notices and the task that arrived is about to be computed it will be computed imprecisely.

So we have the conditions under which a task will be computed imprecisely under \( PNG_H \). Now we ask, how can we predict these imprecise computations before they occur? From Lemma (3.3.3) we see that we need to predict that \( N_s \) will remain greater than \( H \) for the tasks queued life. The following lemma provides us with this tool.

**Lemma 3.3.4** Consider some task \( A \). When \( PNG_H \) is operating \( N_s \) will remain greater than \( H \) in the interval \((T_1(A), T_3(A))\) if the following occurs

1. \( N_s \geq H \) when \( A \) arrives, and then
2. \( N_s \) remains greater than \( H \) up until some time \( t \leq T_3(A) \), and
3. \( N_b(A, t) \geq H \).

where \( N_b(A, t) \) is defined to be the number of tasks queued behind task \( A \) at time \( t \).

**Proof 3.3.4** Clearly 1 and 2 imply that \( N_s \geq H \) has been satisfied up until time \( t \) for task \( A \). Now because the queue is FIFO once tasks are queued behind a particular task they will remain there. Therefore once there are at least \( H \) tasks queued behind task \( A \) there will be at least \( H \) tasks queued behind task \( A \) as \( A \) leaves the queue.

The method to predict imprecise computations under \( PNG_H \) is to find tasks in the queue that satisfy all three conditions of Lemma (3.3.4). \( PNG_H \) has the property
that if $N_s$ ever falls below $H$ then all the tasks in the system are issued with precise notices. The converse of this rule leads us to the following Lemma.

**Lemma 3.3.5** Any task with no precise notice when the system is under PNG_H must satisfy conditions 1 and 2 of Lemma 3.3.4.

**Proof 3.3.5** The proof follows from the converse of the PNG_H rule.

A task that has no precise notice under PNG_H has satisfied conditions 1 and 2 of Lemma 3.3.4. So if this task also satisfies condition 3 of Lemma 3.3.4 this task will be imprecisely computed. We are therefore looking for tasks with no precise notice that have at least $H - 1$ tasks queued behind them. To ensure that imprecise notices are issued as soon as possible we must check condition 3 of Lemma 3.3.4 every time there is a new arrival. This new arrival may have caused condition 3 to become satisfied for a task that has no precise notice. Under PNG_H the first $N_p - 1$ tasks in the queue have precise notices, so the oldest un-notified task will be in $QP_{N_p}$. The task in $QP_{N_p}$ will have at least $H - 1$ tasks queued behind it if $N_s - N_p \geq H$. The imprecise computation predictor for a system under PNG_H follows.

**Algorithm 3.3.10 (ICP_H)** The imprecise computation predictor when PNG_H is operating is defined as follows

\[
\text{IF } E_1 \\
\text{IF } N_s - N_p \geq H \\
\text{issue an imprecise notice for task in } QP_{N_s - H}
\]

The overload handling ability of a PNG is critically dependent on its ability to cause imprecise computation by not issuing precise notices. The only PNG that has no way of reducing the load experienced by the processor is PNG_A. Under this scheme all tasks are executed precisely. The other three schemes have the option of not issuing precise
notices. We define $\rho_{\text{max}}$ to be the absolute maximum offered load that the system can handle. $\rho_{\text{max}}$ is tabulated in Table 3.1 for the system under each of the four schemes.

<table>
<thead>
<tr>
<th>PNG</th>
<th>$\rho_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNG$_A$</td>
<td>1</td>
</tr>
<tr>
<td>PNG$_N$</td>
<td>$1/R$</td>
</tr>
<tr>
<td>PNG$_I$</td>
<td>$1/R$</td>
</tr>
<tr>
<td>PNG$_H$</td>
<td>$1/R$</td>
</tr>
</tbody>
</table>

Table 3.1: Absolute Maximum Offered Load for System Under Each PNG

The operation of a PNG and its corresponding ICP are grouped in one notice generating algorithm called a Notice Generator (NG). We can generalise the state of the system under each of the four NGs. We know that the system state is completely defined by the PNG, the number of tasks in the system and the number of tasks with precise notices. This information is enough to determine the exact notice profile of tasks within the processor and the queue. The profile of tasks within the system may be generalised over all PNG, $N_p$ and $N_s$ as follows.

We view the general system to consist of a set of tasks $\mathcal{G}$ with three subsets defined by the type of notice that a tasks possess. Under all PNGs the following is true. Starting from the head of the system (the processor) we find a set of tasks with precise notices $\mathcal{P}$, then after these we will find a set of tasks with imprecise notices $\mathcal{I}$, then finally a set of tasks without any notice of any kind $\mathcal{W}$. So the set of all tasks is the union of $\mathcal{P}$, $\mathcal{I}$ and $\mathcal{W}$.

$$\mathcal{G} = \{\mathcal{W}, \mathcal{I}, \mathcal{P}\}$$

(3.3)

The number of tasks in each of the sets $\mathcal{W}$, $\mathcal{I}$ and $\mathcal{P}$ is $|\mathcal{W}|$, $|\mathcal{I}|$ and $|\mathcal{P}|$ respectively. We define $\mathbf{g}$ to be a vector of three integers equal to the size of each of the sets in $\mathcal{G}$. Specifically,

$$\mathbf{g} \triangleq (|\mathcal{W}|, |\mathcal{I}|, |\mathcal{P}|)$$
Now because all tasks in the system either have a notice or they do not we have

\[ N_s = |\mathcal{G}| = |\mathcal{W}| + |\mathcal{I}| + |\mathcal{P}| \]  

(3.4)

We note that if \(|\mathcal{P}| \geq 1\) then the processor will be executing a task with a precise notice. Also, if \(|\mathcal{P}| = 0\) and \(|\mathcal{I}| \geq 1\) then the processor will be executing a task with an imprecise notice.

We now study the particulars of \(\mathcal{G}\) under each of the NGs.

### 3.3.1 Structure of System State Under PNG\(A\)

Clearly since PNG\(A\) always issues precise notices \(\mathcal{W}\) and \(\mathcal{I}\) will be empty. We denote the empty set as \(\emptyset\). Therefore

\[ \mathcal{G} = \{\emptyset, \emptyset, \mathcal{P}\} \]

This size of each of the sets in \(\mathcal{G}\) is shown in Figure (3.5), where we see that the system contains \(|\mathcal{G}| = |\mathcal{P}|\) tasks all with precise notices.

<table>
<thead>
<tr>
<th>(\mathcal{W})</th>
<th>(\mathcal{I})</th>
<th>(\mathcal{P})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(</td>
</tr>
</tbody>
</table>

Figure 3.5: GSS\(A\), The Generalised System Structure Under PNG\(A\)

### 3.3.2 Structure of System State Under PNG\(N\)

Clearly since PNG\(N\) never issues precise notices and ICP\(N\) always issues imprecise notices to every task \(\mathcal{W}\) and \(\mathcal{P}\) will be empty. Therefore

\[ \mathcal{G} = \{\emptyset, \emptyset, \mathcal{P}\} \]

This size of each of the sets in \(\mathcal{G}\) is shown in Figure (3.6), where we see that the system contains \(|\mathcal{G}| = |\mathcal{I}|\) all with imprecise notices.
3.3.3 Structure of System State Under PNG\(_1\)

We generalise the structure of the system under PNG\(_1\) as a single expression in \(|\mathcal{G}|\) and \(|\mathcal{P}|\). Specifically we can form \(\mathcal{G}\) as follows. We first calculate \(|\mathcal{I}|\) from \(|\mathcal{G}|\) and \(|\mathcal{P}|\)

\[
|\mathcal{I}| = (|\mathcal{G}| - |\mathcal{P}| - H)^+
\]

where

\[
[x]^+ = \begin{cases} 
    |x| & \text{if } x > 0, \\
    0 & \text{if } x \leq 0.
\end{cases}
\]  

(3.5)

Then using this result we know \(|\mathcal{W}|\) due to Equation (3.4)

\[
|\mathcal{W}| = |\mathcal{G}| - |\mathcal{P}| - |\mathcal{I}|
\]

We have written the generalised system state under PNG\(_1\) given \(|\mathcal{G}|\) and \(|\mathcal{P}|\). Note that \(|\mathcal{P}|\) is limited to 1 under PNG\(_1\). This size of each of the sets in \(\mathcal{G}\) is shown in Figure 3.7.

<table>
<thead>
<tr>
<th>(\mathcal{W})</th>
<th>(\mathcal{I})</th>
<th>(\mathcal{P})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>\mathcal{G}</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 3.7: GSS\(_1\), The Generalised System Structure Under PNG\(_1\)
3.3.4 Structure of System State Under $PNG_H$

We can generalise the structure of $\mathcal{G}$ under $PNG_H$ in exactly the same manner as the system under $PNG_1$. The only difference is that the size of $\mathcal{P}$ is limited $H$ not $1$. The size of each of the sets in $\mathcal{G}$ is shown in Figure 3.8.

| $\mathcal{W}$ | $\mathcal{I}$ | $\mathcal{P}$ |
| $|\mathcal{G}| - |\mathcal{P}| - |\mathcal{P}| - H^+$ | $|\mathcal{G}| - |\mathcal{P}| - H^+$ | $|\mathcal{P}|$

Figure 3.8: $GSS_H$, The Generalised System Structure Under $PNG_H$

We note that the number of tasks without notices ($|\mathcal{W}|$) can be identified from $|\mathcal{G}| \equiv (N_s)$ and $|\mathcal{P}| \equiv (N_p)$ for all four NGs.

In this section we have defined four precise notice generators (PNGs) that are event driven. These PNGs are the controllers in the imprecise computation system studied in this thesis. They aim to provide tasks with fair service where a balance between response time and computational accuracy are desirable properties. We also designed four imprecise computation predictors that help to provide information to the external system about the precision of task results as soon as possible after task arrival.

3.4 System Performance Metrics

To analyse the performance of the notice generators proposed in section 3.3, metrics of system performance are required. These metrics indicate the responsiveness and quality of service that the system provides to the tasks. From these metrics a system administrator should be able to choose the best ‘setup’ for their system.

As one important aim of the system is to optimise response time versus accuracy,
the performance metrics that measure how close this aim is met are the task waiting
time $W_n$ and the computation time $Q_c$ defined below. The second aim of the system is
to notify as soon as possible the external system how accurate the result of each task
will be. The metric that measures how well the system meets its second aim is the
notice waiting period $T_n$. These three performance metrics are expressed in terms of
the following three variables:

- $U$, the utilisation of the processor.
- $N_q$, the mean number of tasks in the queue.
- $N_u$, the mean number of tasks in the system without notices generated.

The performance metrics are then expressed in terms of $U, N_q$ and $N_u$ as follows:

- $W_n$, the mean waiting time of tasks, normalised with respect to the mean precise
computation time: This is defined as
  \[ W_n = \frac{\text{mean time in queue}}{\text{mean precise computation time}} \]
  \[ = \frac{W_q}{1/\mu_m + 1/\mu_o} \quad (3.6) \]
  where $W_q$ is the mean time waiting in the queue. This metric is an indication of
  system timing behaviour. It is desirable that the value of this metric be as low
  as possible, to give good task response time. Using Little's result [46] we have
  $W_q = N_q/\lambda$, where $\lambda$ is the arrival rate, we obtain
  \[ W_n = \frac{N_q/\lambda}{1/\mu_m + 1/\mu_o} \]
  \[ = N_q/\rho \quad (3.7) \]
  where $\rho$ is the offered load.

- $Q_c$, the mean computation time of tasks, normalised with respect to the mean
  precise computation time: This is defined as
  \[ Q_c = \frac{\text{mean task computation time}}{\text{mean precise computation time}} \]
  \[ (3.8) \]
This metric indicates the average computation quality over all tasks processed. If all tasks received precise computation then \( Q_c \) would be 1, as the mean task computation time would be equal to the precise computation time. If all tasks were imprecisely computed then the mean task computation time would be \( 1/\mu_m \), and therefore \( Q_c \) would equal \( R \). The higher the value of \( Q_c \), the higher the average task computation quality.

We can derive the mean task computation time in terms of the processor utilisation \( U \) and the arrival rate \( \lambda \). In the steady state there are \( \lambda \) tasks leaving the system per unit time, thus the inter-departure time is \( 1/\lambda \) units of time. It then follows that the mean task computation time = \( U/\lambda \). We can express the normalised mean computation time as

\[
Q_c = \frac{U/\lambda}{1/\mu_m + 1/\mu_o} = \frac{U}{\rho}
\]  

(3.9)

- \( T_u \), the mean time until notice generation, normalised with respect to the mean precise computation time: We can regard the part of the system that contains unnotified tasks as a queueing system. By this we mean that tasks enter the system at some mean rate \( \lambda \), spend some time in the system then leave the system. The mean time spent in this system will be \( N_u/\lambda \) by Little’s result [46]. So therefore

\[
T_u = \frac{\text{mean number of tasks without notices}}{\lambda \times (1/\mu_m + 1/\mu_o)} = \frac{N_u}{\rho}
\]  

(3.10)

This metric indicates the average time from the instant when a task arrives, to the instant when the notice for the task is generated. It is desirable to notify a task as soon as possible, so ideally we would like \( N_u \) to be zero. The worst case value for \( T_u \) is the mean time spent in the queue (\( W_n \)) plus the mean time spent in the processor (\( Q_c \)), i.e., mean system time (\( W_n + Q_c \)).

Table 3.2 shows the performance bounds for each of the metrics. The table is
particularly useful in helping to judge qualitatively which notice generator offers better performance.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Good Performance</th>
<th>Poor Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$W_n \to 0$</td>
<td>$W_n \to \infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$Q_c \to 1$</td>
<td>$Q_c \to 0$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>$T_u \to 0$</td>
<td>$T_u \to W_n + Q_c$</td>
</tr>
</tbody>
</table>

Table 3.2: Qualitative Interpretation of Performance Metric Values

### 3.5 Analysis of Control Schemes

In analysing queue structures controlled by a particular type of algorithm we consider a system state approach. In this method the system attributes of interest are placed in a state vector. For example, in a queueing system, queue length is often an interesting measurement to make. Therefore the queue length would be included as an element of the state vector. Now if we can find the likelihood that the system will occupy a particular state then we have the likelihood that the queue length will be the value given in the state vector pertaining to that state. If we do this over all states we can find the pdf of the queue length because we have a probability for each queue length as determined by state probabilities.

The path from a system description to system performance is detailed in general. The steps in this path are

1. Derive state vector from system controller and system structure, then
2. Construct a state–transition diagram, then
3. Calculate state probabilities by some method from state–transition diagram, then
4. Derive and calculate expressions for system attributes in terms of state probabilities via the state vector, then

5. Calculate system performance metrics in terms of system attributes.

Let us introduce a specific simple example. Consider a system where there is no controller (except for the obligatory Task Flow Manager) and the tasks are single precision. Specifically, let us study the M/M/1 system [40] where tasks arrive with exponentially distributed inter-arrival times with mean $1/\lambda$ and the service time of tasks is also exponentially distributed with mean $1/\mu$. Say we want to investigate the average queue length $N_q$ of this system. Explicit results are available [40], but for demonstration purposes let us continue. We would create a state vector $s \triangleq (N_s)$, where $N_s \in \mathbb{Z}^+$ is the number of tasks in the system. So we have an infinite number of states in the complete representation of the system. In practice the number of tasks in the stable system will not exceed some finite number. The state dynamics of this system are simple. If a task arrives then the new state will be the old state plus 1, similarly when a task completes execution the new state will be the old state minus 1. The state vector cannot change via any other mechanism. We can see that the state-transition diagram would look like that shown in Figure (3.9).

![State Transition Diagram for M/M/1 System](image)

Figure 3.9: State Transition Diagram for M/M/1 System

Now because the inter-arrival times and computation times are exponentially distributed we can solve for the steady state probabilities using state-transition-rate diagram theory [40]. Once we have these probabilities we can calculate the expected or average number of tasks in the queue because we have the occupancy probabilities of the states. We now proceed to construct state-transition descriptions for each of the controllers proposed in Section 3.3.

In Section 3.4 we derived the metrics of performance that we would like to mea-
sure. These metrics are averages, so we need to calculate the steady state probabilities of the system. Also the metrics were expressed in terms of three system attributes $U$, $N_q$ and $N_u$. We need to incorporate these three attributes in the state vector representation of the system.

### 3.5.1 State Description of System

In Section 3.4 we derived the metrics of performance that we would like to measure. As these metrics are averages we need to calculate the steady state probabilities of the system. Also the metrics were expressed in terms of three system attributes, $U$, $N_q$ and $N_u$. We need to incorporate these three attributes in the state vector representation of the system. So we see that the desired performance metrics drive the choice of state vector.

$N_q$ is incorporated into the state vector as $N_s$, the number of tasks in the system. $U$ is a variable that is dependent of $N_s$ and therefore is not included explicitly in the the state vector. In Section (3.3) we have shown that $N_s$ and $N_p$ is enough information to derive the number of tasks without notices in the system. Therefore the two-tuple $(N_s,N_p)$ contains adequate information to derive instantaneous values of the three system attributes. We define the state vector $s$ to be the two-tuple $(N_p,N_s)$. Let us clearly identify $U$, $N_q$ and $N_u$ as average values of system attributes, and $U(s)$, $N_q(s)$ and $N_u(s)$ as the values of system attributes when the system is in a particular state $s$. Now given that we can find the pdf for $s$ we can also find the pdfs for $U(s)$, $N_q(s)$ and $N_u(s)$. $U$, $N_q$ and $N_u$ are calculated as the mean values of these pdfs. We now define the functions $U(s)$, $N_q(s)$ and $N_u(s)$ for all four PNGs.

The $U(s)$ and $N_q(s)$ function are simple,

$$U(s) = (N_s > 0)$$

$$N_q(s) = N_s - 1$$

(3.11)  

(3.12)

These two functions apply irrespective of the type of PNG operating at the time, unlike $N_u(s)$ which does depend on the PNG in operation. $N_u(s)$ is built from the generalised
system states shown in Figures (3.6)–(3.8). Clearly for PNG\textsubscript{N} and PNG\textsubscript{A}, \( N_u(s) = 0 \) because all tasks in these systems are issued with a notice of some kind as soon as they arrive at the system. For PNG\textsubscript{1} we have from Figure (3.7)

\[
N_u(s) = \begin{cases} 
0 & \text{if } N_s \leq 1, \\
N_s - 1 & \text{if } 1 < N_s \leq H, \\
H & \text{if } N_s > H. 
\end{cases} \quad (3.13)
\]

and for PNG\textsubscript{H} from Figure (3.8) we have

\[
N_u(s) = \begin{cases} 
N_s - N_p & \text{if } N_s - N_p \leq H, \\
H & \text{if } N_s - N_p > H. 
\end{cases} \quad (3.14)
\]

We have built the case for the state vector to be defined as \( s = (N_p, N_s) \). In the next section we construct the transitions between the states. We note that the state space is unbounded in the \( N_s \) dimension and bounded to \((0, H)\) in the \( N_p \) dimension.

### 3.5.2 State Transition Rate Diagram Description of System

In order to solve for the steady state probabilities of the system under one of the PNGs, we have created a state vector to describe the system in a way required for performance analysis. In this Section we describe the state–transitions that occur between possible system states. For instance the state–transition \((4,11) \rightarrow (4,12)\) would occur upon a task arrival. We describe the state–transitions by way of a state–transition–diagram, where the system states are nodes and the possible state–transitions are evidenced by directed branches connecting any two nodes.

As shown in Algorithm (3.3.2), the state vector \( s = (N_p, N_s) \) can only be changed by one of four events

1. Task arrival, \( E_1 \), or

2. Precise notice generated for task, \( E_2 \), or
3. Completion of task with precise notice, \( E_9 \), or

4. Completion of task with imprecise notice, \( E_{10} \).

These four events effect \( N_s \) directly because they correspond to tasks entering and leaving the system. How is \( N_p \) effected? The PNGs defined in Section 3.3 obviously effect \( N_p \) because they issue precise notices to tasks. \( N_p \) can also be altered by a task leaving the system after being computed precisely. Let us conceive the idea of a event driven state-transition diagram where the nodes in the diagram are system states, and possible state-transitions are described by branches connecting a pair of states. We can construct this diagram by considering each possible state vector in turn. We then ask what will happen to the system state if one of three events outlined above occurs. When an event causes a transition we place a directed branch between the two states and label the branch with the event that caused the transition. A point to note about these diagrams is that there are branches due to events \( E_1 \), \( E_9 \) and \( E_{10} \) only. Once we have constructed these event driven transition diagrams we have the structure of possible state transitions in the system. From this structure we can build the more useful State-Transition-Rate Diagram (STRD).

To construct an STRD, we simply take the event driven transition diagram and replace the branch labels with transition rates. In Chapter 2 we showed that in a STRD the distribution that describes the time spent in some state \( A \) before the system changes to some other state \( B \) must be exponentially distributed for all states \( A \) and \( B \). The reciprocal of the mean of this distribution is the transition-rate along the branch connecting states \( A \) and \( B \). We now consider the implication of this exponential distribution on the system model.

The arrival of tasks \((E_1)\) occurs at a mean rate of \( \lambda \) tasks per second. Adherence to the exponentially distribution requirement implies that we must set the inter-arrival times of tasks to be exponentially distributed with mean \( 1/\lambda \). The resulting arrival process is the famed Poisson process \([40]\). As for task execution, we need two exponential distributions, one to describe the computation time of a precise computation \((E_{10})\), and one to describe the computation time of an imprecise computation \((E_9)\).
Recalling the system model from Section (3.2), clearly these two distributions would have means $1/\mu$ and $1/\mu_m$ respectively. Define the following random variables

\[
    t_p \triangleq \text{precise computation time, and} \quad (3.15)
\]
\[
    t_m \triangleq \text{mandatory part computation time, and} \quad (3.16)
\]
\[
    t_o \triangleq \text{optional part computation time.} \quad (3.17)
\]

We have stated that $t_p$ and $t_m$ must be exponentially distributed. The exponential distribution has the property that the variance is the square of the mean. Therefore the variance of $t_p$ ($\sigma_{t_p}^2$) is $1/\mu^2$ and the variance of $t_m$ ($\sigma_{t_m}^2$) is $1/\mu_m^2$. Now using the fact that $t_p = t_m + t_o$ we can derive the moments of $t_o$. The random variables $t_m$ and $t_o$ are independent so we can add the variances to get the variance of $t_p$. Therefore we can calculate the variance of $t_o$ as

\[
    \sigma_{t_o}^2 = \sigma_{t_p}^2 - \sigma_{t_m}^2
    = \frac{1}{\mu^2} - \frac{1}{\mu_m^2}
    = \left( \frac{1}{\mu_m^2} + \frac{1}{\mu_m \mu_o} + \frac{1}{\mu_o^2} \right) - \frac{1}{\mu_m^2}
    = \frac{1}{\mu_m \mu_o} + \frac{1}{\mu_o^2} \quad (3.18)
\]

Now since the mean of $t_o$ is $1/\mu_o$ and the variance must be as shown in Equation (3.18), $t_o$ cannot be exponentially distributed. If $t_o$ were exponentially distributed its variance would be the square of the mean ($1/\mu_o^2$). The relevance of this observation is that the distributions of the two computational parts of tasks are different. We have the opinion that this is an undesirable property. We alleviate this problem by considering the precise computation of a task to occur in two stages, where each stage is exponentially distributed computation time.

We introduce a new state variable called the *mode* of the processor. This variable can take two values, $\text{mode} = 0$ corresponds to the processor executing the mandatory part of a task, and $\text{mode} = 1$ corresponds to the processor executing the optional part of a task. When a task is computed precisely the system first goes into state identified by $\text{mode} = 0$. The computation time $t_m$ is exponentially distributed with mean $1/\mu_m$. 
Once the mandatory part is completed a state-transition occurs. The processor starts computing the optional part of the task so mode = 1. We see that the state-transition is due to mode changing from 0 to 1. The computation time of the optional part is $t_o$ which is now exponentially distributed with mean $1/\mu_o$. Once the optional part is completed the system state will change because both $N_s$, $N_p$ and mode will change. The system state vector for analysis is now

$$s \triangleq (N_p, N_s, \text{mode}) \quad (3.19)$$

Note that the operation of the controllers in our system only requires $N_s$ and $N_p$ but because we want to analyse the system with identically distributed mandatory and optional computation times we are forced to introduce mode.

Including mode in the state vector would allow the state-transition-rate diagram representation of more sophisticated controllers. We could perceive controllers that could interrupt task execution at the end of the mandatory part. This type of controller belongs to the preemptive class of controllers. All of the controllers studied in this thesis are non-preemptive.

The system state vector has three elements, two of which have finite domains. Table (3.3) shows the domains of each element.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_s$</td>
<td>$(0, \infty)$</td>
</tr>
<tr>
<td>$N_p$</td>
<td>$(0, H)$</td>
</tr>
<tr>
<td>mode</td>
<td>$(0, 1)$</td>
</tr>
</tbody>
</table>

Table 3.3: Domain of 3-Dimensional State Vector

It is possible to compress any two finite domain variables into one variable also of finite domain such that the mapping in either direction is one-to-one. Let us combine
3.5. Analysis of Control Schemes

$N_s$ and $mode$ to achieve a two dimensional state vector again. Specifically,

$$N_p^{mode} \triangleq 2N_p + mode$$  \hspace{1cm} (3.20)

So if we know $N_p^{mode}$ we know the unique $(N_p, mode)$ pair, and vice versa. We can get $N_p$ and $mode$ from $N_p^{mode}$ as follows

$$N_p = \lfloor N_p^{mode}/2 \rfloor,$$

$$mode = N_p^{mode} \% 2.$$  

where $\lfloor x \rfloor$ is the greatest integral value less than or equal to $x$, and $x \% y$ is the remainder of $x$ divided by $y$. The mapping of $N_p$ and $mode$ to $N_p^{mode}$ can be thought of as array $\rightarrow$ vector mapping, and vice versa. Now we have the final definition of the state vector for analysis.

$$s \triangleq (N_p^{mode}, N_s)$$

The domain of the state vector for analysis is now given in Table (3.4).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_s$</td>
<td>$(0, \infty)$</td>
</tr>
<tr>
<td>$N_p^{mode}$</td>
<td>$(0, 2H + 1)$</td>
</tr>
</tbody>
</table>

Table 3.4: Domain of 2-Dimensional State Vector for Analysis

We are now in a position to present the state-transition-rate diagrams for the system under each of the four PNGs. The node labels are derived from the state vector mapping just described. The structure is derived by considering the effect that arrivals and departures of tasks have on the system. They have a direct influence via $N_s$ and an indirect influence via the PNGs that are sensitive to task arrivals and departures. The branch labels are derived from the distributions of task inter-arrival time, mandatory computation time, and optional computation time. The state-transition-rate diagrams for the system under each of the four PNGs are shown in Figures (3.10) through (3.13)
Figure 3.10: State Transition Rate Diagram of System Under $PNG_N$

Figure 3.11: State Transition Rate Diagram of System Under $PNG_A$

Figure 3.12: State Transition Rate Diagram of System Under $PNG_I$

Some important observations can be made about these STRDs. The STRDs for $PNG_N$ and $PNG_A$ are homogeneous, i.e., a cross-section taken vertically through the STRD is the same no matter where in the horizontal ($N_s$) dimension this section is cut. Similarly, $PNG_I$ and $PNG_H$ are piecewise homogeneous, i.e., any section through column $j < H$ is the same and any section through $j > H$ is the same. This property indicates the dynamic nature of $PNG_N$ and $PNG_H$, and is useful in the representation of the STRDs in computer data structures.
3.5.3 Solving State Transition Rate Diagrams

In Section (3.5.2) we derived the state–transition–rate diagrams (STRDs) for the system under any of the PNGs defined in Section (3.3). In Chapter 2 we detailed a method to solve a general class of STRDs. We will show that the four STRDs shown in Figures (3.10) though (3.13) fit into this class.

To find a solution using the algorithm described in Chapter 2 we must ensure that $Q_j(-1)$ is invertible for $(2 \leq j \leq n)$. To check that this is true for the STRDs described Figures (3.10) though (3.13) is not difficult because there are only a few
unique $Q_j(-1)$s due to the piecewise homogeneity of the STRDs. We now derive $Q_j(-1)$ for the system under each of the precise notice generators. $Q_j(-1)$ describes the connection and transition-rates from nodes in column $j-1$ to nodes in column $j$. Now by observation from the four STRDs we see that the only connections from column $j-1$ nodes to column $j$ nodes are due to task arrivals and therefore are simply horizontally directed to the next column. These type of transitions result in very simple diagonal $Q_j(-1)$ matrices.

When we row-fill these STRDs to make them rectangular we preserve the diagonal $Q_j(-1)$ structure because the added nodes are connected by arrival-like branches, i.e., the connections are strictly horizontal both among the added nodes and between the added nodes and the original nodes. If we choose to connect the added nodes with branches of rate $\lambda$ then the general $Q_j(-1)$ will be $\lambda I_m$ where $I_m$ is the $m \times m$ identity matrix, and $m$ is the number of rows in the STRD.

Now that we have shown each of the four STRDs may be shaped to fit the algorithm in Chapter 2, solution of the state probability matrix $P$ is possible. What do we do with this solution?

### 3.5.4 System Performance Characterisation from STRD Solution

The solution to the STRD yields steady state probabilities. Each of these states corresponds to a particular value of the state vector. The state vector was constructed as $s = (N_p^{mode}, N_s)$ in Section (3.5.2). So given a state $(i, j)$ we have $P_{i,j}$ from the STRD solution. Now if we want to calculate the mean value of $N_s$ ($\bar{N}_s$) we just calculate

$$\bar{N}_s = \sum_{i,j} j P_{i,j}$$

(3.21)

because $N_s$ is mapped directly into the second dimension of the state vector which we have referred to as $j$. By summing over $(i, j)$ we mean that the sum should include every node in the original STRD, i.e., one of Figure (3.10) though (3.13) and not the
row-packed versions. We will specify exact ranges for summation later. To calculate the performance metrics detailed in Section (3.4) we first calculate $U$, $N_q$ and $N_u$ as the average values of $U(s)$, $N_q(s)$ and $N_u(s)$. These three functions are averaged over the pdf of $s$ in a similar way to the example for $N_s$ in Equation (3.21). These three functions were expressed in terms of $s = (N_p, N_s)$ which is slightly different from the state vector that we use for analysis. The state vector for analysis is $s \triangleq (N_p^{\text{mode}}, N_s)$. We can refer to the value of $s$ in general by the $(i, j)$ pair. Let us re-map these functions in terms of the state vector for analysis $(i, j)$.

Clearly, since $N_s \equiv j$ the functions $U(i, j)$ and $N_q(i, j)$ which depend only on $N_s$, are obtained by simply substituting $j$ for $N_s$ in Equations (3.11) and (3.12).

$$U(i, j) = (j > 0)$$

$$< N_q(i, j) = j - 1$$

$N_p$ is obtained as $\lfloor i/2 \rfloor$ so we can obtain $N_u(i, j)$ as follows from Equations (3.13) and (3.14). For $PNG_1$ we have

$$N_u(i, j) = \begin{cases} 0 & \text{if } j \leq 1, \\ j - 1 & \text{if } 1 < j \leq H, \\ H & \text{if } j > H. \end{cases}$$

(3.24)

and for $PNG_H$ we have

$$N_u(i, j) = \begin{cases} j - \lfloor i/2 \rfloor & \text{if } j - \lfloor i/2 \rfloor \leq H, \\ H & \text{if } j - \lfloor i/2 \rfloor > H. \end{cases}$$

(3.25)

and for $PNG_A$ and $PNG_N$

$$N_u(i, j) = 0, \forall i, j$$

(3.26)

We can now find $U$, $N_q$ and $N_u$ from the steady state probabilities as follows. The processor utilisation $U$ is the probability of finding the processor active. The processor is active if it is not idle. But, we know the probability that the processor is idle, this is $P[N_s = 0]$. Therefore

$$U = 1 - P_{0,0}$$

(3.27)
and the average number of tasks in the queue is

\[ N_t = \sum_{i,j} N_t(i,j)p_{i,j} \]
\[ = \sum_{i,j} (j - 1)p_{i,j} \]  

(3.28)

The average number of tasks without notices under \( PNG_1 \) is

\[ N_u = \sum_{i,j} N_u(i,j)p_{i,j} \]
\[ = \sum_{i,j} (j - 1)p_{i,j} + \sum_{i,j > H} HP_{i,j} \]  

(3.29)

and for \( PNG_H \) is

\[ N_u = \sum_{i,j} N_u(i,j)p_{i,j} \]
\[ = \sum_{i,j - \lfloor i/2 \rfloor \leq H} (j - \lfloor i/2 \rfloor)p_{i,j} + \sum_{i,j - \lfloor i/2 \rfloor > H} HP_{i,j} \]  

(3.30)

We now can calculate the performance metrics \( W_n, Q_c \) and \( T_u \) using Equations (3.7) through (3.10).

### 3.6 Theoretical Performance Evaluation Under Different Control Schemes

Theoretical limits for each of the three performance metrics detailed in Section 3.4 can be derived. We find it convenient to group the precise notice generators into two groups. We place \( PNG_A \) and \( PMG_N \) in the Static group because they are not sensitive to state information and, we place \( PNG_1 \) and \( PNG_H \) in the Dynamic group because they do use state information.

Let us first consider the load on these systems. There are some trivial limiting values that need not be explained, i.e., the limit of \( W_n \) as \( \rho \) tends to zero is uninteresting. We first consider the two simple precise notice generators, \( PNG_A \) and \( PNG_N \) in the Static group. It is possible to derive \( W_n \) as a function of \( \rho \) explicitly for these two
3.6. Theoretical Performance Evaluation Under Different Control Schemes

PNGs via well known queueing theory. From a theoretical view point both of these systems may be modelled as a particular type of server, serving a FIFO queue with Poisson arrivals. Also, only one type of precision is executed under either of these PNGs so the tasks are served in a fixed way. Specifically, the offered load to these systems is \( \rho = \lambda / \mu_m + \lambda / \mu_o \). Now if PNG\(_A\) is operating then the server (or processor) load \( \rho_{s,A} = \rho \), however, if PNG\(_N\) is operating the load experienced by the server is \( \rho_{s,N} = \lambda / \mu_m = R \rho \). At this point we note that maximum offered load under PNG\(_A\) is 1 and the maximum under the under schemes is \( 1/R \). We see that the maximum offered load is dependent on the PNG in operation, we denote this quantity \( \rho_{\text{max}}(PNG) \). If the load offered to the system exceeds \( \rho_{\text{max}}(PNG) \) then the queue length will grow without bound. We can now fall back on the rich field of queue theory which has solved problems similar to the two outlined above.

### 3.6.1 Theoretical Evaluation of System Under PNG\(_A\)

When tasks are executed precisely they undergo two stages of service, where the time spent in each stage is exponentially distributed. The server model for PNG\(_A\) is shown in Figure 3.14

![Two Stage Server Model of System Under PNG\(_A\)](image)

Figure 3.14: Two Stage Server Model of System Under PNG\(_A\)

The symbols \( \mu_m \) and \( \mu_o \) are the service rates of the two servers. These two symbols are also the reciprocals of the means of the exponential distributions that describe the time spent in each server. Now we can use an extremely well known formula for the
average number of customers in an M/G/1 system. The M/G/1 system is defined to have one service facility with a Poisson arrival process and arbitrary service time distribution. This is precisely the situation that we find when using $PNG_A$.

We use the Pollaczek–Khinchin mean–value formula [40] that provides an expression for the average number of tasks in the system in terms of the first two moments of the service time distribution and the arrival rate. In Section 3.5.2 we defined the random variable $t_p$ to represent the precise computation time. Let the first and second moments of $t_p$ be $\bar{t}_p$ and $\sigma_{t_p}^2$ respectively. Also define

$$C_{t_p}^2 = \frac{\sigma_{t_p}^2}{(\bar{t}_p)^2}$$

(3.31)

Now the average number of tasks in the system is defined by the Pollaczek–Khinchin formula as

$$\bar{N} = \rho + \rho^2 \frac{1 + C_{t_p}^2}{2(1 - \rho)}$$

(3.32)

Let us now derive the two moments of $t_p$. The first moment is the mean of $t_p$, denoted $\bar{t}_p$, which can be calculated due to the fact that the mean of the sum of two independent random variables is the sum of the means of the two random variables. Now because $t_p = t_m + t_o$ (from Section 3.5.2) we have that

$$\bar{t}_p = \frac{1}{\mu_m} + \frac{1}{\mu_o}$$

(3.33)

The second moment of $t_p$ is the variance of $t_p$, denoted $\sigma_{t_p}^2$. We can calculate $\sigma_{t_p}^2$ from the fact that the variance of the sum of independent random variable is sum of the variances of the random variables. Therefore

$$\sigma_{t_p}^2 = \left(\frac{1}{\mu_m}\right)^2 + \left(\frac{1}{\mu_o}\right)^2$$

(3.34)

Calculating $C_{t_p}^2$ using Equations (3.31), (3.33) and (3.34) we get

$$C_{t_p}^2 = \frac{\left(\frac{1}{\mu_m}\right)^2 + \left(\frac{1}{\mu_o}\right)^2}{(\frac{1}{\mu_m} + \frac{1}{\mu_o})^2}$$

$$= \frac{\left(\frac{1}{\mu_m}\right)^2 + \left(\frac{1}{\mu_o}\right)^2}{\left(\frac{1}{\mu_m}\right)^2 + \frac{2}{\mu_m\mu_o} + \left(\frac{1}{\mu_o}\right)^2}$$


We can eliminate $\frac{1}{\mu_o}$ using Equation (3.2). Specifically, we substitute $\frac{1}{\mu_o} = \frac{1}{\mu_m} (\frac{1}{R} - 1)$.

$$C_{tp}^2 = \frac{\left(\frac{1}{\mu_m}\right)^2 + (\frac{1}{R} - 1)^2 \left(\frac{1}{\mu_m}\right)^2}{\left(\frac{1}{\mu_m}\right)^2 + 2(\frac{1}{R} - 1) \left(\frac{1}{\mu_m}\right)^2 + (\frac{1}{R} - 1)^2 \left(\frac{1}{\mu_m}\right)^2}$$

$$= \frac{\left(\frac{1}{R} - \frac{2}{R} + 2\right) \left(\frac{1}{\mu_m}\right)^2}{(R - 2) \left(\frac{1}{\mu_m}\right)^2 + \left(\frac{1}{R^2} - \frac{2}{R} + 1\right) \left(\frac{1}{\mu_m}\right)^2}$$

$$= 1 - 2R + 2R^2$$

Now we substitute $C_{tp}^2$ into Equation (3.32) to get an expression for the average number of tasks in the system.

$$N_s = \rho + \rho^2 \frac{2 - 2R + 2R^2}{2(1 - \rho)}$$

$$= \rho + \rho^2 \frac{1 - R + R^2}{1 - \rho}$$

(3.35)

It is easy to show that the average number of tasks in the queue can be expressed in terms of the average number of tasks in the system [40].

$$N_q = N_s - \rho$$

$$= \frac{\rho^2}{1 - \rho} \left(1 - R + R^2\right)$$

(3.36)

The normalised mean waiting time follows via Equation (3.7). With this theoretical basis we now study the theoretical limits of performance for $PNG_A$.

**Performance at Load Limits**

From Equations (3.7) and (3.36) we get

$$W_n = \frac{\rho}{1 - \rho} \left(1 - R + R^2\right)$$

(3.37)

We see that the denominator goes to infinity as $\rho \to 1$ Therefore the limit of $W_n$ as $\rho \to 1$ is $\infty$.

The other two metrics are trivial for $PNG_A$. Tasks are always computed imprecisely so the mean computation time is $1/\mu_m + 1/\mu_o$ which is normalised to be

$$Q_e = \frac{\frac{1}{\mu_m} + \frac{1}{\mu_o}}{\frac{1}{\mu_m} + \frac{1}{\mu_o}}$$

$$= 1$$

(3.38)
Also since tasks are always computed imprecisely the time until notice metric $T_u$ is zero for all $\rho$.

The system performance under $PNG_A$ at the load limits is shown in Table (3.5).

<table>
<thead>
<tr>
<th>Metric</th>
<th>any $\rho$</th>
<th>$\rho \to 0$</th>
<th>$\rho \to 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$\frac{\rho}{1 \cdot \rho} (1 - R + R^2)$</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$T_u$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.5: Performance at Limits of $\rho$ for System Under $PNG_A$

**Performance at $H$ Limits**

$PNG_A$ does not use $H$ to implement its control and therefore the performance of the system under $PNG_A$ will not depend on $H$.

### 3.6.2 Theoretical Evaluation of System Under $PNG_N$

Under this PNG tasks are always executed imprecisely. Therefore each task undergoes one stage of execution, unlike the system under $PNG_A$ where each tasks passes through two computational stages. The load experienced by the processor when $PNG_N$ is operating is $R$ times the load presented to the system. The random variable representing the service time of each tasks’ precise component is $t_p$ as defined in Section 3.5.2. Now because $t_p$ is exponentially distributed we can apply a very simple result relating to M/M/1 systems. M/M/1 systems are defined by a Poisson arrival process and one service facility with service time exponentially distributed. This is nothing more than our system under $PNG_N$. 
The fundamental result for the M/M/1 system is that the average number of tasks in the system is

$$N_s = \frac{\rho_s}{1 - \rho_s}$$

where $\rho_s$ is the load experienced by the server (the same as the load entering the queue for non-imprecise system). When the system is under $PNG_N$ the load experienced by the processor is $\rho_s = R\rho$. So we have

$$N_s = \frac{R\rho}{1 - R\rho}$$

Using the result that the average number of tasks in the queue is $\bar{N}_s$ minus the utilisation of the processor [40] we obtain

$$N_s = \frac{R\rho}{1 - R\rho} - R\rho = \frac{R\rho - R\rho(1 - R\rho)}{1 - R\rho} = \frac{R^2\rho^2}{1 - R\rho}$$

(3.39)

The normalised mean waiting time follows via Equation (3.7). With this theoretical basis we now study the theoretical limits of performance for $PNG_N$.

Performance at Load Limits

From Equations (3.7) and (3.39) we get

$$W_n = \frac{R^2\rho}{1 - R\rho}$$

(3.40)

We see that the denominator goes to infinity as $\rho \to 1/R$. Therefore the limit of $W_n$ as $\rho \to 1/R$ is $\infty$. Obviously as $\rho$ tends to zero so does $W_n$.

The other two metrics are trivial for $PNG_N$. Tasks are always computed precisely so the mean computation time is $1/\mu_m$ which is normalised to be

$$Q_c = \frac{1}{\frac{1}{\mu_m} + \frac{1}{\mu_o}}$$

$$= R$$

(3.41)
Also since tasks are always computed imprecisely the time until notice metric $T_u$ is zero for all $\rho$.

The system performance under $PNG_N$ at the load limits is shown in Table (3.6)

<table>
<thead>
<tr>
<th>Metric</th>
<th>any $\rho$</th>
<th>$\rho \to 0$</th>
<th>$\rho \to \frac{1}{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$\frac{R^2 \rho}{1 - R \rho}$</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$R$</td>
<td>$R$</td>
<td>$R$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.6: Performance at Limits of $\rho$ for System Under $PNG_N$

**Performance at $H$ Limits**

$PNG_N$ does not use $H$ to implement its control and therefore the performance of the system under $PNG_N$ will not depend on $H$.

### 3.6.3 Theoretical Evaluation of System Under $PNG_1$

We cannot solve for $W_n$ explicitly because the system is not homogeneous. We therefore resort to theoretical limiting values.

**Performance at Load Limits**

As the load tends to its maximum of $1/R$ most tasks will be executed imprecisely, therefore the average computation quality will tend to $R$, also $W_n$ will tend to infinity. As we described in Section (3.3) imprecise notices can be issued to a given task when there are at least $H$ tasks queued behind the task. When most tasks are being executed
imprecisely because the load is near maximum the system will be in state $g = (H, |G| - H, 0)$ as shown in Figure 3.7. Therefore, generally speaking, a task will arrive, wait for $H$ arrivals, then receive an imprecise notice. The average time taken for $H$ tasks to arrive is simply $H\lambda$. Therefore the normalised time until notice generation when the load is high is

$$\lim_{\rho \rightarrow \frac{1}{R}} T_u = H\rho$$  \hspace{1cm} (3.42)

The system performance under PNG$_1$ at the load limits is shown in Table (3.7)

<table>
<thead>
<tr>
<th>Metric</th>
<th>any $\rho$</th>
<th>$\rho \rightarrow 0$</th>
<th>$\rho \rightarrow \frac{1}{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>?</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>?</td>
<td>1</td>
<td>$R$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>?</td>
<td>0</td>
<td>$H\rho$</td>
</tr>
</tbody>
</table>

Table 3.7: Performance at Limits of $\rho$ for System Under PNG$_1$

**Performance at $H$ Limits**

We begin by considering the limiting value of $W_n$ as $H \rightarrow \infty$. If the load on the system is less than one ($\rho < 1$), the random variable that describes the queue length has some finite mean and some finite variance. We therefore conclude that $N_s$ will never exceed $H$ as $H \rightarrow \infty$ given that the load is less than one. Therefore PNG$_1$ will always issue precise notices and all tasks will be executed precisely when $H$ tends to $\infty$. The system will therefore behave in the same way as PNG$_A$. We now have the following limit from Equation 3.37

$$\lim_{H \rightarrow \infty} W_n \bigg|_{\rho < 1} = \frac{\rho}{1 - \rho} \left(1 - R + R^2\right)$$  \hspace{1cm} (3.43)

Also, because $N_s$ never exceeds $H$ as $H \rightarrow \infty$, every new arrival be given a precise notice as it enters the processor. The average time from arrival to entering the processor
is $N_q$. Therefore $T_u$ has the same limit as $W_n$.

$$\lim_{H \to \infty} T_u \bigg|_{\rho < 1} = \frac{\rho}{1 - \rho} \left(1 - R + R^2\right)$$  \hspace{1cm} (3.44)

Of course when all tasks are being executed precisely the average normalised computation time is 1.

$$\lim_{H \to \infty} Q_c \bigg|_{\rho < 1} = 1$$  \hspace{1cm} (3.45)

The limit as $H \to 0$ implies that the $N_s > H$ condition is always true and therefore tasks will always be executed imprecisely under $PNG_1$. All the limits that applied to $PNG_N$ now apply to $PNG_1$ when $H$ is tending to zero.

$$\lim_{H \to 0} W_n = \frac{R^2 \rho}{1 - R\rho}$$  \hspace{1cm} (3.46)

$$\lim_{H \to 0} Q_c = R$$  \hspace{1cm} (3.47)

Since all tasks are being executed imprecisely the system must rely on $ICP_1$ to generate the notices for tasks. However, $ICP_1$ will only generate imprecise notices when there are more than $H$ tasks in the queue. When $H$ is zero this means that a task can be given an imprecise notice as soon as it arrives. Therefore

$$\lim_{H \to 0} T_c = 0$$  \hspace{1cm} (3.48)

The system performance under $PNG_1$ at the limits of $H$ is shown in Table (3.8)

<table>
<thead>
<tr>
<th>Metric</th>
<th>$H \to 0$</th>
<th>$H \to \infty$ and $\rho &lt; 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$R^2 \rho$ (\div) $1 - R\rho$</td>
<td>$\frac{\rho}{1 - \rho} \left(1 - R + R^2\right)$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$R$</td>
<td>$1$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>$0$</td>
<td>$\frac{\rho}{1 - \rho} \left(1 - R + R^2\right)$</td>
</tr>
</tbody>
</table>

Table 3.8: Performance at Limits of $H$ for System Under $PNG_1$
3.6.4 Theoretical Evaluation of System Under $PNG_H$

Again we cannot solve for $W_n$ explicitly because the system is not homogeneous. We therefore resort to limiting values.

Performance at Load Limits

As the load tends to its maximum of $1/R$ most tasks will be executed imprecisely, therefore the average computation quality will tend to $R$, also $W_n$ will tend to infinity. As we described in Section (3.3) imprecise notices can be issued to a given task when there are at least $H$ tasks queued behind the task. When most tasks are being executed imprecisely because the load is near maximum the system will be in state $g = (H, |G| - H, 0)$ (see Figure 3.8). In this state there are $H$ tasks with no notice at the end of the queue. Therefore a task will arrive, wait for $H$ arrivals, then receive an imprecise notice. The average time taken for $H$ tasks to arrive is simply $H\lambda$. Therefore the normalised time with notice at high load is

$$\lim_{\rho \to \frac{1}{R}} T_u = H\rho$$

The system performance under $PNG_H$ at the load limits is shown in Table (3.9)

<table>
<thead>
<tr>
<th>Metric</th>
<th>any $\rho$</th>
<th>$\rho \to 0$</th>
<th>$\rho \to \frac{1}{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>?</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>?</td>
<td>1</td>
<td>$R$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>?</td>
<td>0</td>
<td>$H\rho$</td>
</tr>
</tbody>
</table>

Table 3.9: Performance at Limits of $\rho$ for System Under $PNG_H$
Performance at $H$ Limits

We begin by considering the limiting value of $W_n$ as $H \to \infty$. If the load on the system is less than one ($\rho < 1$), the random variable that describes the queue length has some finite mean and some finite variance. We therefore conclude that $N_s$ will never exceed $H$ as $H \to \infty$ given that the load is less than one. Therefore $PNG_H$ will always issue precise notices and all tasks will be executed precisely when $H$ tends to $\infty$. The system will therefore behave in the same way as $PNG_A$. We now have the following limit from Equation (3.37)

$$
\lim_{H \to \infty} W_n \bigg|_{\rho < 1} = \frac{\rho}{1-\rho} \left( 1 - R + R^2 \right) 
$$

(3.50)

Also, because $N_s$ never exceeds $H$ as $H \to \infty$, every new arrival be given a precise notice as it arrives. Therefore

$$
\lim_{H \to \infty} T_u = 0
$$

(3.51)

Of course when all tasks are being executed precisely the average normalised computation time is 1.

$$
\lim_{H \to \infty} Q_c \bigg|_{\rho < 1} = 1
$$

(3.52)

The limit as $H \to 0$ implies that the $N_s > H$ condition is always true and therefore tasks will always be executed imprecisely under $PNG_H$. All the limits that applied to $PNG_N$ now apply to $PNG_H$ when $H$ is tending to zero.

$$
\lim_{H \to 0} W_n = \frac{R^2 \rho}{1 - R \rho}
$$

(3.53)

$$
\lim_{H \to 0} Q_c = R
$$

(3.54)

Since all tasks are being executed imprecisely the system must rely on $ICP_H$ to generate the notices for tasks. However, $ICP_H$ will only generate imprecise notices when there are more than $H$ tasks in the queue. When $H$ is zero this means that a task can be given an imprecise notice as soon as it arrives. Therefore

$$
\lim_{H \to 0} T_u = 0
$$

(3.55)

The system performance under $PNG_H$ at the limits of $H$ is shown in Table (3.10)
3.7. Performance Evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>$H \to 0$</th>
<th>$H \to \infty$ and $\rho &lt; 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$\frac{R \rho}{1 - R \rho}$</td>
<td>$\frac{\rho}{1 - \rho} (1 - R + R^2)$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$R$</td>
<td>$1$</td>
</tr>
<tr>
<td>$T_u$</td>
<td>$0$</td>
<td>$0$</td>
</tr>
</tbody>
</table>

Table 3.10: Performance at Limits of $H$ for System Under $PNG_H$

3.7 Performance Evaluation

The behaviour of the performance metrics $W_n$, $Q_c$, and $T_u$ as functions of the system load parameters $\rho$, $R$ and $H$ are now presented. Note that $\rho$ and $R$ are characteristics of the load, and $H$ is a precise notice generator configuration parameter.

Before evaluating the system performance using the state-transition-rate diagram (STRD) approach, some dynamic system simulations are conducted to verify the validity of the assumptions made about steady state conditions. The SIMSCRIPT II [47, 48] simulation language is used to model the system, generate the task arrivals and calculate performance metrics. The metrics derived by simulation are plotted using points on all subsequent Figures. As can be seen from Figures 3.15-3.23, the simulation results agree with the numerical solutions obtained from the STRD solution. The assumption that the system reaches steady state and is therefore representable by a STRD is confirmed.

To evaluate system performance, the offered load is varied between lightly loaded ($\rho < 1$), fully loaded ($\rho = 1$) and overloaded ($\rho > 1$). In addition, the performance of the system with $H$ as the variable feature is studied. These two tests would allow a system administrator to best choose the type of PNG and value of $H$ for a particular environment.
3.7.1 Performance Under Variable Load

Figures 3.15—3.20 show the performance metrics with the offered load $\rho$ as a variable. There are two sets of graphs for each metric as $R$ is made a parameter ($R = 0.4$ and $0.8$). $R = 0.4$ implies that the mandatory load is 40% of the total offered load (and the other 60% constituting the optional offer load), while $R = 0.8$ means that 80% of each task is mandatory. Each graph contains the system performance under each PNG for two values of $H$ ($H = 3$ and 10). This shows the effects of a different control threshold on the system under variable load.

The general performance limits derived in Section (3.6) are evaluated for the particular load conditions tested here. This information is shown in Table 3.11. Comparison between these limits and the results obtained in Figures 3.15–3.20 shows good correspondence.

<table>
<thead>
<tr>
<th>Metric</th>
<th>PNG</th>
<th>$R$</th>
<th>$H$</th>
<th>$\rho \rightarrow 0$</th>
<th>$\rho \rightarrow \rho_{\text{max}}(\text{PNG})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>any</td>
<td>any</td>
<td>any</td>
<td>0</td>
<td>$\infty$</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$PNG_A$</td>
<td>any</td>
<td>any</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$PNG_N$</td>
<td>0.4</td>
<td>any</td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>any</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>$PNG_1, PNG_H$</td>
<td>0.4</td>
<td>any</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>any</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>$T_u$</td>
<td>$PNG_A, PNG_N$</td>
<td>any</td>
<td>any</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>$PNG_1, PNG_H$</td>
<td>0.4</td>
<td>3</td>
<td>0</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>3</td>
<td>0</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.11: Performance at Limits of $\rho$ for Simulation Environment

We now discuss the effects of $H$, $R$, and varying $\rho$ on the performance of the system. At the end of each section the relative performance of the PNGs is summarised.
Response time performance: $W_n$ vs $\rho$

Figures 3.15 and 3.16 show the response time $W_n$ as the offered load ($\rho$) increases. Note that the theoretical upper limit of the load is $1/R$ for the PNGs that utilise imprecise computations only.

When the load on these systems is very low a new arrival to the system will not have to be queued. This explains why $W_n$ tends to 0 as the load tends to zero. $W_n$ increases as the load increases. The curves in Figure 3.16 follow an exponential rise with increasing load, but the curves in Figure 3.15 show some unusual behaviour. Specifically, with $H = 10$, $W_n$ can be seen to decrease in value as the load increases. This behaviour can be explained by considering performance asymptotes for precise and imprecise computations. These asymptotes are the performance given by $PNG_A$ and $PNG_N$ respectively. For $\rho \ll 1$, $PNG_H$ and $PNG_1$ will decide to issue precise notices, because the number of tasks in the system ($N_s$) will not exceed $H$. The waiting time curve will therefore follow the $PNG_A$ asymptote. As the load increases above 1, $N_s$ will exceed $H$ and some tasks will be imprecisely computed. Consequently the curve will start to follow the $PNG_N$ asymptote. As the load increases still further all tasks will be imprecisely computed and the curve will tend to the $PNG_N$ asymptote. The value of $H$ determines at what load the curve will start to break away from the precise asymptote. This phenomenon is not as prominent in the $R = 0.8$ curves because the $PNG_A$ and $PNG_N$ asymptotes are close together. Observe that $PNG_A$ and $PNG_N$ form the boundaries of performance for $PNG_1$ and $PNG_H$.

The control threshold $H$ can affect how the systems react to changes in load. If $H$ is large (e.g., $H = 10$) then the mean waiting times are elevated compared with the same PNG with $H = 3$. This is because a large value of $H$ means that the system will wait until the queue grows large before initiating any imprecise computations. It is evident that the value of $H$ reflects the responsiveness of the system to transient increases in load. With $H = 3$ the PNG will take action against rising queue length earlier than with $H = 10$. 
The relative performance of the four PNGs is always in the same order for a given load. The feature of the PNGs that causes this ordering is linked to the maximum number of precise notices that can exist in the system at any given time instant. We denote this quantity as $V$. $PNG_H$ has the highest $V$ ($= H$), and $PNG_1$ has $V = 1$. One can think of $PNG_N$ having $V = 0$, (the lowest possible), and $PNG_A$ having $V = N$, (the highest possible). We can see that $PNG_A$ will issue the most precise notices, followed by $PNG_H$. $PNG_1$ will issue at most one precise notice and $PNG_N$ will never issue a precise notice. $PNG_A$ will give the worst response time because it has the largest potential precise computation commitment. Similarly, $PNG_N$ gives the most responsive performance because it has the smallest potential precise computation commitment.

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNG_N</td>
<td>Best</td>
</tr>
<tr>
<td>PNG_1</td>
<td></td>
</tr>
<tr>
<td>PNG_H</td>
<td></td>
</tr>
<tr>
<td>PNG_A</td>
<td>Worst</td>
</tr>
</tbody>
</table>

Table 3.12: $W_n$ versus $\rho$ Performance Comparison

Computation quality performance: $Q_c$ vs $\rho$

The results are shown in Figures 3.17 and 3.18. The static group yields trivial results in this metric. Simply precise computations due to $PNG_A$ yield task result quality $Q_c = 1$ and imprecise computations due to $PNG_N$ yield task result quality $Q_c = R$.

Consider the dynamic group. When the load is very light all tasks are computed precisely and therefore $Q_c = 1$. While the load is low ($\rho < 1$) most tasks are precisely computed because $N_s$ rarely exceeds $H$. As the load on the system increases above one $N_s$ will start to increase and exceed $H$ more often. This increase will mean that
more tasks will be executed imprecisely in an attempt to keep $N_n$ below $H$. As a result the average precision of task computation will start to fall. As the load increases still further most tasks will be imprecisely computed. Under very heavy load all tasks will be imprecisely computed and therefore $Q_c$ will equal to $R$.

$H$ has some effect on system performance. $H = 10$ provides better computation quality than $H = 3$; $H = 10$ means that the NG will wait longer before starting to issue imprecise notices and thus reducing the average computation quality.

$PNG_A$ provides the best computation quality since it has the highest $V$, and similarly $PNG_N$ has the lowest computation quality because of its low $V$. Note that this order is the exact opposite of the $W_n$ performance order. Already we see the dilemma in deciding which is the “best” PNG.

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNG_A$</td>
<td>Best</td>
</tr>
<tr>
<td>$PNG_H$</td>
<td></td>
</tr>
<tr>
<td>$PNG_1$</td>
<td></td>
</tr>
<tr>
<td>$PNG_N$</td>
<td>Worst</td>
</tr>
</tbody>
</table>

Table 3.13: $Q_c$ versus $\rho$ Performance Comparison

**Time until notice generation performance: $T_u$ vs $\rho$**

The graphs in Figures 3.19 and 3.20 show the notice generation performance of these PNGs. At this point we see that the static notice generators ($PNG_A$ and $PNG_N$) have the ideal value of $T_u$ and therefore the following discussion relates to the dynamic group ($PNG_H$ and $PNG_1$) only.

For $PNG_H$ and $PNG_1$, $T_u$ increases in an exponential manner with load until the load exceeds 1, when $T_u$ starts to decrease with increasing load.
The exponential rise is caused by $N_s$ exceeding $H$ more and more often as the load tends toward one. Once $N_s$ exceeds $H$, new arrivals will have notice generated for them under two scenarios.

1. $N_s$ falls back below $H$ and all tasks have precise notices generated for them, or

2. $N_s$ stays above $H$ for such a time as to allow tasks to be given imprecise notices when $H$ tasks are waiting behind them, and they themselves have not had a notice generated for them.

As the load keeps increasing $N_s$ will increase, and more and more imprecise notices will be issued. This is why $T_u$ starts to decrease as the load gets very heavy. Note that $T_u$ is the mean time until any type of notice (not just precise notices) is generated for a task.

$PNG_H$ performs better that $PNG_1$ in the notice generation area because it can give precise notices anywhere in $(T_1, T_3)$, but $PNG_1$ can only issue precise notices at $T_3$ (see Figure 3.3).

$PNG_H$ demonstrates some special behaviour. When the load is low $H = 10$ performs better that $H = 3$, but when the load is high $H = 3$ performs better. This is because $H = 10$ will perform better at precise notice generation, and $H = 3$ will perform better at imprecise notice generation. In fact, a “fast” precise notice generation would require $H = \infty$, and a “fast” imprecise notice generation would require $H = 0$.

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNG_A$ and $PNG_N$</td>
<td>Best</td>
</tr>
<tr>
<td>$PNG_H$</td>
<td></td>
</tr>
<tr>
<td>$PNG_1$</td>
<td>Worst</td>
</tr>
</tbody>
</table>

Table 3.14: $T_u$ versus $\rho$ Performance Comparison
3.7.2 Performance Under Variable $H$

We now study the performance of the notice generators with $H$ as a variable. By comparing $N_s$ to $H$, the dynamic PNGs can make their decisions. $PNG_A$ and $PNG_N$ do not use $H$ as a control variable, and therefore their performance is invariant with $H$.

The general performance limits derived in Section 3.6 are evaluated for the particular load conditions present here. This information is shown in Tables 3.15 and 3.16. In Table 3.16 “n.a.” implies this bound is not available. Comparison between these limits and the results obtained in Figures 3.21—3.23 shows good correspondence.

<table>
<thead>
<tr>
<th>Metric</th>
<th>PNG</th>
<th>$R$</th>
<th>$\rho$</th>
<th>any $H$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_n$</td>
<td>$PNG_A$</td>
<td>0.4</td>
<td>0.75</td>
<td>2.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>0.75</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>$PNG_N$</td>
<td>0.4</td>
<td>0.75</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0.27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.5</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>0.75</td>
<td>1.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>3.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.125</td>
<td>7.20</td>
</tr>
<tr>
<td>$Q_c$</td>
<td>$PNG_A$</td>
<td>any</td>
<td>any</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$PNG_N$</td>
<td>0.4</td>
<td>any</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8</td>
<td>any</td>
<td>0.8</td>
</tr>
<tr>
<td>$T_u$</td>
<td>$PNG_A, PNG_N$</td>
<td>any</td>
<td>any</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.15: Performance of $PNG_A$ and $PNG_N$ at Limits of $H$ for Simulation Environment

Response time performance: $W_n$ vs $H$

Figure 3.21 show how the PNGs perform with varying $H$. There are three groups of lines corresponding to three different load levels. It is apparent from the figures that the PNGs provide very similar tends in waiting time performance as a function of $H$. 
## Table 3.16: Performance of PNG\(_1\) and PNG\(_H\) at Limits of \(H\) for Simulation Environment

The two groups with \(\rho \geq 1\) show that \(W_n\) increases in proportion to \(H\). This is because when the load is greater than one the queue length will increase without bound until imprecise computations are used to limit \(N_s\). When the load is less than one the system can compute every task precisely and still maintain bounded queue length. Note again that the static PNGs are the bounding performance for the dynamic PNGs.
Table 3.17: $W_n$ versus $H$ Performance Comparison

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNG_N$</td>
<td>Best</td>
</tr>
<tr>
<td>$PNG_1$</td>
<td></td>
</tr>
<tr>
<td>$PNG_H$</td>
<td></td>
</tr>
<tr>
<td>$PNG_A$</td>
<td>Worst</td>
</tr>
</tbody>
</table>

**Computation quality performance: $Q_c$ vs $H$**

The computation quality graphs shown in Figure 3.22 demonstrate that $Q_c$ is independent of $H$ for high $H$. $H$ only has an effect on the computation quality when $H < \approx 15$. The two groups that have $\rho \leq 1$ can compute all tasks precisely ($Q_c = 1$), but when the load is greater than one some tasks must be executed imprecisely to maintain bounded response time. These imprecise computations degrade $Q_c$.

Table 3.18: $Q_c$ versus $H$ Performance Comparison

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNG_A$</td>
<td>Best</td>
</tr>
<tr>
<td>$PNG_H$</td>
<td></td>
</tr>
<tr>
<td>$PNG_1$</td>
<td></td>
</tr>
<tr>
<td>$PNG_N$</td>
<td>Worst</td>
</tr>
</tbody>
</table>

**Time until notice generation performance: $T_u$ vs $H$**

The time until notice generation performance is shown in Figures 3.23. It can be seen that $H$ has a strong influence on $T_u$ when the load is high. This is because when the load is high imprecise notices are generated for tasks that have $H$ tasks arrive after them. The larger the $H$ the longer the time that the tasks will have to wait. When the
load is low, $N_s$ may rarely exceed $H$ if $H$ is large enough. So if $N_s$ is always less than $H$, then tasks arriving to a $PNG_H$ will have notices generated for them immediately, i.e., $T_u = 0$. In $PNG_1$ precise notices are generated when tasks enter the processor (if $N_s \leq H$), i.e., $T_u = W_n$.

<table>
<thead>
<tr>
<th>PNG</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PNG_A$ and $PNG_N$</td>
<td>Best</td>
</tr>
<tr>
<td>$PNG_H$</td>
<td></td>
</tr>
<tr>
<td>$PNG_1$</td>
<td>Worst</td>
</tr>
</tbody>
</table>

Table 3.19: $T_u$ versus $H$ Performance Comparison

### 3.8 Conclusions

We have proposed a notice generation approach to decide on task execution precision in dynamic imprecise soft real-time systems. Four notice generators have been designed, analysed and evaluated. We have categorised these four into static and dynamic notice generators, based on whether they use system state variables, $N_s$ (the number of tasks in the system) and $N_p$ (the number of tasks that have precise notices) in their notice generation process. $PNG_A$ and $PNG_N$ are static while $PNG_1$ and $PNG_H$ are dynamic.

We proposed three performance metrics to reflect the requirements of the system in task waiting time and computation quality, in order to evaluate the performance of the system controlled by these four notice generations. The metrics were mean waiting time, mean computation time and mean time until notice generation, all are normalised with respect to the mean precise computation time. To analyse the systems, state-transition-rate theory is used to construct state-transition-rate models, and the performance metrics are calculated explicitly from the state probabilities. Performance
limits were also derived. Both limits and numerical results agreed with simulation by
the SIMSCRIPT II event driven simulation language.

The performance evaluation showed that $PNG_N$, a static notice generator, offered
the most responsive service; its waiting time is the lower bound of the imprecise system.
The other static notice generator $PNG_A$ provided the highest computation quality.
The dynamic notice generators $PNG_H$ and $PNG_1$ offered performance in between
the two static PNGs. The most important feature of the dynamic group performance
was that the mean task waiting time was kept low when the system was overloaded,
and simultaneously reasonable computation quality was maintained. This is a direct
consequence of using imprecise computations. The external system was provided with
advanced information with regard to task result accuracy in all four schemes. The
static group of PNGs executed tasks deterministically and therefore provided the best
time until notice performance. This deterministic behaviour was obtainable at a cost
to control flexibility.

The four proposed notice generators operating in an imprecise computation en-
vironment can be applied to a number of traditionally more difficult applications. One
example is in managing transient increase in computational load occurring in computer
controlled industrial process during a system initialisation or an operation emergency
[36]. Others include time/accuracy tradeoffs in intelligent real-time control in which
some artificial intelligence or computational intensive Kalman filter-type algorithms
may be used in order to meet difficult mission requirements. The schemes presented
here can be used as elements of a larger more flexible control scheme.
Figure 3.15: $W_n$ Under Variable Load with $R = 0.4$
Figure 3.16: $W_n$ Under Variable Load with $R = 0.8$
Figure 3.17: \( Q_c \) Under Variable Load with \( R = 0.4 \)
Figure 3.18: $Q_c$ Under Variable Load with $R = 0.8$
Figure 3.19: $T_u$ Under Variable Load with $R = 0.4$
3.8. Conclusions

Figure 3.20: $T_n$ Under Variable Load with $R = 0.8$
Figure 3.21: $W_n$ Under Variable $H$ with $R = 0.4$
Figure 3.22: $Q_c$ Under Variable $H$ with $R = 0.4$
Figure 3.23: $T_u$ Under Variable $H$ with $R = 0.4$
Chapter 4

Controlling the System During Load Transitions

4.1 Introduction

There may be engineering considerations that recommend a change of precise notice generation (PNG) during run-time. This type of operation may be required when a large transient overload occurs. Swapping between schemes during run-time is difficult because the system state must be preserved while the swap is implemented. A simple and inefficient method to swap schemes would be to buffer tasks until the queue was empty then to fill the queue by emptying the buffer, but with the system being controlled by the new control scheme. This method has one obvious inefficiency in that the task notification is unnecessarily suspended. In this chapter we study methods to swap schemes during run-time while keeping notice generation active.

As we have seen in Section 3 the system state under a particular PNG is well defined. Moreover, the system state under a given PNG may not be an element of the set of all possible system states under another PNG. This fact means that we cannot simply switch the PNG algorithm. To do so may violate system state continuity. To overcome this problem we propose to use a transitional PNG to convert the system
state into something that is compatible with the future PNG.

### 4.2 Problem Formalisation

We denote the PNG in operation before the PNG swap is requested as $PNG_{bs}$. The value of $PNG_{bs}$ is taken from the set $\{PNG_A, PNG_N, PNG_1, PNG_H\}$. Similarly we denote the PNG that we wish to swap to as $PNG_{as}$ which is also a member of the set $\{PNG_A, PNG_N, PNG_1, PNG_H\}$. In Chapter 3 we defined the generalised system state ($\mathcal{G}$) under each of the four NGs. The system state was represented by $\mathcal{G} = \{\mathcal{W}, \mathcal{I}, \mathcal{P}\}$, where the subsets correspond to the tasks without notice, with imprecise notices and with precise notices respectively. The Figures depicting the generalised system state are repeated here for reference.

Note that $|\mathcal{P}| \in (0, 1)$ under $PNG_1$, and $|\mathcal{P}| \in (0, H)$ under $PNG_H$. This is why $\mathcal{G}_1$ is not compatible with $\mathcal{G}_H$ in general. It is convenient to refer to the size of each of the sets in $\mathcal{G}$ so we define the vector $\mathbf{g}$ as follows

$$\mathbf{g} \triangleq (|\mathcal{W}|, |\mathcal{I}|, |\mathcal{P}|)$$

The difficulty in swapping from $PNG_{bs}$ to $PNG_{as}$ is that $\mathcal{G}_{bs}$ may not be valid under $PNG_{as}$. For example, consider $PNG_{bs} = PNG_N$ and $\mathbf{g}_{bs} = (0, 1, 0)$, now say we wanted to swap to $PNG_1$. This is not possible immediately because as shown in Figure 4.1, $\mathbf{g} = (0, 1, 0)$ is not valid under $PNG_1$. The answer to this problem is to utilise Transitional Notice Generators that act as an interface between $PNG_{bs}$ and $PNG_{as}$. We have shown that that there are four generalised system states, one four each PNG. The problem of swapping PNGs is now a problem of manipulating generalised system states.

The general algorithm used in converting from $PNG_{bs}$ to $PNG_{as}$ is
4.2. Problem Formalisation

(a) $G_A$, The System Under $PNG_A$

(b) $G_N$, The System Under $PNG_N$

(c) $G_1$, The System Under $PNG_1$

(d) $G_H$, The System Under $PNG_H$

Figure 4.1: Generalised System States

**Algorithm 4.2.1 (TPNG_A)** The mechanism for converting between PNGs is

1. **Stop** $PNG_{bs}$
2. **Execute** appropriate $TNG$
3. **Start** $PNG_{as}$
The $TNG$ will terminate due to its own internal control. Keep in mind that while the $TNG$ is operating two random processes are occurring. Specifically, the arrival and computation process are randomly effecting the system state. Any $TNG$ must be able to guarantee $G$ transformation under these random conditions.

### 4.3 Transitional Precise Notice Generator Derivation

The objective of Transitional Notice Generators (TNGs) is to start with any valid state under $PNG_{ba}$ and convert that to a state that is valid under $PNG_{as}$. In general, the state of the system while $TNG$ is operating will not be valid under $PNG_{ba}$ or $PNG_{as}$. We define a parameter $\tau$ as a measure of the time required to implement a PNG swap. Let us now consider $TNG$ for each $PNG_{as}$.

#### 4.3.1 Swapping to $PNG_A$ via $TNG_A$

In this case we must convert from any $G$ to a system with all tasks having precise notices. We need to convert from $(W_{ba}, T_{ba}, P_{ba})$ to $(\emptyset, \emptyset, P)$, where $\emptyset$ is the empty set. Now because we can't give precise notices to tasks that have imprecise notices we must wait until all of the tasks with imprecise notices have been computed. We can however issue precise notices to tasks that haven't already been notified. The rule for converting to $PNG_A$ is therefore

ALGORITHM 4.3.1 ($TNG_A$) *The transitional NG for converting from arbitrary $PNG_{ba}$ to $PNG_A$ is*

```plaintext
   issue precise notices to tasks in $W$
   WHILE $T \neq \emptyset$ DO
      IF $E_1$ issue precise notice to arriving task
```
The condition for termination of $TNG_A$ implies that $I_{bs} = \emptyset$ and $P_{bs} = \emptyset$ because $P$ is always followed by $I$ in the queue. After $I_{bs}$ has been emptied the only remaining original tasks are from $W_{bs}$, but these have been given precise notices and therefore become members of $P_{as}$. Now because new arrivals are given precise notices all tasks will have precise notices at termination.

The time taken for the algorithm to change the system state depends on whether or not there exist any tasks with imprecise notices. If $|I_{bs}|$ is non-empty then the tasks in $P_{bs}$ must first be computed. Given some $|I_{bs}|$, the average normalised time until $PNG_A$ can be started is

$$\tau_A = \begin{cases} |I_{bs}|R + |P_{bs}| & \text{if } |I_{bs}| \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

4.3.2 Swapping to $PNG_N$ via $TNG_N$

In this case we must convert from any $G$ to an system with all tasks having imprecise notices. We need to convert from $(W_{bs}, I_{bs}, P_{bs})$ to $(\emptyset, I, \emptyset)$. Again we cannot change the type of notice that a task already has so we must wait until all of the tasks with precise notices have been computed. We can however issue imprecise notices to tasks that haven’t already been notified. The rule for converting to $PNG_N$ is therefore

**ALGORITHM 4.3.2 ($TNG_N$) The transitional NG for converting from arbitrary $PNG_{bs}$ to $PNG_N$ is**

1. **issue imprecise notices to tasks in $W$**
2. **WHILE $P \neq \emptyset$ DO**
   - **IF $E_1$ issue imprecise notice to arriving task**

The condition for termination of $TNG_N$ is that $P = \emptyset$. After $P_{bs}$ has been emptied the only remaining original tasks are in $W_{bs}$ and $I_{bs}$. The tasks in $W_{bs}$ have been given imprecise notices and therefore become members of $I_{as}$. Now because new arrivals are given imprecise notices all tasks will have imprecise notices at termination.
The average time taken for this algorithm to change the system state is the average time required to compute \(|\mathcal{P}_{bs}|\) tasks precisely is
\[
\tau_N = |\mathcal{P}_{bs}|.
\]

### 4.3.3 Swapping to PNG\(_1\) via TNG\(_1\)

Due to the more complex nature of \(\mathcal{G}_1\) compared with \(\mathcal{G}_A\) and \(\mathcal{G}_N\) we consider each possible \(PNG_{bs}\) separately.

**Swapping from PNG\(_A\) to PNG\(_1\)**

The maximum size of \(\mathcal{P}\) under PNG\(_1\) is 1. TNG\(_A,1\) must therefore reduce the size of \(\mathcal{P}_{bs}\). All new arrivals will be members of \(\{W_{aa}, I_{aa}\}\) because no new precise notices will be given. The distribution of new arrivals into \(\{W_{aa}, I_{aa}\}\) must be made in a way that will result in \(\{W_{as}, I_{as}\}\) compatible with the imprecise computation predictor for PNG\(_1\) (ICP\(_1\)). ICP\(_1\) works by moving the oldest task in \(W\) into \(I\) when \(|W|\) becomes greater that \(H\). The rule for converting from PNG\(_A\) to PNG\(_1\) is therefore

**Algorithm 4.3.3 (TNG\(_A,1\))** The transitional NG for converting from PNG\(_A\) to PNG\(_1\) is

\[
\text{WHILE } |\mathcal{P}| > 1 \text{ DO}
\]

\[
\text{EXECUTE ICP}_H
\]

The terminating condition will result in the appropriate reduction in \(\mathcal{P}\). The imprecise notice generation is performed using the ICP defined for the system under PNG\(_H\). This is because ICP\(_H\) works for an arbitrary sized \(\mathcal{P}\) whereas ICP\(_1\) assumes that \(N_p \in (0, 1)\).

The average time taken for this algorithm to change the system state is the average time required to compute \([|\mathcal{P}_{bs}| - 1]\) tasks precisely. We need to know the
distribution of the queue length to be able to calculate the average value of $|P_{bs}| - 1$. Queue length distributions are not studied here. Therefore a result for $\tau_{A,1}$ is not possible.

**Swapping from $PNG_N$ to $PNG_1$ via $TNG_{N,1}$**

Under $PNG_N$ the generalised system state is $g = (0, |I_{bs}|, 0)$. In this case $P_{bs} = \emptyset$ which will stay empty at least until $I_{bs}$ is emptied. This is because we never have the case where tasks with imprecise notices are followed by tasks with precise notices in the queue. The only way that $|P|$ can be zero and $|I|$ non-zero under $PNG_1$ is if $|W| = H$. This means that we can swap after $H$ arrivals if $|I|$ is still non-zero. What happens if $I$ is emptied before $H$ arrivals occur? Well, in this case we can swap as soon as $I$ is emptied because $|G|$ will be less that $H$ and no tasks in the system will have notices. The rule for converting from $PNG_N$ to $PNG_1$ is therefore

**ALGORITHM 4.3.4 ($TNG_{N,1}$)** The transitional NG for converting from $PNG_N$ to $PNG_1$ is

\[
\text{WHILE } |W| < H \text{ AND } I \neq \emptyset \text{ DO}
\]

\[
do \text{ nothing}
\]

\[
\text{IF } I = \emptyset
\]

\[
\text{issue precise notice to task in } QP_1
\]

The terminating condition is reliant on two expressions. One of which is guaranteed to occur. The two processes that are not mentioned in the loops of these transitional NGs are the arrival process and the computational process. The arrival process will eventually cause $|W|$ to exceed $H$ and the computation process will eventually cause $I$ to empty. This TNG terminates as soon as one of the processes causes the violation of the WHILE condition.

If the $|W|$ condition is violated then we can start $PNG_1$ in $g = \{H, |G| - H, 0\}$. If the $I$ condition is violated then $I$ is empty, and $|G| \leq H$, so we can start $PNG_1$ in
This algorithm will take some finite time to change the system state to one valid under $PNG_1$. Specifically the average normalised time under $TNG_{N,1}$ will be

$$\tau_{N,1} = \min\left( H \rho^2, \frac{R^2 \rho^2}{1 - R \rho} \right)$$

where the first term comes from the $H$ arrivals constraint, and the second term is the average number of tasks in the queue under $PNG_N$ multiplied by the average imprecise computation time, then normalised.

**Swapping from $PNG_H$ to $PNG_1$ via $TNG_{H,1}$**

As can be seen from Figure 4.1 the system states are very similar under $PNG_H$ and $PNG_1$. This makes swapping between them simple. The only difference between $G_1$ and $G_H$ is the restriction on $|P|$. Under $PNG_H$ the size of $P$ can be anywhere between 0 and $\min(H, |G|)$, whereas under $PNG_1$ the size of $P$ can be only be 0 or $\min(1, |G|)$. Therefore if $|P| \leq 1$ under $PNG_H$ we can swap immediately. If however $|P| > 1$ we must reduce $|P|$ while maintaining the correct form of $\{W, I\}$. The rule for converting from $PNG_H$ to $PNG_1$ is therefore

**ALGORITHM 4.3.5 ($TNG_{H,1}$)** The transitional NG for converting from $PNG_H$ to $PNG_1$ is

**WHILE** $|P| > 1$ **DO**

**EXECUTE** $ICP_H$

The terminating condition of $TNG_{H,1}$ guarantees that $P$ contain at most one task. The imprecise notices are generated using $ICP_H$ because of its generality in terms of the number of tasks with precise notices.

Again a result is not possible for $\tau_{H,1}$ because we need to know the queue length distribution. We can upper bound $\tau_{H,1}$ as the average time required to compute $H - 1$
4.3.4 Swapping to $PNG_H$ via $TNG_H$

Again due to the more complex nature of $g_1$ compared with $g_A$ and $g_N$ we consider each possible $PNG_{bs}$ separately.

Swapping from $PNG_A$ to $PNG_H$ via $TNG_{A,H}$

The maximum size of $P$ under $PNG_H$ is $H$. $TNG_{A,H}$ must therefore reduce the size of $P_{bs}$ if $P_{bs}$ is greater than $H$. All new arrivals will be members so $\{W_{ss}, I_{ss}\}$ because no new precise notices will be given. The distribution of new arrivals into $\{W_{ss}, I_{ss}\}$ must be made in a way that will result in $\{W_{ss}, I_{ss}\}$ compatible with $ICP_H$. $ICP_H$ works by moving the oldest task in $W$ into $I$ when $|W|$ becomes greater than $H$. The rule for converting from $PNG_A$ to $PNG_H$ is therefore

**ALGORITHM 4.3.6 ($TNG_{A,H}$)** The transitional NG for converting from $PNG_A$ to $PNG_H$ is

```markdown
WHILE $|P| > H$ DO
    EXECUTE $ICP_H$
```

The terminating condition will result in the appropriate reduction in $P$. The imprecise notice generation is performed using the $ICP$ defined for the system under $PNG_H$. Although $ICP_H$ is designed to work with $P \leq H$ it is general enough to work with the arbitrarily size $P$ produced under $PNG_A$.

The average time taken for this algorithm to change the system state is the average time required to compute $|P_{bs} - H|$ tasks precisely. We need to know the distribution of the queue length to be able to calculate the average value of $|P_{bs} - H|$.

$$\tau_{H,1} \leq (H - 1)$$
Queue length distributions are not studied here. Therefore a result for $r_{A,H}$ is not possible.

**Swapping from $PNG_N$ to $PNG_H$ via $TNG_{N,H}$**

Under $PNG_N$ the generalise system state is $g = (0, |I_{bs}|, 0)$. In this case $P_{bs} = \emptyset$ which will stay empty at least until $I_{bs}$ is emptied. This is because we never have the case where tasks with imprecise notices are followed by tasks with precise notices in the queue. The only way that $|P|$ can be zero and $|I|$ non-zero under $PNG_H$ is if $|W| = H$. This means that we can swap after $H$ arrivals if $|I|$ is still non-zero. What happen if $I$ is emptied before $H$ arrivals occur? Well, in this case we can swap as soon as $I$ is emptied because $|G|$ will be less than $H$ and no tasks in the system will have notices. The rule for converting from $PNG_N$ to $PNG_H$ is therefore

**Algorithm 4.3.7 ($TNG_{N,H}$)** The transitional NG for converting from $PNG_N$ to $PNG_H$ is

```
WHILE $|W| < H$ AND $I \neq \emptyset$ DO
    do nothing
    IF $I = \emptyset$
        issue precise notices to all tasks
```

The terminating condition is reliant on two expressions. One of which is guaranteed to occur. The two processes that are not mentioned in the loops of these transitional NGs are the arrival process and the computational process. The arrival process will eventually cause $|W|$ to exceed $H$ and the computation process will eventually cause $I$ to empty. This TNG terminates as soon as one of the processes causes the violation of the WHILE condition.

If the $|W|$ condition is violated then we can start $PNG_H$ in $g = \{H, |G| - H, 0\}$. If the $I$ condition is violated then $I$ is empty, and also $|G| \leq H$, so we can start $PNG_H$ in $g = \{0, 0, |P|\}$.
This algorithm will take some finite time to change the system state to one valid under $PNG_H$. Specifically the average normalised time under $TNG_{N,H}$ will be

$$
\tau_{N,H} = \min \left( H\rho, \frac{R^3 \rho^2}{1 - R\rho} \right)
$$

where the first term comes from the $H$ arrivals constraint, and the second term is the average number of tasks in the queue under $PNG_N$ multiplied by the average imprecise computation time, then normalised.

Swapping from $PNG_1$ to $PNG_H$ via $TNG_{H,1}$

As can be seen from Figure 4.1 the system states are very similar under $PNG_H$ and $PNG_1$. This makes swapping between them simple. The only difference between $G_1$ and $G_H$ is the restriction on $|\mathcal{P}|$. When the number of tasks in the system is greater than $H$, $|\mathcal{I}_{bs}|$ under $PNG_1$ will be non-zero. In this state the system can be swapped immediately to $PNG_H$ because the system state is valid under both $PNG_1$ and $PNG_H$. If $\mathcal{I}_{bs}$ is empty then there must be less than $H$ tasks in the system ($|G_{bs}| \leq H$). Under $PNG_H$ all tasks would have precise notices when $|G| \leq H$. The rule for converting from $PNG_1$ to $PNG_H$ is therefore

**ALGORITHM 4.3.8 (TNG_{1,H})** The transitional NG for converting from $PNG_1$ to $PNG_H$ is

IF $|\mathcal{I}_{bs}| = \emptyset$

issue all tasks in $\mathcal{W}$ with precise notices

We see that swapping from $PNG_1$ to $PNG_H$ is immediate.

We have described algorithms for converting between any of the four notice generators studied and derived a measure of the speed of the conversion. The swapping of PNGs allows the system to manage the task load more dynamically than would be possible with a fixed scheme. In the next section we present an application of this scheme swapping feature.
4.4 Transitional Notice Generator Example

In this section we validate the use of transitional notice generators by way of example. We present an example of a system that uses two PNGs, one for normal load and one for high load. Real-time simulation is implemented via the SIMSCRIPT II event driven simulation language. The task generation, control schemes, and performance recording were all implemented using this language.

In a practical system imprecise computation will probably not be necessary during periods of normal load. We use $PNG_A$ as the precise notice generator to manage normal load. We select $PNG_H$ to be the precise notice generator for the high load periods. As we have seen in Chapter 3, $PNG_H$ does not sacrifice task precision as much as some of the other schemes unless $H$ is very small. The adjustment of $H$ should enable the system to handle most high loads effectively.

The load presented to the system is modelled as a periodic variation between normal load ($\rho_n$) and high load ($\rho_h$). This variation is achieved by varying the arrival rate between $\lambda_n = 1$ for the low load and $\lambda_h = 2$ for the high load. The mean computation times of the mandatory and optional parts are fixed at $1/\mu_m = 1/\mu_o = 3/8$, yielding $R = 0.5$, $\rho_n = 0.75$, and $\rho_h = 1.5$.

The time variation of the load is shown in Figure 4.2. The queue length is plotted versus time for the system using both PNGs in Figure 4.4, and also for the system using only $PNG_A$ in Figure 4.3. We can see that the performance under $PNG_A$ alone is unsatisfactory. However, the performance under the multi-PNG scheme is acceptable. In such a system the increase in load must be detected by some external process. We can view this PNG switching technique as second order dynamic control where the first order control is executed by $PNG_1$ and $PNG_H$. The flexibility of the system is quite impressive with any arbitrary choice of four PNGs applicable at any time during system run-time.
4.5. Summary

In this chapter we have studied the application of the four precise notice generators defined in Chapter 3. We have described algorithms that allow us to swap control schemes during run-time. Less sophisticated methods would cause the generation of notices to be suspended. In our swap method task notification is continuous. We presented an example where the benefits of being able to swap schemes during run-time were demonstrated.
Figure 4.3: Queue Length versus Time for System Under $PNG_A$

Figure 4.4: Queue Length versus Time for System Under $PNG_A$ and $PNG_H$
Chapter 5

Conclusions and Future Work

5.1 Conclusion

We have proposed a notice generation approach to decide on task execution precision in dynamic imprecise soft real-time systems. Four notice generators have been designed, analysed and evaluated. We have categorised these four into the static and dynamic notice generators, based on whether they use system state variables $N_s$ (the number of tasks in the system) and $N_p$ (the number of tasks that have precise notices) in their notice generation process. $PNG_A$ and $PNG_N$ are static while $PNG_1$ and $PNG_H$ are dynamic.

We proposed three performance metrics in order to evaluate the performance of the system controlled by these four notice generations. The metrics were mean waiting time, mean computation time and mean time until notice generation, all were normalised with respect to the mean precise computation time. To analyse the systems the state-transition-rate theory was used to construct state-transition-rate models of the system under each of the four schemes. The performance metrics were calculated explicitly from the state probabilities, and some performance limits were derived. Simulation via SIMSCRIPT II verified the numerical method and the limits.
The performance evaluation showed that PNG$$N$$, a static notice generator, offered the most responsive service; its waiting time is the lower bound of the imprecise system. This was expected because no task is every executed precisely under PNG$$N$$. The other static notice generator PNG$$A$$ provided the highest computation quality due to the guaranteed precise execution provided by PNG$$A$$. The dynamic notice generators PNG$$H$$ and PNG$$I$$ offered performance in between the two static PNGs. The most important feature of the dynamic group performance was that the mean task waiting time was kept low when the system was overloaded, and at the same time maintained at a reasonable computation quality. This is a direct consequence of using imprecise computations. This performance was not observed in the static group.

We observed that the static PNGs should only be used when the load on the system is at some extreme value. Specifically, PNG$$N$$ should be used when the load is very high, and PNG$$A$$ when the load is consistently low. Under PNG$$N$$ the task result quality is determined by $$R$$. All loads between the extremes would be better served by one of the dynamic PNGs. PNG$$H$$ offered service with a good quality bias, and PNG$$I$$ offered service with a good response time bias.

The external system was provided with advanced information with regard to task result accuracy in all four schemes. The static PNGs performed best in this area because task execution precision is deterministic under these two schemes. The notice generation performance of the dynamic group was not as good as the static group. PNG$$H$$ was able to provide notice more quickly than PNG$$I$$ in general.

There is no clear winner in the performance of the schemes. The choice of a scheme for a particular system must take into account the operational requirements of the system.

The proposed notice generators operating in an imprecise computation environment can be applied to a number of traditionally more difficult applications. One example is in managing transient increase in computation load occurring in computer controlled industrial process during a system initialisation or an operation emergency [36]. Others include time/accuracy tradeoffs in intelligent real-time control in which
some artificial intelligence or computational intensive Kalman filter-type algorithms may be used in order to meet difficult mission requirements.

To offer more system configuration flexibility a set of methods to swap between any of the four PNGs during run-time was developed. The complexity involved is the preservation of system state continuity during the scheme swap. The methods constructed achieved this continuity and preserved the aims of the general soft real-time system while the system state was converted to one valid under the new PNG. This technology was applied to an example were the benefits of dynamic PNG selection were demonstrated to be significant.

5.2 Future Work

This work could be extended in many directions. Perhaps the most interesting would be the construction of an algorithm that could choose between the a menu of control schemes based on some objectives. Although the mechanism for swapping between schemes was presented in this thesis the criteria for initiating the swap was not discussed in detail.

In a real system there may be periodic tasks embedded in the soft real-time tasks that we have considered in this work. Of course the current schemes would serve these periodic tasks, but if they had deadlines then these could not be guaranteed by the current form of the control schemes. Perhaps some hybrid schemes could be constructed where the deadlines of the periodic tasks are met but the soft real-time tasks are not compromised. This field has already attracted some consideration.

A physical system will be the ultimate proof of the effectiveness of the imprecise computation technique. There are many realisations of the imprecise computation system environment, some more difficult to implement than others. Perhaps a network of two workstations with dummy loads and some sort of real–operating system may be a possibility.
Bibliography


