PERFORMANCE MODELLING AND EVALUATION OF PROTOCOLS
BASED ON FORMAL SPECIFICATIONS

By

Sijian Zhang
B. Sc. (Computer Science) Beijing University, China
M. Sc. (Computer Science) Beijing University, China

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES
COMPUTER SCIENCE

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
July 1995
© Sijian Zhang, 1995
In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Computer Science
The University of British Columbia
2366 Main Mall
Vancouver, Canada
V6T 1Z4

Date: Aug. 30, 1995
Abstract

Protocol performance issues are important in communication protocol design and network management, especially for those protocols which run in high-speed networking environments. An accurate performance modelling and evaluation approach is necessary to obtain reliable performance estimations and to improve system performance. Using queueing models (QMs) or finite state machines (FSMs) alone is difficult to achieve this goal because many aspects that affect performance are not taken into consideration by the model.

This thesis proposes a new performance model called *performance extended finite state machine* (PEFSM). PEFSM makes use of the strengths of both QM and FSM. A PEFSM is based on an FSM which is extended to include *time* and *probability*. Furthermore, the transition time is refined and divided into two parts: the transition wait time and the transition service time. This allows the PEFSMs to integrate message arrival and queueing models which provide useful and essential information necessary for studying real world protocols.

PEFSMs are classified into three categories based on the message arrival characteristics: synchronous PEFSMs (SyPEFSMs), asynchronous PEFSMs (AsPEFSMs) and hybrid PEFSMs (HyPEFSMs). While the hybrid model is the most useful and realistic model for communication protocols, the other two models are also presented for completeness, and as a way to explain the hybrid model. A method for computing performance metrics based on SyPEFSMs is given in the thesis. Two types of AsPEFSMs — AsPEFSM-α and AsPEFSM-β — and their performance evaluation methods are also presented. Then a class of HyPEFSM which is a hybrid model of SyPEFSM and AsPEFSM-α is introduced. The pro-
posed performance evaluation method for this class of HyPEFSM is basically the combination of those for SyPEFSMs and AsPEFSM-αs. Our performance modelling and evaluation approach has been applied to various examples, including the alternating bit protocol and multi-stage interconnection network (MIN).

The performance evaluation method for PEFSMs makes use of stochastic process and queueing theory. A new queueing property for an M/G/1 with multiple job classes and an analogous property for AsPEFSM-αs have been discovered and proved.

As a first step in improving system performance, the thesis defines software performance bottlenecks based on PEFSMs. Two bottleneck identification methods are proposed and tested.

This thesis also proposes a testing method called t-test which in most cases is able to obtain the service times of invisible transitions when the protocol implementation under test is given as a black box. Transition service times are important parameters in FSM-based performance models. Studies in the past have usually assumed the transition times to be known a priori without discussing how they may be obtained.

Simulations and measurement experiments have been conducted to validate the methodologies proposed in this thesis. The results are quite promising.
Contents

Abstract ii
Contents iv
List of Tables viii
List of Figures ix
Acronyms and Terms xi
Acknowledgement xiii
Dedication xv

1 Introduction 1
1.1 Background ........................................ 1
1.1.1 Objectives ...................................... 2
1.2 Queueing Models ..................................... 3
1.3 Formal Description Techniques .................... 4
1.4 Problems ............................................ 6
1.5 Proposed Solution Approaches ..................... 7
1.6 Summary of Contributions ........................ 10
1.7 Layout of The Rest of The Thesis ................. 11

2 Survey of Related Work 13
2.1 Introduction .......................................... 13
2.2 Modeling Approaches .............................. 15
   2.2.1 Performance models ......................... 15
   2.2.2 Queueing models vs FSMs .................. 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.1</td>
<td>$\alpha$-arrival model</td>
<td>81</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Queueing and service model</td>
<td>81</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Performance Metrics</td>
<td>84</td>
</tr>
<tr>
<td>5.3</td>
<td>AsPEFSM-$\beta$</td>
<td>99</td>
</tr>
<tr>
<td>5.3.1</td>
<td>$\beta$-arrival model</td>
<td>99</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Queueing and service model</td>
<td>102</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Performance metrics</td>
<td>102</td>
</tr>
<tr>
<td>5.4</td>
<td>Bounds of Performance Measures</td>
<td>111</td>
</tr>
<tr>
<td>5.5</td>
<td>Related work</td>
<td>111</td>
</tr>
<tr>
<td>5.6</td>
<td>Summary</td>
<td>113</td>
</tr>
<tr>
<td>6</td>
<td>Performance Evaluation of Hybrid PEFSMs</td>
<td>114</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>114</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Hybrid transition</td>
<td>115</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Hybrid state</td>
<td>116</td>
</tr>
<tr>
<td>6.2</td>
<td>Models and Evaluation</td>
<td>119</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Message arrival model</td>
<td>119</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Queueing and service model</td>
<td>120</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Performance metrics</td>
<td>120</td>
</tr>
<tr>
<td>6.3</td>
<td>An Application Example</td>
<td>125</td>
</tr>
<tr>
<td>6.4</td>
<td>Summary and Discussion</td>
<td>129</td>
</tr>
<tr>
<td>7</td>
<td>Analysis of Performance Bottlenecks using PEFSMs</td>
<td>133</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>134</td>
</tr>
<tr>
<td>7.2</td>
<td>Throughput Bottleneck</td>
<td>138</td>
</tr>
<tr>
<td>7.3</td>
<td>Queue Wait Time Bottleneck</td>
<td>139</td>
</tr>
<tr>
<td>7.4</td>
<td>Related Work</td>
<td>150</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary</td>
<td>152</td>
</tr>
<tr>
<td>8</td>
<td>On Transition Time Testing based on FSMs</td>
<td>154</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>154</td>
</tr>
<tr>
<td>8.2</td>
<td>T-test</td>
<td>157</td>
</tr>
<tr>
<td>8.3</td>
<td>Testability</td>
<td>166</td>
</tr>
<tr>
<td>8.4</td>
<td>Test Sequence Generation</td>
<td>172</td>
</tr>
<tr>
<td>8.5</td>
<td>Experimental Results</td>
<td>178</td>
</tr>
<tr>
<td>8.6</td>
<td>Summary</td>
<td>181</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions and Future Work</td>
<td>184</td>
</tr>
</tbody>
</table>
9.1 Conclusion .................................................. 184
  9.1.1 Contributions ........................................ 185
  9.1.2 Applicability of PEFSM to EFSM .................... 188
9.2 Future Work .............................................. 190
  9.2.1 System-wise performance evaluation .................. 190
  9.2.2 Queueing theory ..................................... 192
  9.2.3 New topics in stochastic processes .................. 193
  9.2.4 On-line performance tuning ......................... 193

Bibliography .................................................. 195

A Mathematical Background .................................. 209
  A.1 Introduction ........................................... 209
  A.2 Markov Process ........................................ 210
  A.3 Continuous-time Semi-Markov Process .................. 212
    A.3.1 Steady-state state probability .................... 213
    A.3.2 Limiting state lodging probability ................. 213
    A.3.3 First passage time ................................ 214
    A.3.4 State Recurrence .................................. 214
    A.3.5 Transition recurrence .............................. 215

B Computation of $\overline{\nu_{ij}}$ .......................... 217

C An Estelle specification for sliding window protocol .... 219

D An Estelle specification of the hot-potato protocol .... 227

E An application example of HyPEFSM ....................... 230

F Bottleneck identification results ........................ 237

G T-test results of a LAPB implementation .................. 240
List of Tables

2.1 The relationship of Modeling and Analysis approaches .................. 27
6.1 Comparison of analytical result and simulation result .................. 128
7.1 A simulation result of bottleneck identification ......................... 149
8.1 Simulation and Testing Results ............................................ 181
F.1 A simulation result .......................................................... 238
G.1 Testing results of the LAPB implementation ............................... 241
# List of Figures

2.1 An example of a queueing model ........................................... 21  
2.2 An example of an FSM .................................................. 23  
3.1 An FSM of the alternating bit protocol .................................. 40  
3.2 An FSM of CCITT/ISO X.25 LAPB ....................................... 41  
3.3 An abstract FSM ........................................................ 44  
3.4 The components of a transition time ................................... 47  
3.5 An FSM of the hot potato protocol ..................................... 55  
3.6 An FSM of an ATM serving two types of data ....................... 56  
3.7 Model $W$ of the PEFSM in Figure 3.3 .................................. 60  
5.1 The queueing system for an AsPEFSM-$\alpha$ .......................... 83  
5.2 Reduction of a general FSM into a one-state EFSM ................. 88  
5.3 Messages come to and leave state $i$ in an AsPEFSM-$\alpha$ .......... 93  
5.4 The queueing system for an AsPEFSM-$\beta$ ............................ 102  
6.1 An FSM of the alternating bit protocol ............................... 115  
6.2 The possible situations of two transitions from a state ............ 117  
6.3 Mean queue wait time vs. mean data arrival rate ..................... 129  
6.4 Mean queue wait time vs. mean data arrival rate ..................... 129  
6.5 Mean queue wait time vs. packet loss probability .................... 130  
6.6 Modelling of a hybrid state ........................................... 131  
7.1 Service order implied by the FSM of a PEFSM ....................... 141  
7.2 The subpaths to transition $ij$ in a PEFSM ............................ 143  
7.3 Algorithm for weight computation ..................................... 146  
7.4 A simulation architecture ............................................. 147  
8.1 A sample conceptual testing architecture ............................. 157
## Acronyms and Terms

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi$</td>
<td>Vector of steady state probabilities of a PEFSM</td>
</tr>
<tr>
<td>$\pi_i$</td>
<td>Steady-state probability of state $i$ in a PEFSM</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Utilization</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Throughput rate</td>
</tr>
<tr>
<td>$e_i$</td>
<td>Entrance rate of state $i$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Arrival rate of incoming messages</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>Arrival rate of class $i$ incoming messages</td>
</tr>
<tr>
<td>$p_{0,i}$</td>
<td>Probability that protocol is idle in state $i$</td>
</tr>
<tr>
<td>$\theta_{ij}$</td>
<td>First passage time from state $i$ to $j$ in a CSMP</td>
</tr>
<tr>
<td>$\psi_{rij}$</td>
<td>Transition time for transition $ij$ in a PEFSM</td>
</tr>
<tr>
<td>$w_{rij}$</td>
<td>Transition wait time for transition $ij$ in a PEFSM</td>
</tr>
<tr>
<td>$s_{rij}$</td>
<td>Transition service time for transition $ij$ in a PEFSM</td>
</tr>
<tr>
<td>$W$</td>
<td>Matrix of probability density functions of Transition Wait Times</td>
</tr>
<tr>
<td>$H$</td>
<td>Matrix of probability density functions of transition service times</td>
</tr>
<tr>
<td>$V^H$</td>
<td>Matrix of probability density functions of transition times</td>
</tr>
<tr>
<td>$M$</td>
<td>An FSM</td>
</tr>
<tr>
<td>$Q$</td>
<td>The state set of an FSM or PEFSM</td>
</tr>
<tr>
<td>$\Psi$</td>
<td>A PEFSM</td>
</tr>
<tr>
<td>$P$</td>
<td>Matrix of single-step transition probabilities of a PEFSM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AM</td>
<td>Algebraic Model</td>
</tr>
<tr>
<td>AsPEFSM</td>
<td>Asynchronous PEFSM</td>
</tr>
<tr>
<td>ASP</td>
<td>Abstract Service Primitive</td>
</tr>
<tr>
<td>CCITT</td>
<td>International Telegraph and Telephone Consultative Committee</td>
</tr>
<tr>
<td>CSMP</td>
<td>Continuous-time Semi-Markov Process</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>DSMP</td>
<td>Discrete-time Semi-Markov Process</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>EFSM</td>
<td>Extended Finite State Machine</td>
</tr>
<tr>
<td>HyPEFSM</td>
<td>Hybrid PEFSM</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>IUT</td>
<td>Implementation Under Test</td>
</tr>
<tr>
<td>LAPB</td>
<td>Link Access Procedure B</td>
</tr>
<tr>
<td>MP</td>
<td>Semi-Markov Process</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>PCO</td>
<td>Points of Control and Observation</td>
</tr>
<tr>
<td>p.d.f.</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PEFSM</td>
<td>Performance Extended Finite State Machine</td>
</tr>
<tr>
<td>PTE</td>
<td>Protocol Testing Environment</td>
</tr>
<tr>
<td>PN</td>
<td>Petri Net</td>
</tr>
<tr>
<td>QM</td>
<td>Queueing Model</td>
</tr>
<tr>
<td>SAP</td>
<td>Service Access Point</td>
</tr>
<tr>
<td>SMP</td>
<td>Semi-Markov Process</td>
</tr>
<tr>
<td>SUT</td>
<td>System Under Test</td>
</tr>
<tr>
<td>SyPEFSM</td>
<td>Synchronous PEFSM</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>TL</td>
<td>Temporal Logic</td>
</tr>
<tr>
<td>X.25</td>
<td>CCITT Packet-switching Protocol</td>
</tr>
</tbody>
</table>
Acknowledgement

I would like to thank my research supervisor, Dr. Sameul T. Chanson. Without his guidance and support over the past several years, I could hardly finish this thesis. Sam is responsible and he has never disrupted his supervision even though he is on-leave at thousands of miles away. Throughout the meetings and discussions with Sam, I have learned a lot from him. I am always prodded to think of more and better results by the motive of his serious, strict and critical attitude in research. This I believe in turn helped me to come up with the results in this thesis.

I would also like to thank Dr. Hong Chen, Dr. Norm Hutchinson and Dr. Mabo R. Ito for their willingness to serve on my supervisory committee. Dr. Hong’s joining increased my confidence when I met the stochastic process problems and found myself was new to this area. Besides, I would like to present my special gratitude to Dr. Tao Yang (Dept. of Industrial Engineering, Tech. Univ. of Nova Scotia). Although we have never met until today, he has been very helpful. His expertise and insight provided the useful clue to the short proof of the equality of mean queue wait times using PASTA.

I thank Dr. Son Vuong for his encouragements and discussions, particularly during the early days of my graduate program at UBC. He is one of the most easygoing professors in the department and he always let me borrow the reference material he collected.

I am very grateful to Baokang Chen, Joel DeYoung, Colin Hawes, Christopher Healey, Dr. Runping Qi, Dr. Ying Zhang and Jinsong Zhu. They spent lots of time and effort on proofreading the early versions of my thesis. Without their
proofreadings, this thesis would not be so readable. Many other fellows have also provided various help during my staying at UBC. I highly appreciate their long lasting friendships.

I particularly thank my family which has given me so much. Their understanding and faith always keeps me going. I feel deeply indebted to my wife, Yuanmeng Lu, for her dedicated love and patience.

Last but not least, I would like to thank UBC which provides so enjoyable living and studying environments for its students.
... to my father and his elder brother
Chapter 1

Introduction

1.1 Background

The performance of a computer communication system is determined by two factors: protocol performance and network performance. The first factor refers to the performance of the raw data processing of a communication protocol (for example, TCP and ISO TPs) on a computer. The second factor refers to the performance of data transmission between computers.

We have seen a rapid evolution of computer networks in the past few years. The use of optical fiber has greatly improved the bandwidth of communication channels, making data transmission rates of several gigabits per second (Gbps) possible. As a result, transmission delays between computers have been significantly reduced. For instance, a packet of length 1KB takes only 8 microseconds to transmit on a 1 Gbps line.

Unfortunately, the improvement in processing speed of protocols has been
much less impressive. Although computer architectures, operating system support and new communication protocols have been developed in an effort to support high speed communications [Clark85, Giar89, Hehm89, Watson89, Zitt89, Doer90, AbKa91], the CPU processing time of protocol packets is still of the order of tens of microseconds, on average [Svob89, Clark89, Nich91].

As a consequence, the performance bottlenecks in computer communications have shifted from the communication links to the communicating hosts that are unable to process, send and receive data as fast as the communication channels can transmit the data [Clark89]. Therefore, the performance of communication protocols is becoming an important issue.

1.1.1 Objectives

Prior to the implementation and deployment of a protocol, it is very desirable to know things like: how well the protocol will perform in certain operating environments, where the potential software bottlenecks of the protocol are, and how the performance can be improved. For example, given the workload of the communication network, what is the ideal buffer size to store incoming messages for each protocol entity? What are the service delays and queue wait delays for each type of incoming message? Which processing path is the most critical for improving the performance of the protocol? What is the network traffic generated by a protocol entity? The answers to these and other performance related questions are useful in understanding and improving communication performance. Therefore, a proper performance modelling and evaluation method is needed in the protocol design phase as well as in tuning the performance of protocol implementations.
1.2 Queueing Models

In general, there are two approaches to evaluate the performance of a computer system: simulation and analytic. Whichever way is chosen, performance modelling is always the first step.

Traditionally, performance modelling is often done using queueing models. A single server queueing system is usually adopted to represent the computer on which a protocol entity is running. The service time of the messages in this queueing system is assumed to be independent and identically distributed (i.i.d) (e.g., exponential distribution). The arrivals of the messages to the queueing system are also assumed to be i.i.d. in most cases to simplify the analysis.

In a computer network, however, the actual arrival pattern of messages to a computer is quite complex. This traffic is usually managed by a combination of flow control, rate control, congestion control and other mechanisms defined in a network protocol. Moreover, the order in which the messages are served depends on the protocol's state. A particular class of messages of a protocol is served only when the protocol is in a specific state. This important information is neglected in queueing models for analyzing protocol performance. Although simplifications make the conventional models tractable, they lead to inaccurate results. As computer networks are becoming faster, this inaccuracy becomes more and more significant.

A new performance model is needed. The model should take into account more systematic information about a protocol during performance evaluation than a queueing model does.
1.3 Formal Description Techniques

Formal Description Techniques (FDTs) provide some answers to the problems discussed in the previous sections.

Until recently, almost all communication protocols have been specified in natural languages, often resulting in different interpretations by different users of the same protocol. FDTs were introduced as a means of eliminating or minimizing ambiguity in describing communication protocols and distributed systems.

Many FDTs have been developed and used. One may categorize FDTs into two groups: formal models such as finite state machines (FSMs), Petri Nets, algebra and temporal logic, and formal specification languages such as SDL, Estelle and LOTOS.

Unlike general-purpose computer programming languages, a formal specification language is usually designed to describe a certain formal model. For example, a formal specification in SDL or Estelle describes an extended FSM (EFSM), and a specification in LOTOS is based on a calculus of communicating systems (CCS) and communicating sequential processes (CSP). A formal specification language provides the functionality of variable definitions which formal models usually lack. Up to now, three formal languages were standardized by ISO and CCITT for descriptions of communication protocols: Estelle[ISO8807], LOTOS[ISO9074] and SDL[SDL].

Briefly, FDTs provide a formal way of

- specifying a communicating protocol unambiguously,
• validating the design of a protocol,

• generating test cases for conformance testing,

• simulating the behaviour of a protocol before implementation,

• implementing a protocol semi-automatically, and

• predicting the performance of a protocol.

With a specification in an FDT, validation and verification as well as implementation can be done automatically or semi-automatically [Vuong86, Azema89]. Test sequences for conformance testing can also be derived from a formal specification [Sari86, Sari87, Ural87a, Wu89]. In the early 1980s, some basic ideas were proposed by Rudin to use a formal protocol definition as the basis for the automated prediction of protocol performance [Rudin83]. Since then, other research on this subject has appeared in the literature (see, for example, [Krit87, BoVa88, LiLi88, Shaw89, GuRu89, AFF90, FVV91]). Chapter 2 contains a survey of this research and other work on protocol performance analyses.

FDTs are useful in performance studies and provide different levels of abstraction for both analysis and simulation of protocols. Furthermore, a formal specification or a formal model explicitly describes the important aspects of a communication protocol systematically. These aspects influence system performance. For example, from a given specification of a sliding window protocol (see Appendix C for details), one can identify the factors which affect performance such as window size, timeout value, transition probabilities and data arrival rates. This is an important advantage of FDTs. [ZhCh93] proposed an approach to evaluate protocol performance considering the amplifier effect, i.e., one input and
multiple outputs. The coefficient matrix of an amplifier is derived from a given Estelle specification.

To exploit the power of FDTs, it is useful to have a framework for systematic performance analyses. This is called single-specification / multiple-techniques in [Heck91]. More and more system designers using FDTs find the need for performance tools associated with FDTs. Developing performance models and tools based on an FDT is the general objective of this thesis.

1.4 Problems

Some research using FDTs for performance evaluation of protocols already exists [AFF90, Rico91, KrWh93]. The main problem is that either the models are incomplete or insufficient to capture the requirements discussed above, or the analyses are computationally expensive or inaccurate.

For example, [KrWh93] treats the FSM of a protocol as a discrete-time semi-Markov process and then predicts the performance of the protocol. However, this approach does not model the arrival process and does not study queueing properties, which are important performance indices for most communication protocols and computer systems. For this reason, performance is expressed only in the form of bounds.

The earlier work [Krit86, Krit87] did not take information about the state-dependent service of a protocol into account. All the messages coming to/from a protocol entity were treated aggregately. As a result, only the overall queueing delay of messages and the overall queue length of a protocol entity could be computed. Furthermore, since no state-dependent service is considered, it is
difficult to have an accurate estimate of the performance.

Other research such as [LiLi88, BaBu91] uses validation/verification methods (mainly reachability analysis) for performance analysis. This approach inevitably runs into the state space explosion problem because the size of a reachability tree grows exponentially. A great amount of effort has been spent on reducing the size of the state space in this kind of approach. However, the reduction techniques further increase the inaccuracy of performance measures. Moreover, this reachability approach usually cannot derive queueing characteristics.

1.5 Proposed Solution Approaches

To address the above problems, a new performance model called Performance Extended Finite State Machine (PEFSM) is proposed. We introduce this model to reduce the computational complexity of performance analysis of a protocol without sacrificing too much accuracy. This performance model is essentially a hybrid model combining the FSM and queueing models.

FSMs are chosen because they provide a reasonable abstraction about a communication protocol and are used commonly in communication systems. Other models are often transformed into FSMs before analysis. Examples include Petri Nets [Moll82] and LOTOS [Mars94]. More discussion is given in Chapter 2.

FSMs are also chosen because many communication protocols are given directly in this form. Using FSMs, the performance modelling and analysis are more straightforward and the performance indices are easier to understand. Among the three standardized formal specification languages, two of them, Estelle and SDL, are based on FSMs or extended FSMs. There exist many support tools for FSMs.
A disadvantage of FSMs is that they do not model arrival processes, resource contentions and queue wait delays easily. Thus, our modelling approach tackles this problem by taking advantage of the strengths of both queueing models and FSMs.

Similar to previous work in [LiLi88, BoVa88, KrWh93], the FSM of a PEFSM is extended with transition time and transition probability for performance evaluation. Each transition of the FSM is associated with an incoming and an outgoing message class, labeled with I/O. A transition time in a PEFSM is divided into two parts: the wait time for an incoming message associated with the transition and the service time to execute the transition. Transition wait times are related to the arrival process of incoming messages, so an arrival model is associated with each PEFSM. The transition wait times of a PEFSM are computed from the given arrival model and the PEFSM.

Arrival models depend on assumptions about the environment in which the communication protocols run. There are three basic types of arrival patterns for incoming messages: synchronous, asynchronous or a combination of the two. Accordingly, PEFSMs are classified into three categories: synchronous PEFSMs (SyPEFSMs), asynchronous PEFSMs (AsPEFSMs) and hybrid PEFSMs (HyPEFSMs). The classification of a PEFSM is determined by the arrival model used.

A queue is assigned to each class of incoming messages in a PEFSM. This makes the analysis of queue wait delays possible. Queueing theory is employed to compute these delays. However, note that the queue wait delay of a message includes not only the service times of the messages in front of it in the same queue but also possibly other transition times of transitions which have to be
processed before this message. This is because the service in a protocol entity is state-dependent. Therefore, some input values for the queueing analysis will come from the result of analyzing the FSM of the associated PEFSM.

To compute the statistics of the FSM, the state changing process of an FSM is examined. This process is actually a stochastic process and thus the theory of stochastic processes is employed. In this thesis, we only study the stochastic processes which are continuous-time semi-Markov processes (CSMPs) [Howa71]. More complex stochastic processes are left for future research.

Software performance bottlenecks are also studied in the framework of PEFSMs. A bottleneck with respect to a performance metric is defined as a bottleneck transition in the FSM of a PEFSM. A methodology to identify these critical transitions in PEFSMs is proposed. This allows the designer to focus on the few transitions that constrain system performance.

This thesis also provides a solution to another fundamental issue. By definition, transition times are primitive data for a PEFSM. This data is essential in performance evaluation. However, previous work has assumed the transition times to be known a priori. How these values are obtained has widely been ignored. In the framework of performance evaluation for PEFSMs, we propose a technique to measure/compute transition times when an implementation of a protocol is given as a black box.

We have mentioned that in a PEFSM, transition time consists of transition wait time and transition service time. The first component is computed from a given arrival model and the PEFSM. The computation methods of transition wait times are given with respect to each type of PEFSM. To obtain transition
service times accurately through measurement, an algorithm for *transition time testing* called *t-test* is presented in the thesis. The methodology to generate test sequences for *t-test* is also proposed. The initial work on *t-test* was published in [ZhCh94].

### 1.6 Summary of Contributions

The contributions of this thesis can be summarized in three aspects:

- *performance prediction*,
- *performance improvement*, and
- *performance measurement*.

A new performance model — PEFSM together with the framework of performance evaluation based on this model — is proposed. The computation methodologies to obtain performance metrics of three classes of PEFSMs (SyPEFSMs, AsPEFSMs and HyPEFSMs) are developed. Accompanying this modelling approach, an interesting queueing property of the incoming messages in an AsPEFSM or a HyPEFSM is discovered and presented. This property is also used in deriving performance metrics.

To maximize the improvement of the performance of a protocol, software performance bottlenecks are defined based on PEFSMs. A method for the identification of transition bottlenecks is proposed. To our knowledge, no other similar research exists which tries to pinpoint potential software performance bottlenecks based on an FDT. The bottleneck identification problem was one of the
outstanding issues without a good solution. Our experimental results showed that the proposed method can accurately identify the bottleneck transition in a PEFSM with respect to a performance metric.

Finally, we suggest a methodology called the \textit{t-test} which can be used to obtain \textit{transition service times} through measurement and testing. The transition service times are important parameters needed in all FSM and EFSM based models but have been assumed to be known a priori in previous work.

1.7 Layout of The Rest of The Thesis

Chapter 2 provides a survey of related work, focusing mainly on performance evaluation methods related to FDTs. This survey serves in fact as a starting point for this thesis.

Chapters 3 to 6 introduce the framework for modelling and evaluating performance of communication protocols based on the new performance model, \textit{PEFSM}. Chapter 3 gives a formal definition of a general PEFSM. Three embedded models of a PEFSM are defined. This chapter provides the foundation for the analyses and performance metric derivations in the following three chapters.

PEFSMs are categorized into three classes, SyPEFSMs, AsPEFSMs and HyPEFSMs, according to incoming message arrival patterns. Chapter 4 deals with SyPEFSMs, which assume incoming messages arrive synchronously. Chapter 5 presents AsPEFSMs, which assume incoming messages arrive asynchronously. Chapter 6 deals with HyPEFSMs, which assume incoming messages arrive both synchronously and asynchronously. While the hybrid model is the most interesting and realistic one in the real world, the other two models are also presented
for completeness and as a way to explain the hybrid model in a systematic way. The mathematical background necessary for this study is given in Appendix A.

Chapter 7 defines software performance bottlenecks based on the PEFSMs. The methods to identify the bottlenecks are presented.

Chapter 8 presents the testing methodology for deriving the transition service times of a PEFSM accurately, although the protocol implementation is given as a black box.

Chapter 9 summarizes the contributions of this thesis and identifies some future research directions.
Chapter 2

Survey of Related Work

Abstract

This chapter provides a survey of the related work on performance evaluation for computer systems, focusing mainly on communication protocols. It examines the previous work of performance evaluation from three different points of view: modelling approaches, analytic methods and simulation tools. The conclusion of this survey is that queueing models (QMs) and finite state machines (FSMs) are complementary techniques in modelling communication protocols. A hybrid model combining QM and FSM is suggested as a suitable way to model protocols for performance evaluation.

2.1 Introduction

Performance evaluation of computer systems is not a new research area. Numerous evaluation methodologies have been developed and published, ranging from very theoretical and complex mathematical derivations such as queueing theory and automata theory to very practical approaches such as performance
benchmarks and simulators.

It is not our intention to present a wide range of issues and methods on performance evaluation or specific (ad hoc) performance modelling and analysis methods. Instead, we will focus on the work related to formal description techniques (FDTs) which may exist in two forms: formal models and formal specification languages.

Generally, the approaches to performance evaluation of a system can be classified into the following three broad categories:

1. analysis,

2. simulation, and

3. measurement

The first two approaches require performance models and could use a formal description of the system to be evaluated. Measurement is usually carried out on an implementation which is executable, and will not be covered in this chapter.

As mentioned in the previous chapter, FDTs are becoming more and more widely used in the design and development of large, complex computer networks and protocols. Rudin did pioneering work towards bridging the gap between formal specifications and the prediction of protocol performance [Rudin83]. He pointed out three possible approaches in the early 1980s:

1. analysis of a protocol to check for logical self-consistency, including time specifications;

2. simulation to predict performance;
3. analysis to predict performance.

Rudin noted that the first approach had proven to be the most difficult and little progress had been made at the time. However, much research has been done since (see, for example, [Drira95]) but mostly for verification purposes.

The latter two approaches are currently the most widely used approaches in performance evaluation based on FDTs. In the following sections, the techniques or methodologies in these two categories are surveyed. The survey is presented in three domains: modelling techniques, analytic methods and simulation methods.

The rest of the chapter is organized as follows. Section 2.2 compares different performance modelling approaches. Section 2.3 investigates the different analytic methods and their relationships with the modelling approaches mentioned in Section 2.2. Section 2.4 lists simulation tools according to their underlying models. Section 2.5 discusses software metrics, complexity problems and performance metrics. Section 2.6 summarizes the chapter.

2.2 Modeling Approaches

In both analytic and simulation studies, a performance model describing the computer system or communication protocol to be evaluated must be formulated. In this section, formal models based on FDTs are presented and their relative strengths and weaknesses in performance evaluation are examined.

2.2.1 Performance models

There are many ways to model communication systems. Some of them are general, while others are very specific. Some popular modelling techniques are:
• Queueing Model (QM)
• Finite State Machine (FSM)
• Petri Net (PN)
• Algebraic Model (AM)
• Temporal Logic (TL)

Queueing Models are perhaps the most traditional and also most popular way to model computer systems and analyze their performance. Queueing theory was first introduced and studied in operations research in the 1950s [Mors58]. Ever since John D. C. Little's formulation of what has become known as Little's Law [Litt61], namely

\[ L = \lambda \cdot W, \]

the applications of QMs in computer systems have proliferated. A great amount of work has been published since the 1970s, both in theory and in practice. One of the most popular and widely cited books on queueing theory in the past two decades was written by L. Kleinrock [Klein75].

QMs have become popular partially due to their simplicity in modelling complex computer systems. These models focus on resource contention. With this approach, all components are reduced to two basic constructs – queue and server. Servers represent resources shared and/or consumed by jobs. Queues represent buffers or lines where jobs wait to be served.

QMs are generally classified into two categories: queueing systems and queueing networks. A service center with only one server forms a queueing system.
Queueing systems that are interconnected together constitute a *queueing network*.

Queueing models model the behaviours of jobs in queues and servers as well as the overall system behaviours. The performance measures of a system being modeled are estimated by analyzing or simulating the model. These measures include *delays* (queueing delays and service delays), *server utilisations* and *throughputs*. Queueing models have been successfully used in modelling computer memory, disk subsystems, processors and computer networks [Lazo84]. Recently, queueing models that take into consideration the behaviour of the traffic sources have been used in analyzing the performance of *asynchronous transfer mode* (ATM) networks [DiDe90, Herr93, Kesi93].

**FSM**

*FSM*\(^1\) consists of states and transitions. FSMs originated from the study of computer languages and computability. However, there are many variations of FSMs whose applications are far beyond these original domains. Today, FSMs are one of the most widely used tools in modelling communication protocols [PhGr88, Tane88, DeBu89, Stall91, ISO2576, ISO7776]. This is because most communication protocols are designed as FSMs [Dan80].

FSMs augmented with variables are called *extended FSMs* (EFSMs). FSMs with associated times in their transitions or states are called *timed automata* [LyAt92]. FSMs with probabilities assigned to the transitions are called *probabilistic automata*.

With times and probabilities, FSMs (or EFSMs) can be used for performance modelling. One typical approach is found in the work of Kritzinger and his

\(^1\)FSMs are also called *finite state automata* in some literature.
colleagues. [Krit86] introduced a way to model OSI communication architectures using FSMs. [Krit87] presented a way to reduce the complexity of communication protocol analysis using image protocols. [KrWh93] transformed a timed and probabilistic FSM into a \textit{discrete-time semi-Markov process}. The theory of this stochastic process is then applied to derive the performance measures for the protocol specified by the FSM. Other work using FSMs as performance models can be found in [LiLi88, BoVa88, Heck91].

None of the above models considers \textit{message arrival processes} in performance modelling and evaluation. The transition times usually take only the service times of transitions into account and thus the performance measures derived are upper bounds.

\textbf{Petri Nets} are another useful tool in modelling communication systems. In general, Petri Nets can model control flow, concurrency and synchronization very well. This modelling technique has been intensively studied since it was first introduced by Petri over thirty years ago [Petri62]. A PN consists of a number of tokens, places and transitions. A great number of modified PN models, such as place-coloured PNs, predicate-transition PNs, time PNs and stochastic PNs have been proposed. [Diaz82] provides a general survey of PNs, and the introduction of [BeDi91] contains a brief survey of time PNs.

PNs have been used variously to model, validate and evaluate distributed systems, communication protocols and real-time systems [Moll82, Berth83, Holl87, Moll89a, Mars90]. Graphical PNs are elegant in modelling simple protocols such as the alternating bit protocol [Bart69]. However, PNs often lead to very complex network models for more realistic protocols such as the sliding window protocol.
[Tane88] and the CCITT n°7 protocol [Diaz82]. For instance, the PN for the alternating bit protocol has 14 places and 17 transitions [Moll82], and the PN for the CCITT n°7 protocol has 21 places and 22 transitions [Diaz82]. Much research effort has been put into how to reduce the PNs before analysis. Some researchers have proposed to transform PNs into FSMs and then do analysis using FSM techniques [Moll82].

Algebraic Models in communication systems originated from Hoare’s Communicating Sequential Processes (CSP) [Hoar78, Hoar85] and Milner’s Calculus of Communicating Systems (CCS) [Miln80, Miln89]. In this approach, a computing process is represented using an algebra. LOTOS [ISO8807] is an example of an algebraic model.

To model and evaluate performance, timing and other information should be integrated into the algebra. For example, time attributes and probability attributes are defined as mappings in algebra [Nou85]. [Moller] introduced a kind of timed CCS, and other performance models using algebra with extensions can be found in [Rico91, Migu93, Mars94].

Like PNs, in current research, most timed algebraic models are transformed to other forms (mostly FSMs) before performance analysis. The reason is not that AMs provide too much detail; rather, they are too abstract. Some of the useful information for performance analysis is omitted in an AM.

Temporal Logic can also be used for performance modelling. It enhances traditional logic with temporal formulas which are able to specify time properties about a system. Three different concepts, program, property and satisfaction, are
replaced by the single logical concept, *formula*. Details about a system, such as procedures and operations, are omitted.

TL was first introduced to model and validate concurrent programs [Pnue77, Lamp91, MaPn92]. TL is a useful and intuitive tool constructing prototypes of system behaviour related to time [Hale90]. It is easy to use TL to prove some logical properties about a system. For instance, TL can be used to check if certain performance or real-time requirements are satisfied in a system being modeled. However, it is difficult to predict or compute performance measures using TL. For this reason, little progress has been made in performance evaluation using TL, although there exists some work using TL to do performance validation and verification [Pnue77, Lamp91, MaPn92].

In summary, different models focus on different aspects of a system to be modeled. QM and FSM are the two most popular approaches for performance modelling. Each of them provides reasonable abstractions about the underlying computer systems. This usually results in acceptable complexities in analysis. In some of the previous work, other models were transformed into one of these two types of model before the performance analysis was carried out.

In the following, we present a more detailed comparison between these two types of model in describing communication protocols. This shall help in understanding the new performance model defined in the next chapter.
2.2.2 Queueing models vs FSMs

A QM consists of queues and servers. Both queues and servers represent the system resources such as buffers and processors. An example of a QM is given in Figure 2.1. This figure shows a queueing network consisting of three service centres. Each service centre is a queueing system.

![Figure 2.1: An example of a queueing model](image)

The major weaknesses of the QM approach in modelling communication protocols are:

1. Many important factors have to be omitted or simplified in a queueing model. For example, message arrival patterns, message lengths and error rates are usually described in probabilistic distributions, such as exponential distributions. However, in communication protocols, message arrivals cannot always be assumed to be exponentially distributed, and arrivals are not independent. For instance, a connection confirm (CC) message will usually arrive as a response after a connection request (CR) message is sent.
Obviously, the conventional way used in QMs does not provide an adequate means to model the ordering relationship among message arrivals.

2. Most protocols are specified as FSMs (or EFSMs). As a result, the services for a protocol are not identically distributed. The services are state-dependent. For instance, the order of transition execution in an FSM is not random. This correlation is specified in the FSM. However, it is difficult for a QM to model this information.

3. The queueing delay of a message in a protocol not only results from the other messages in the same queue ahead of it, it has to wait until the system is in the right protocol state. QMs alone do not consider this aspect.

4. In a QM, a job or an entire job class is usually modeled as a single unit. The coarse granularity makes the QM unsuitable for locating performance bottlenecks.

A communication FSM consists of states and transitions. A transition usually represents the process of the system receiving an incoming message, executing some actions and then generating an outgoing message. It is straightforward to specify the service ordering relationship of the transitions in an FSM (or EFSM). An example of an FSM is illustrated in Figure 2.2. In this example, there are four states and seven transitions. Each transition is labeled with its incoming and outgoing messages. The graph clearly shows the state-dependent service of the protocol. For example, when the protocol is in state 4, the only incoming message that can be served is I7 which is associated with the transition from state 4 to state 1.
Nevertheless, FSMs alone are not perfect performance models for protocols. The problems of FSMs are:

1. It is difficult to specify information about resources such as memory and CPU time, and to describe contention for shared resources.

2. An FSM itself has no queue in its definition. So it is difficult for an FSM to describe the queueing behaviour of incoming messages of a protocol.

Note that the disadvantages of FSMs in performance evaluation of protocols are the advantages of QMs, and vice versa.

In summary, although QMs and FSMs are not by themselves ideal performance models for communication protocols, QMs and FSMs offer complementary functions. An FSM can be used to model the logic behaviour of a system which is static, while a queueing model can be used to model the service behaviour which is dynamic. A combination of both can provide a good model for performance analysis of protocols. The performance extended finite state machine (PEFSM)
which is described in the next chapter follows this approach.

2.2.3 Specifying performance elements

With the exception of QMs, all the above described models were initially developed to describe logical properties of a system. These models are not intended for performance analysis. Additional features have to be added so that essential performance aspects such as time and probability can be integrated into the models. Examples include timed FSMs, timed Petri Nets, and time LOTOS. In this subsection, a brief survey is given on the integration of time and probability to formal models.

Time and probability are the two major performance attributes used in current performance studies based on FDT specifications. Timing information is critical for performance. Almost all performance measures are related to time, such as throughput, turnaround time, service delay time, queue wait time, response time and interarrival time.

Different kinds of time constraints have been used in FDTs. Examples include:

- statement execution time [DeBu87, Qi88, Shaw89, FVV91, BaBu91]
- delay time [DeBu87, BoVa88, Qi88, DeBu89, BeDi91, Shaw89, Groo91]
- timeout [Diaz82-1, BoVa88, DeBu89, Shaw89, QAF89]
- response time [Qi88]
- resource holding time [BoVa88]
Time specifications take different forms in different modelling approaches. Generally speaking, there are at least three ways to specify time in a formal model or a formal specification:

- timed statement [LiLi88, QAF89]
- assertion with time [Shaw89]
- data type with time [BoWi80]

In addition to time, probability also plays an important role in performance computation. Many unpredictable factors affect communications performance such as job arrival patterns, service times and transmission errors.

Briefly, there are two kinds of uncertainties in communication systems:

- external uncertainties, and
- internal uncertainties.

External uncertainties refer to those that are not controllable by the system such as job interarrival time and the class of the next arriving job. These external factors are usually modeled with probabilistic distributions or probabilities. Internal uncertainties are those that arise from inside a system because of non-deterministic choices. Probabilities are usually assigned to each possible choice.

The traditional QMs do not provide any mechanism to model internal uncertainties. Most of the previous work using the models introduced earlier do not distinguish these two uncertainties. For instance, in [KrWh93], only the transition probabilities of an FSM are considered. These transition probabilities are actually a combination of both uncertainties.
2.3 Analytic Methods

In this section, we provide a brief survey of the major analytic techniques used for system performance.

For each modelling approach discussed in the previous section, at least one methodology has been developed to estimate system performance. The most common methods are listed below:

- Reachability Analysis (RA) [Krit86, LiLi88, BaBu91, Jain91, BeDi91]
- Markov Process (MP) [Klein75, GuRu89]
- Queueing Theory (QT) [Klein75, SaCh81, Allen90]
- Algebraic Theory (AT) [Nou85]
- Temporal Logic and Proof Theory (TL) [Lamp91]

Each method is usually applicable to only a subset of the modelling approaches. Table 2.3 shows whether there is a known way of applying a particular methodology to a given model.

Similar to modelling methods, many analytic methods have to be enhanced with time and probability so that realistic estimates of performance measures can be computed. For example, reachability analysis is commonly used in validation and verification of FSM or PN models of communication systems [Choi85, LiLi88]. This analysis technique is based on a (state) reachability graph in which nodes stand for states and edges represent transitions. Theoretically, it is possible to derive all the reachable states and thus analyze the properties of
Table 2.1: The relationship of Modeling and Analysis approaches

<table>
<thead>
<tr>
<th></th>
<th>RA</th>
<th>MP</th>
<th>QT</th>
<th>TL</th>
<th>AT</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFSM</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PN</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>QM</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TL</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Notes: 'Y' means at least one paper has been published applying this analytic method to the model. '-' means to our knowledge, there has been no work which has applied this analytic method to the specific model.

A system. It has been proven effective in analyzing small to medium size systems. In order to use a reachability graph to reason about performance, time and probability attributes have to be added to the graph. This results in a time reachability graph [LiLi88].

The main problem of reachability analysis is state space explosion. The performance metrics which can be obtained are also very limited, mainly only throughputs and utilizations [LiLi88, BaBu91]. To our knowledge, no work using this approach has derived queueing statistics.

The theory of Markov Processes is another widely used method in performance evaluation based on FSMs. By adding time and probability to each transition (or state) of an FSM, it is straightforward to view the dynamics of the system modeled as a kind of stochastic processes. The stochastic processes in this approach are usually assumed to be discrete-time Markov chains [GrDu87, KrWh93].

Markov process analysis can also be applied to the algebraic model which is
not an FSM. Direct analyses of algebraic models for performance are difficult. Therefore, algebraic models are sometimes transformed into Markov processes before analysis. One example is reported in [Rico91] where a LOTOS specification is transformed into a Markov process. A LOTOS specification is described in a CCS-like communication algebra.

**Queueing Theory** is a theory used to analyze queueing models. Queueing statistics are important performance metrics for many systems. The queueing statistics are obtained by studying the queueing process of the system. Queueing processes are mostly stochastic processes which are often more complex than Markov processes [Allen90]. In queueing theory, Little’s Law is almost always employed.

Queueing theory provides a direct analytic technique for QMs. Performance metrics are obtained via analyzing queueing processes. These measures include queueing delay, queue length, server utilization, system residence time and throughput. Various approximation methods have been proposed to simplify the computation.

**Algebraic Theory** is a branch of mathematics studying the use of variables and operations relating the variables [West82]. The algebraic theory for communication is based on Hoare’s Communicating Sequential Processes (CSP) [Hoar78, Hoar85] and Milner’s Calculus of Communicating Systems (CCS) [Miln80, Miln89].

**Temporal Logic** itself is not only a modelling tool but also an analytic tool. It inherits most of its inference rules from conventional logic which is the science of proper reasoning. The properties of a computing system are expressed as logical formulas, which can be reasoned or proved by using inference rules within a logic
system and can be used as invariants.

2.4 Simulation Tools

Given a performance model, performance evaluation can be achieved either by analysis or simulation. This section presents some previous work on simulation tools.

Simulators used for performance purposes may be classified into two categories: QM-based simulators and FDT-based simulators. The most traditional performance simulators are based on QMs. Recently, however, many FDT-based simulators have been developed to analyze performance of computer networks and communication protocols.

2.4.1 Simulators based on FDTs

A survey of FDT-based tools for protocol development is given in [Chan93a]. The paper provides a detailed classification of most of the existing tools based on FDTs. As one can see, many simulation tools are not specifically designed for performance analysis. However, some of them can be used for performance evaluation with or without extensions.

This subsection describes the simulators based on FDTs which can be adopted for performance evaluation.

ANALYST is one of the earlier published tools which can be used to evaluate the performance of protocols [Barg86]. This system is the analysis engine of Columbia's Unified Protocol Integrated Development environment (CUPID). ANALYST takes a CCS-like model of protocols and unifies both performance
and functional analyses. Two aspects of protocol performance behaviour are addressed: meeting timing requirements and computing performance measures.

**HIT** is a performance modelling environment [Bei88]. [Heck91] presents a way to map an SDL\(^2\) specification to a model which HIT can accept using the HIT-SLANG language. In HIT-SLANG, the mapping of SDL components to computer resources and implementation-dependent parameters is given. Both the SDL specification and the HIT-SLANG code are needed as inputs to the tool.

**PEW** (Protocol Engineering Workbench) is an integrated tool for the design, testing and performance analysis of protocols specified in the Estelle FDT [KrWh89, WhKr91]. The facilities provided in PEW are mainly for performance analysis. The FDT model in PEW is EFSM. The input is an Estelle specification extended with times and probabilities to represent transitions.

**SDT** is a design tool for SDL. More precisely, it is a Computer Aided Software Engineering (CASE) tool used for real-time system development [Atle89]. SDT consists of several modules such as a graphical editor, an analyzer and converter, a simulator and a report generator. SDT is not designed specifically for performance evaluation purposes. However, some performance measures can be obtained by analyzing the simulation results.

**TOPNET** is a software package for simulating communication networks [Mars90]. It is based on a class of timed Petri Nets called *PPROT Nets* [BrMa86]. TOPNET allows graphical descriptions of the models, provides visualization of the dynamics of the simulation at run time, and presents the results graphically.

\(^2\)SDL is a formal description technique standardized by CCITT.
TOP/PDT is a toolkit for the development of communication protocols. It is based on PDL, a specification language which is derived from SDL with additional constructs. A set of functions is provided with the language PDL to compute performance parameters. Below is one of the performance computation statements provided by PDL:

\[
\text{MEASURE quantity [LOCALLY|GENERALLY]} \\
\text{PRINT [MEAN] [VARIANCE] [DEVIAITION]} \\
\text{WHEN condition}
\]

One can define a performance metric in \textit{quantity} and specify the \textit{condition} under which the data of the metric is collected and analyzed. This statement is added to a specification at the point where one wants to know a performance measure.

Io is an Estelle simulator for performance evaluation [FVV91]. It is based on EFSMs and allows the user to define transition execution and system evaluation times. The user can also control execution steps. The measurable performance metrics are the CPU load and the average response time. The computable performance metrics are transmission delays, throughputs, processor loads, protocol efficiency and memory utilization.

TSDL-Tool is an SDL based simulation tool for performance evaluation and validation [BaBu91]. It takes as input a textual description in an extension of SDL called \textit{timed TSDL} and a control file which contains initial values of variables and other parameters.

EDT is a set of tools that help in the development of communication protocols [Budk92]. The tools include a compiler, a simulator and a debugger. EDT accepts as input an Estelle specification which is essentially an EFSM. The prior version of
EDT called EDB [Chari89b] could not do performance analysis. EDT is extended to allow an Estelle specification with time parameters and thus some performance metrics can be obtained.

**LOTOS-SIM** is a LOTOS based performance evaluation tool [Migu93]. The tool takes as input a specification in a variant of LOTOS called LOTOS-TP, converts the specification into a LOTOS description and then compiles the LOTOS description into C for execution.

### 2.4.2 Simulators based on queueing models

Simulators based on QMs are used mainly to study queueing processes. The performance metrics obtained using these simulators are typically queue wait time, queue length, server utilization, system residence time and throughput rate.

Queueing theory has been extensively studied and a number of simulators have been developed in the past several decades. Some of the most commonly used tools are introduced below.

Basically, simulators based on QMs can be classified into two categories: *language embedded simulators* and *language independent simulators*. Language embedded simulators are those which are integrated into a particular simulation language. Simulation languages are similar to programming languages. A description of the system to be simulated is specified in the simulation language. This specification is then compiled into an executable simulator.

**GPSS** is an example of a language embedded simulator. It is one of the most widely used simulators for discrete event systems. GPSS comes with a language called GPSS V originally available on IBM mainframes. Later it evolved into
GPSS/H which is a superset of GPSS V. GPSS/H is now available for a wide range of computers [Schr74, Schr90]. Users describe queueing systems or queueing networks in GPSS/H which is then compiled into an executable form. Users can have flexible control over the simulations by using control statements in the GPSS program.

The Qsim tools also belong to the category of language embedded simulators. Qsim is used for creating and simulating queueing system models [Kimb89]. The models to be simulated are written in the programming language C. Some special purpose C functions, data structures, and macros such as DEFINE_QUEUE and create() are provided with Qsim. Qsim also provides many routines for computing special mathematical functions.

Other language embedded simulators are SIMSCRIPT [DiMa64], GASP [PrKi69] and DYNAMO [Pugh63]. [Nayl66] provides a chapter which surveys the simulation languages.

A language independent simulator takes as input the description of a QM to be simulated and the description of the workload model. The workload model typically describes the arrival patterns and service time distributions. The two descriptions are usually very simple and written in a specific format. This format is typically not a programming language which has variable definitions. The simulator itself is fixed and the users do not have control over it. Typical among those are SimPack [Fish94], RESQ [SaMa85, KuGo86], QNAP2 [VePo85], and Q+ (or PAW) [MeMo85, Mela86].

SimPack is a set of tools that support experiments in computer simulation with a variety of model types. This simulation package is a public domain software
and is still under development. A graphical interface called *Xsimcode* comes with it and provides a user-friendly way to construct the model of the system to be simulated. The multimodel functionality of SimPact that allows users to specify multiple models in one system is still under development.

One generally needs some knowledge of mathematical statistics, probability theory and differential equations in order to use the simulation tools that were developed decades ago. Tools such as MAP [Lazo84], RESQ [SaMa85, KuGo86] and QNAP2 [VePo85] are two of the most referenced tools which try to liberate the user from the requirement of being an expert in stochastic modelling and performance analysis. These tools adopt queueing networks together with some extensions as the underlying specification technique, and employ analytical techniques as well as simulation.

*Language embedded simulators* are more flexible than *language independent simulators*. A *language embedded simulator* provides more functionality which allows users to define controls on the simulator which can be applied throughout the simulation procedure. A *language independent simulator* does not have such flexibilities, and the performance metrics which can be obtained from the simulators of this kind are usually predefined.

However, a *language embedded simulator* may introduce inaccuracies in the simulation results. This is because the simulator allows a user to insert code which takes up some computational time. The user of the simulator normally is not able to compensate for this effect.
2.4.3 Simulation tools based on multiple models

The simulation tools described above are based on one model only. For example, SimPack can model a queueing network which may have multiple nodes in the network. However, all the nodes in the network are QMs. This limits the power in modelling computer networks. In a computer network, the nodes and the links are best modeled as a QM. However, the protocol entity in each node controls how the system operates. These control mechanisms are usually described as FSMs. So two types of models - FSMs and QMs - are needed in order to have a more accurate description of the system.

Recently, some simulators which are based on multiple models have been developed. Some examples are presented below.

**SMURPH (a System for Modeling Unslotted Real-time PHenomena)** is a package for simulating communication protocols at the medium access control (MAC) level. A simulation program in SMURPH is a combination of a protocol specification language based on C++ and an event-driven, discrete-time simulator that provides a virtual environment for protocol execution. SMURPH can be used in designing low-level communication protocols and investigating their quantitative and qualitative properties.

**Ptolemy** is a system-level design framework that allows a combination of multiple models in a computation [PTOLEMY]. Ptolemy covers practically all aspects of designing signal processing and communication systems, ranging from the design of algorithms and communication strategies, to simulation, hardware and software design, parallel computing, and real-time prototyping. This tool is a powerful and versatile system and is about 80 MB in size. The tool can
Chapter 2: survey of related work

even model VLSI circuits and power networks. A significant part of the Ptolemy code has to do with interaction between diverse models of computation. The formal models which can be handled in Ptolemy include the synchronous data flow models, the Boolean data flow models, and stochastic processes.

Another huge and powerful tool kit for network performance modelling, analysis and simulation is known as OPNET [OPNET]. OPNET is similar to TOPNET with respect to the hierarchical modelling ability. OPNET can integrate and simulate a system comprising of QNs and FSMs at different levels which are called domains. The tool allows three domains: process, node and network, and is more sophisticated than most conventional tools. For instance, in traditional queueing networks, each node is usually modeled as a single-server queueing system without further detailed description. However, in OPNET, one can easily define control schemes such as communication FSMs for flow control, rate control and congestion control as processes in a node. Furthermore, in each FSM, one can model in detail transition times and transition firing rates. Queues can be defined in at least two levels: process and node domains. This allows more realistic modelling of computer networks.

The construction of network models and FSMs can be easily done in a provided graphical environment similar to XFig or MacDraw. The OPNET package has libraries for some networks and protocols like X.25 and TCP/IP. OPNET is quite flexible and the queueing network models are not necessarily restricted to computer networks.
2.5 Summary

We have outlined some prior work on performance evaluation using formal models such as QMs and FSMs. Each performance model has its advantages and disadvantages in modelling communication protocols. Among the models introduced, QM and FSM complement each other in that the strengths of one model are the weaknesses of the other and vice versa. Therefore, a hybrid model combining both QM and FSM will be a good performance model for protocols.

In this chapter, both analytical methods and simulation tools based on formal models have been surveyed. As can be seen, more and more simulators are capable of supporting more than one type of model. A typical example is the newly released OPNET. Comparatively, much less work using multiple models has been done in analytic approaches, although [KrWh93] made an attempt along this direction.

Simulation approaches have many advantages over analytic approaches. For example, simulation is able to model the system more realistically. However, it is difficult for a simulator to obtain performance relations such as server saturation condition. A simulation result is numeric and is for a specific set of system and workload parameters. Although analytic approaches may be less accurate due to inevitable simplifying assumptions, the analytic results are very useful because they provide more general answers to the behaviours of the analyzed systems. This thesis investigates an analytic approach using a hybrid model to achieve better accuracy in modelling without introducing more complexity in computation than the traditional approaches.
A new performance model called Performance Extended Finite State Machine (PEFSM) is presented in this chapter. A PEFSM is defined as a finite state machine (FSM) extended with two performance elements – time and probability – on each of the transitions in the FSM. The stochastic process of state changes in a PEFSM is defined. In addition, three other embedded stochastic processes of the PEFSM are defined and their relationships are examined. Moreover, according to the arrival characteristics of incoming messages, PEFSMs are classified into three categories: synchronous PEFSMs (SyPEFSMs), asynchronous PEFSMs (AsPEFSMs) and hybrid PEFSMs (HyPEFSMs).

3.1 Introduction

As discussed in Chapter 2, the finite state machine (FSM) is a formal description technique (FDT) having many advantages in performance modelling of communication protocols. An FSM consists of a set of states, a set of input symbols, a
set of output symbols, and a set of state change rules called transitions. All these sets are finite.

FSMs can be used to model various computer hardware and software systems ranging from compilers to communication protocols. Our focus is on communication protocols. For the rest of the thesis, one should keep in mind that the underlying systems to be modelled by FSMs are communication protocols except when otherwise indicated.

A communication protocol embeds a set of control mechanisms such as flow control, rate control and congestion control. Each control mechanism is an FSM or an FSM extended with variables. Figure 3.1 shows graphically the FSM of the transmitter of a simple communication protocol. This protocol is known as the alternating bit protocol\(^1\) in [Tane88](p.229) and is one of the most referenced protocols in the literature. The protocol consists of one transmitter and one receiver. Only the FSM of the transmitter is shown in Figure 3.1.

Figure 3.2 depicts the FSM of another popular but more complex protocol called CCITT/ISO X.25 LAPB [ISO7776]. In this example, the FSM models both the transmitter and the receiver.

A protocol entity may consist of more than one FSM. These FSMs can be combined as one FSM. We assume only one FSM in a protocol entity hereafter.

When the FSM of a protocol starts running and communicating with others, it typically receives input symbols one after another. When an FSM has finished processing one symbol and generating the corresponding output symbol if there is one, the FSM changes its state. The next state can be the same as the current one.

\(^1\)The alternating bit protocol is also called the *stop and wait* protocol.
Chapter 3: a new performance model

This next state of the FSM usually depends on the current state, the current input symbol, and the selection rule(s) when more than one input symbol is available. The current input symbol is generally unpredictable and thus the next state is random. As a result, the process of state change is actually a stochastic process.

Properly modelling and analyzing the state changing process of the FSM of a protocol is important. It provides a means to obtain the performance of the FSM.

Treating an FSM of a communication protocol as a stochastic process for performance analysis is not new. For example, [KrWh93] transformed an FSM into a transition relation graph and then analyzed the graph as a discrete-time semi-Markov process. The transition relation graph of an FSM is in fact also an FSM except that the states and the transitions in the original FSM become the transitions and the states in the graph, respectively. More discussion on this paradigm can be found in Chapter 2.
Chapter 3: a new performance model

Although most of the previous work on performance modelling is based on FSMs, there exist two problems in general. One problem is that they do not model the arrival processes of input symbols. Without considering any arrival process, the previous work assumes that input symbols to an FSM are always available at any time and idle periods due to arrival delays do not exist. Therefore, the performance measures obtained from the existing research are mostly bounds.

Another common problem is that no queueing process due to service delays is considered. Queueing delays are important performance indices for many com-
puter systems. Without considering queueing process, queue wait times of input symbols due to processing delays of other symbols cannot be computed. In asynchronous systems, a request may arrive at any time and may have to wait its turn to receive service.

Both these two processes — the arrival process and the queueing process — are in fact very important in providing accurate modelling for a realistic FSM and obtaining performance measures for the FSM.

To incorporate these two important performance aspects in the FSM formalism, we propose a new performance model called Performance Extended FSM (PEFSM) in this chapter. A PEFSM is an FSM extended with time and probability on its transitions. The time of a transition is further refined and divided into two parts: the time to wait for the input symbol associated with the transition and the time to process the symbol and the transition. Defined in this way, a PEFSM models the stochastic process of state change in an FSM accurately.

The formal definitions of PEFSM and its related models are given in this chapter. A brief mathematical background which introduces stochastic processes and the related theory is provided in Appendix A. It is intended to help the reader understand the PEFSMs and the analyses in the later chapters.

The rest of this chapter is organized as follows. It starts with a formal definition of an FSM in Section 3.2. Readers who are already familiar with FSMs can skip this section. Section 3.3 presents the definitions of the new performance model — PEFSM. Section 3.4 outlines a classification of PEFSMs according to the arrival characteristics of the input symbols. Section 3.5 discusses three embedded processes of a PEFSM and their relationships. Section 3.6 wraps up the
3.2 Finite State Machine (FSM)

A finite state machine (FSM) is formally defined as a six-tuple

\[ M = (Q, I, O, \delta, \xi, q_0) \]  \hspace{1cm} (3.1)

where

- \( Q \) — a finite set denoting states;
- \( I \) — a finite set denoting input symbols;
- \( O \) — a finite set denoting output symbols;
- \( \delta \) — a function denoting transitions;
- \( \xi \) — a function denoting transition outputs;
- \( q_0 \) — an initial state.

\( \delta \) is a function from \( Q \times I \) to \( Q \). \( \xi \) is a function from \( Q \times I \) to \( O \). And \( q_0 \in Q \).

The FSMs defined above are essentially Mealy machines [HoU79].

A directed graph can be used to present an FSM graphically. Figures 3.1 and 3.2 are examples of FSMs. For the sake of simplicity, an abstract FSM in Figure 3.3 is used to illustrate the contents of each component of an FSM:

\[ Q = \{1, 2, 3\}; \]
\[ I = \{I_{11}, I_{12}, I_{21}, I_{23}, I_{31}\}; \]
\[ O = \{O_{11}, O_{12}, O_{21}, O_{23}, O_{31}\}; \]
\[ \delta : \delta(1, I_{11}) = 1, \delta(1, I_{12}) = 2, \delta(2, I_{21}) = 1, \]
\[ \delta(2, I_{23}) = 3, \delta(3, I_{31}) = 1; \]
\[ \xi : \xi = (1, I_{11}) = O_{11}, \xi(1, I_{12}) = O_{12}, \xi(2, I_{21}) = O_{21}, \]
\[ \xi(2, I_{23}) = O_{23}, \xi(3, I_{31}) = O_{31}; \]
Chapter 3: a new performance model

\[ \xi(2, I_{23}) = O_{23}, \xi(3, I_{31}) = O_{31}; \]

\[ q_0 = 1. \]

Figure 3.3: An abstract FSM

Note that an FSM of a communication protocol does not necessarily have any final state. This is because the execution of an FSM can span over an infinite period of time without termination.

3.3 PEFSM: a new performance model

Before giving the formal definition of PEFSM, we start with the definitions of the related terms.

3.3.1 Classification of messages

In the domain of communications, the data which a protocol receives and the data which the protocol outputs are often called *messages*. We shall use *I* of an FSM to refer to the set of incoming message classes to the FSM and *O* to the set of outgoing message classes generated from the FSM.
From the definition of FSM in Equation (3.1), each incoming message class of an FSM is associated with a transition. We assume that FSMs in this thesis are deterministic. For a deterministic FSM, the mapping between the incoming message classes of an FSM and the transitions of the FSM is one-to-one, so is the mapping between the outgoing message classes and the transitions of an FSM.

Therefore, each class of incoming messages of an FSM is uniquely identified with the identifier of the transition associated with the messages. A transition identifier of an FSM usually consists of the transition’s starting state index and the ending state index. For instance, transition $ij$ identifies the transition from state $i$ to state $j$. Hence, incoming message class $ij$ (or class $ij$ where no ambiguity occurs) refers to the incoming message class associated with transition $ij$, i.e.,

$$I_{ij} = \{ x \mid \delta(i, x) = j \}$$

where $I_{ij} \in I$ and $x$ represents an incoming message; outgoing message class $ij$ (or class $ij$ where no ambiguity occurs) refers to the outgoing message class associated with transition $ij$, i.e.,

$$O_{ij} = \{ y \mid x \in I_{ij} \land \xi(i, x) = y \}$$

where $O_{ij} \in O$ and $y$ represents an outgoing message.

An incoming message of class $ij$ of an FSM is called a firable message of state $i$ in the FSM, or a firable message when the FSM is in state $i$.

### 3.3.2 Transition time

When a firable message is selected by an FSM, the transition associated with this message will be processed. In other words, the FSM is making a state transition.
Each transition is associated with a transition time which is the time period from the start to the completion of the transition. In general, a transition time from state $i$ to $j$ consists of two parts: the transition wait time and the transition service time. The transition wait time is the time to wait for a firable message of class $ij$ and the transition service time is the time to service transition $ij$.

Let $\psi_{ij}$, $w_{ij}$ and $s_{ij}$ denote the transition time, the transition wait time and the transition service time of transition $ij$, respectively. Formally, $\psi_{ij}$ is written as

$$\psi_{ij} = w_{ij} + s_{ij}.$$ 

To define $\psi_{ij}$, $w_{ij}$ and $s_{ij}$ precisely, we first define the time instant of an FSM leaving state $i$ and entering state $j$.

**Definition 3.1 (leaving and entering a state)** The time instant of an FSM leaving state $i$ and entering state $j$ is the epoch when the service of transition $ij$ is completed.

Definition 3.1 precisely defines when a transition of an FSM begins and ends.

**Definition 3.2 (transition time)** The transition time from state $i$ to $j$ in an FSM, $\psi_{ij}$, is the interval between the time when the FSM enters state $i$ and the time when the FSM enters state $j$.

**Definition 3.3 (transition wait time)** The transition wait time from state $i$ to $j$ in an FSM, $w_{ij}$, is the interval between the time when the FSM enters state $i$ and the time when the FSM receives an incoming message associated with transition $ij$ and begins to process the message.

**Definition 3.4 (transition service time)** The transition service time from state
i to j of an FSM, $\tau_{ij}$, is the time to execute the computer instructions specified in transition $ij$.

Note that $\tau_{ij}$ is neither negative nor constant. It depends on the workload. The transition wait time is not the time due to queueing either. Any wait time due to queueing will be called queue wait time defined later in the thesis.

$\tau_{ij}$ is also the service time of an incoming message of class $ij$. So $\tau_{ij}$ includes some necessary processing time for a message such as checksum computation time.

Figure 3.4 shows graphically the components of a transition time.

Figure 3.4: The components of a transition time

According to the above definitions, it is not difficult to see that $\varphi_{ij}$ is zero when the FSM enters state $i$ and a firable message of class $ij$ is immediately executed. In this case, $\varphi_{ij} = \tau_{ij}$.

Often, the transition time in an FSM is longer than the associated service time, i.e., $\varphi_{ij} > \tau_{ij}$. This is because firable messages of a state of an FSM may not always be immediately available when the FSM enters that state. Waiting for a firable message is inevitable when an FSM enters a state but finds no firable message in this state.
3.3.3 State occupancy and stochastic process

Definition 3.1 precisely defines when an FSM enters and leaves a state. With this, we are now able to define the state holding time and the state occupancy of an FSM, and specify the trajectory of the occupied states.

Definition 3.5 (state holding time) The holding time of state $i$ in an FSM ($i \in Q$) is the interval between the time when the FSM enters state $i$ and the time when the service of a transition starting from state $i$ is completed.

Definition 3.6 (state occupancy) An occupancy of state $i$ by an FSM is an event covering the period from the time when the FSM enters state $i$ to the time when the service of the first transition starting from state $i$ is completed. In other words, it describes the state the FSM is in, and the holding time of that state.

According to the definition of FSM in Equation (3.1), a self-loop transition from state $i$ to $i$ ($i \in Q$) may occur in an FSM. If an FSM has a self-loop transition from state $i$ to $i$ and the transition is selected, then the FSM is considered to enter state $i$ again (i.e. one more time) when the service of this transition is completed.

For a protocol in execution, its FSM goes through a sequence of state changes. This process is known as the state changing process of the FSM.

The state changing process of an FSM cannot be fully predetermined. This is because a transition wait time $w_{ij}$ and a transition service time $\tau_{ij}$ ($i, j \in Q$) are not constant in actual practice. In general, $w_{ij}$ and $\tau_{ij}$ can be regarded as random variables. This means the holding time of the current state of an FSM is also a random variable. Moreover, before a transition of an FSM is selected, the next state of the FSM is also unknown. The selection of the next transition
of an FSM depends on the current state of the FSM, the set of available firable messages of the current state, and the transition selection policy which deals with the case when more than one firable message of the current state is available. In particular, whether or not a firable message in the current state is available depends on the arrival model of the FSM which is usually a random process.

All these uncertainties indicate that there are several random factors which collectively determine which next state an FSM is in and when. Consequently, the states occupied by an FSM over the time are a family of values of a random variable related to time. Therefore, the state changing process of an FSM is a stochastic process. We denote this stochastic process as \( \{X(t) \mid t \geq 0\} \), where \( X(t) \in Q \) and \( Q \) is the set of the states of the FSM. \( X(t) \) represents the state that FSM occupies at time \( t \). \( X \) is called the state variable of the FSM and \( Q \) is the state space of \( X \). Obviously, the values of \( X \) are discrete and finite.

To characterize the possibility of the next state which an FSM enters, we introduce the single-step transition probability.

**Definition 3.7 (single-step transition probability)** The single-step transition probability \( p_{ij} \) of an FSM is the probability that state \( j \) will be the next state when the FSM enters state \( i \), i.e.,

\[
p_{ij} = Pr\{X_{n+1} = j \mid X_n = i\}
\]

where \( X_n \) and \( X_{n+1} \) denote the current state and the next state, respectively.

To be general, we assume that the times, \( \tau_{ij} \) and \( \tau^*_{ij} \) \((i, j \in Q)\) of an FSM are continuous. As such, \( \tau_{ij} \) \((i, j \in Q)\) are also continuous.
The state changing stochastic process of an FSM, \( \{X(t)|t \geq 0\} \), is a continuous-time semi-Markov process (CSMP)\(^2\) when the next state of the FSM does not depend on the states occupied by the FSM other than the current state. In other words, if all \( p_{ij} \), \( ^w \tau_{ij} \) and \( ^r \tau_{ij} \) \((i, j \in Q)\) are independent of the past history of \( X(t) \), \( X(t) \) is a CSMP. CSMP is commonly used in describing realistic systems as demonstrated in the next three chapters.

### 3.3.4 Formal definition of PEFSM

We have described in the previous section how an FSM extended with time and probability can model the stochastic process of the state changes of the FSM. Since both time and probability are needed to model performance, we call an FSM with this extension a *performance extended FSM* (PEFSM).

Formally, a PEFSM, denoted as \( \Psi \), is defined as a pair

\[
\Psi = (M, P)
\]

where \( M = (Q, I, O, \delta, \xi, q_0) \) is the kernel which is an FSM whose formal definition is given in (3.1), and \( P = (P, ^w H, ^r H) \) is the running environment expressed in terms of time and probability. The components of \( P \) are

- **P**  - a matrix of the single-step transition probabilities;
- \(^w H\)  - a matrix of the probability density functions (p.d.f.s) of the transition wait times;
- \(^r H\)  - a matrix of the p.d.f.s of the transition service times.

In a PEFSM, \( M, P, ^w H \) and \(^r H\) are primitive data. \( M \) and \(^r H\) are assumed to be provided directly by the performance evaluator. \( P \) and \(^w H\) will be derived.

\(^2\)The formal definition of CSMP is given in Appendix A.
from \( M \), \( \mathcal{H} \) and the more primitive data which are the arrival and service models when \( P \) and \( \mathcal{H} \) are not directly given.

Let \( P = [p_{ij}] \);
\[
\mathcal{H} = [\mathcal{h}_{ij}(t)];
\]
\( \mathcal{H} = [\mathcal{h}_{ij}(t)]; \)
\{ \( X(t), t \geq 0 \) \} be the state change process of the PEFSM (defined in Section 3.3.3);
\( t_0, t_1, ..., t_n, ... \) be the sequence of the epochs right before the processing of a transition is completed;
\( X_0, X_1, ..., X_n, ... \) be the sequence of the states in the PEFSM corresponding to the time instant sequence \( t_0, t_1, t_2, ..., t_n, ... \), respectively.

The components of a PEFSM and their relationships are formally defined in the following:

1. \( X(t) \in Q \) for all \( t \geq 0 \).
2. \( X(0) = X_0 = q_0 \).
3. \( X_n = X(t_n^-) \) \(^3\) and \( X_{n+1} = X(t_n) \).
4. \( Pr\{X_{n+1} = j | X_n = i\} = p_{ij} \).
5. \( \mathcal{h}_{ij}(t) \) is the p.d.f. of \( \mathcal{r}_{ij} \), i.e. \( Pr\{\mathcal{r}_{ij} = t | X_n = i, X_{n+1} = j\} = \mathcal{h}_{ij}(t) \).
6. \( \mathcal{r}_{ij}(t) \) is the p.d.f. of \( \mathcal{r}_{ij} \), i.e. \( Pr\{\mathcal{r}_{ij} = t | X_n = i, X_{n+1} = j\} = \mathcal{r}_{ij}(t) \).
7. If \( X_n = i \) and \( X_{n+1} = j \), then
\[
t_n - t_{n-1} = \mathcal{r}_{ij} = w_{r_{ij}} + \mathcal{r}_{ij} \quad (n \geq 1).
\]

\(^3\) \( t_n^- \) denotes the epoch right before the time instant \( t_n \). \( t_n^- < t_n \) but \( t_n^- \) is infinitely close to \( t_n \).
8. Let $X_{n+1} = i$. If there is no available firable message in state $i$ at $t_n$, then the PEFSM waits. If $j$ is the next state after $i$, then the waiting time is $w_{rij}$. This means that during the period $[t_n, t_n + w_{rij})$, the PEFSM is idle and during the period $[t_n + w_{rij}, t_n + w_{rij} + r_{ij})$, transition $ij$ is being processed.

One can clearly see from the above that the trajectory of the state variable $X$ of a PEFSM is governed by $M$, $P$, $^aH$ and $^H$. $M$ determines the state space of $X$ and the possible next value of $X$ statically. The others realize the dynamic control of $X$.

By the definition of PEFSM, $M$ and $^aH$ of a PEFSM are the primitive data which are independent of the state variable $X$ of the PEFSM. Therefore, if the state changing process of a PEFSM is a CSMP, $P$ and $^aH$ of the PEFSM should not depend on the past state occupancies of the PEFSM. However, $P$ and $^aH$ of a PEFSM are determined by the arrival model of incoming messages to the PEFSM as well as the service model of the PEFSM which defines the selection rules to choose a firable message to process at each state. Therefore, in order to make the state changing process a CSMP, it is necessary to restrict both the arrival and service models with additional assumptions. These assumptions are given in the later chapters where each specific class of PEFSMs is described.

The arrival model of a PEFSM specifies the arrival patterns of incoming messages. We note that the arrival patterns are closely related to the type of communication, synchronous, asynchronous or a combination of the two. In the next section, these three types of communication and their impacts on PEFSMs are discussed.
3.4 Synchronous, Asynchronous and Hybrid PEF-SMs

Synchronizations may occur during communication when the participants rendezvous with each other at one or more points. When the FSM model is used, the points of rendezvous are usually defined as states in the FSM called synchronization states. Synchronizations in communication are often realized through messages exchanges. From the PEFSM point of view, the fireable messages of a synchronization state arrive only after the PEFSM enters this state. Incoming messages which are not of this type can arrive at any time. Therefore, two types of message arrivals are possible:

**Definition 3.8 (synchronous arrival)** The arrival model of incoming messages to a PEFSM is synchronous if these messages arrive only when the PEFSM is in a synchronization state.

**Definition 3.9 (asynchronous arrival)** The arrival model of incoming messages to a PEFSM is asynchronous if these messages arrive regardless of the current state of the PEFSM.

The distinction of incoming messages to PEFSMs according to synchronous and asynchronous arrivals gives rise to the following classification of PEFSMs:

1. If all the incoming messages of a PEFSM arrive synchronously, the PEFSM is called a synchronous PEFSM (SyPEFSM).

In a SyPEFSM, the next incoming message does not arrive before all the previously arrived messages have been processed and the outgoing messages (if any) have been sent. Whenever the SyPEFSM enters a state, it must
wait for a firable message in the state. In this sense, the SyPEFSM is synchronized in each state. This also implies that no queue is needed for any incoming message of a SyPEFSM and consequently there is no queue wait time for each incoming message in a SyPEFSM.

A simple example of a SyPEFSM is the FSM of a protocol called hot-potato protocol. A formal specification in Estelle of this protocol is given in Appendix D. This protocol describes the procedure of a team of members tossing a hot potato among themselves in a circle. Each member catches the hot potato with his/her left hand from the member on his/her left side, passes it over to his/her right hand and then tosses it to the member on his/her right side. Each member can be represented using an FSM. One example of the FSMs is shown in Figure 3.5. There are two states in an FSM. State 0 stands for a member's his/her left hand waiting to catch a potato, and state 1 for the member's his/her right hand waiting to catch a potato. $E_1$ occurs only after the FSM enters state 0; $E_2$ occurs only after the FSM enters state 1. Obviously, the PEFSM of this FSM is a SyPEFSM.

Other examples of SyPEFSM include the PEFSMs of the communication protocols which assume zero buffer for incoming messages [HsWa94]. A PEFSM of this type has to wait for a firable message in every state.

2. If all the incoming messages of a PEFSM arrive asynchronously, the PEFSM is called an asynchronous PEFSM (AsPEFSM).

Obviously, an AsPEFSM allows incoming messages to arrive before all the previously arrived messages have been processed. We assume that a firable message of a state in an AsPEFSM that arrives before the system is ready
to process it is buffered in a queue. If a firable message in a state is found in the queue when the AsPEFSM enters this state, there is zero transition wait time. However, this situation does not always occur in general, especially when the workload of the communication is light. In other words, AsPEFSMs may have to wait for a firable message in the same way that SyPEFSMs do.

An example of an AsPEFSM is the two-state FSM shown in Figure 3.6(a). This FSM models a switch processing the traffic data of ON-OFF Markov sources in an ATM network. The traffic model of ON-OFF Markov sources shown in Figure 3.6(b) is a well-known data model used in studying the behavior of ATM networks [AtMa93, HsWa94]. In this example, two types of data generated at different rates are modelled: video and text. Both types of data come to an ATM switch asynchronously from different senders. Therefore, the service model of an ATM switch given in Figure 3.6(a) is an
Chapter 3: A new performance model

As PEFSM.

(a) an FSM for processing two sources of data

(b) ON-OFF Markov sources of traffic model

Figure 3.6: An FSM of an ATM serving two types of data

3. If a PEFSM has both synchronous and asynchronous incoming messages, the PEFSM is known as a hybrid PEFSM (HyPEFSM).

In reality, incoming messages to the FSM of a protocol are often a combination of synchronous and asynchronous messages. For instance, the messages
for handshaking during a connection setup in the LAPB protocol [ISO7776] and the messages for timeout in the same protocol are usually synchronous because they only arrive when the protocol is in certain states. The commands and data from the protocol user (or more generally, from the upper layer of the protocol stack) are usually asynchronous because they may arrive without foreknowledge of the current state of the protocol. The FSMs of this kind of protocols are HyPEFSMs.

The FSM shown in Figure 3.1 can be an example of a HyPEFSM. We assume that the data to be transmitted comes to this FSM asynchronously. The FSM always waits for an acknowledgement (ACK) from the receiver each time it sends out a packet of data. A timer is set after each data transmission. When a timeout occurs, the unacknowledged data is retransmitted. The timeout value is assumed to be large enough that no delayed ACK will arrive after a timeout. As a result, the timeout events come to this FSM synchronously. Therefore, both synchronous and asynchronous incoming messages are sent to this FSM, making it a HyPEFSM.

We have seen that PEFSMs are classified into three categories according to the two types of arrival patterns of incoming messages — synchronous and asynchronous. Likewise, we define two types of transitions:

Definition 3.10 (synchronous transition) A transition of a PEFSM is a synchronous transition if the message arrivals for the transition are synchronous.

Definition 3.11 (asynchronous transition) A transition of a PEFSM is an asynchronous transition if the message arrivals for the transition are asynchronous.
In a SyPEFSM, all the transitions are synchronous transitions. In an AsPEFSM, all the transitions are asynchronous transitions. In a HyPEFSM, there are both synchronous and asynchronous transitions.

We have mentioned in the definition of PEFSM that $P$ and $H$ of a PEFSM are assumed to be given directly by the performance evaluator or computed from the information provided by the performance evaluator. The former case usually applies in SyPEFSMs and the latter one in AsPEFSMs and HyPEFSMs. Different arrival models make the computations of $H$ for a PEFSM very different. This will be discussed in subsequent chapters.

### 3.5 Embedded Stochastic Processes of a PEFSM

The state changing process of a PEFSM, $\{X(t), t \geq 0\}$, embeds many stochastic processes. This section discusses three of them and investigates their relationships. These processes are used in the performance evaluation studies in the remaining chapters of the thesis.

#### 3.5.1 Pure state changing process (Model $E$)

If all the transition times in a PEFSM are ignored, i.e., only the instant values of $X(t)$ right before the completion of each transition are considered, then the stochastic process of the PEFSM, $\{X(t), t \geq 0\}$, is a Markov process (MP). This is equivalent to the assumption that the values of the random variables of a PEFSM, $w_{ij}$ and $\tau_{ij}$ ($i, j \in Q$), are all zero. With this assumption, the state changing process in fact describes only the sequence of the states entered by the PEFSM. We call this process the pure state changing process and denote it as
Model $E$.

Formally, Model $E$ of a PEFSM is defined as:

$$E = (M, P)$$  \hspace{1cm} (3.3)

where $M$ and $P$ are as defined in Equation 3.2.

### 3.5.2 Pure service delay process (Model $S$)

If the firable messages of a PEFSM are always immediately available, i.e., $w_{ij} = 0$ $(i, j \in Q)$, the state changing process of the PEFSM, $\{X(t), t \geq 0\}$, is a stochastic process, which depicts only the service delays (i.e., the transition service time) at each state of the PEFSM in addition to the state changes. We call this process the *pure service delay process* and denote it as Model $S$.

Formally, Model $S$ of a PEFSM is defined as

$$S = (M, P, H)$$  \hspace{1cm} (3.4)

where $M$, $P$ and $H$ are as defined in Equation 3.2.

### 3.5.3 Pure wait process (Model $W$)

If the transition service times of a PEFSM are always zero, i.e., $\tau_{ij} = 0$, the state changing process of the PEFSM, $\{X(t), t \geq 0\}$, forms a stochastic process which characterizes the wait times of the PEFSM for incoming messages at each state. We call this process the *pure wait process* of the PEFSM and denote it as Model $W$. 
Formally, Model $W$ of a PFSM is defined as

$$W = (M, P, ^{\omega}H)$$

(3.5)

where $M$, $P$ and $^{\omega}H$ are as defined before in Equation 3.2.

Figure 3.7 shows a graph of Model $W$ of the PEFSM given in Figure 3.3. In the model, all outputs are removed from the transitions to denote the exclusions of the transition services in Model $W$.

A transition in $W$ starts immediately after its start state is entered and ends when a firable message of the state is received. The transition times of Model $W$ of a PEFSM are simply the transition wait times of the PEFSM. During the transition wait times, the PEFSM is essentially idling. Therefore, Model $W$ depicts the idle times of the PEFSM.

### 3.5.4 Relationships

First of all, Model $E$, Model $W$, Model $S$ and $\Psi$ of a PEFSM share the same components:
• the states;
• the transitions;
• the single-step transition probabilities.

We define the steady-state state probability of state $i$ in a stochastic process is the probability that the process is seen in state $i$ at changing instants by an arbitrary observer after the process runs a very long time and is in steady state.\footnote{The steady-state state probability is also the stationary state probability.} Let $\pi_i$ be this probability. $\pi_i$ can be computed by solving the following matrix equation

$$\pi \cdot P = \pi$$

and

$$\sum_{i \in Q} \pi_i = 1$$

where $\pi = [\pi_i]$ ($i \in Q$). The proof is given in Appendix A. Since $P$ is the same for $E$, $W$, $S$ and $\Psi$, these stochastic processes always have the same steady-state state probabilities.

Let $\mathbf{H} = [\psi_{ij}(t)]$ be the matrix of the p.d.f. of the state holding times of the PEFSM. Since $\psi_{ij} = w_{ij} + \tau_{ij}$, and $w_{ij}$ and $\tau_{ij}$ are independent, therefore

$$\psi_{ij}(t) = \int_0^t \psi_{ij}(\tau) \cdot h_{ij}(t - \tau) \cdot d\tau.$$ 

In the extreme case when the PEFSM has no idle time and never waits for any firable message, then

$$\psi_{ij}(t) = h_{ij}(t)$$
and

\[ \psi_{r_{ij}} = \ast_{r_{ij}}. \]

In this case, the PEFSM behaves as in Model S. If \( \simeq \) is used to denote this behaviour equivalence, we can write in short

- \( \Psi \simeq S \) if \( w_{r_{ij}} \equiv 0 \), i.e. \( \psi_{r_{ij}} \equiv \ast_{r_{ij}} \) \((i, j \in Q)\).

Likewise, by the definitions of \( W \) and \( E \), we can also write

- \( \Psi \simeq W \) if \( \ast_{r_{ij}} \equiv 0 \), i.e. \( \psi_{r_{ij}} \equiv w_{r_{ij}} \) \((i, j \in Q)\).

- \( \Psi \simeq E \) if both \( w_{r_{ij}} \equiv 0 \) and \( \ast_{r_{ij}} \equiv 0 \) \((i, j \in Q)\).

In general, when a PEFSM arrives at a state, if at least one firable message in this state is already available, the PEFSM will immediately execute the transition associated with the message; otherwise the PEFSM waits for the first firable message. After having received a firable message and processed the corresponding transition, the PEFSM then changes its state to the end state of the transition.

Since waiting may occur, in general, it is always true that

\[ \psi_{r_{ij}} \geq \ast_{r_{ij}} \quad (i, j = 0, 1, \ldots, |Q|) \]

where \( \psi_{r_{ij}} \) and \( \ast_{r_{ij}} \) are as defined earlier in the chapter.

If \( \Psi \) is a CSMP, Model \( E \) is a Markov process (MP). \( E \) is also a Markov chain (MC). This is because its state space \( Q \), by definition, is finite and discrete. Furthermore, this Markov chain is irreducible because every state in the FSM is reachable from the idle state in a correct communication protocol, and after a sequence of transitions, the system always returns to the idle state. And this
Chapter 3: a new performance model

FSM is *positive recurrent* because the number of states in $Q$ is finite [Moll89b]. If we further assume that $E$ is *aperiodic*\(^5\), then it is *ergodic* [Allen90].

When $E$ of a PEFSM is ergodic and the running time of the PEFSM approaches infinity, $E$ will achieve *equilibrium* (or *steady-state*). In this case, *limiting state probabilities* exist which are the probabilities that the FSM is seen in each state by an arbitrary observer at any time, and they are equal to the *steady-state state probabilities*, $\pi = [\pi_i]$ ($i \in Q$).

### 3.6 Summary

The formal definitions of a new performance model – PEFSM (denoted by $\Psi$), its embedded models $E$, $W$ and $S$, and their relationships – have been presented.

Two types of message arrivals – synchronous and asynchronous – and their effects on PEFSMs have also been discussed. With this distinction, PEFSMs are classified into three categories: SyPEFSMs, AsPEFSMs and HyPEFSMs. This classification determines the performance indices of interest to each class of PEFSMs. For instance, *throughput* of a specific outgoing messages is important to all types of PEFSMs; *queueing statistics* do not make sense to SyPEFSMs at all but are very important to AsPEFSMs and HyPEFSMs. All prematurely arrived firable messages to an AsPEFSM or HyPEFSM are buffered. Therefore, any measure involving the queues is a useful performance index to an AsPEFSM or a HyPEFSM.

\(^5\)This assumption is not necessary for $\Psi$ being a CSMP if the times of the PEFSM are continuous random variables [GrHa85](p.52). Moreover, this assumption can also be eliminated for $E$ if there exists a *reset* in the communication protocol specified by the PEFSM at the protocol’s initial state. This reset forms a self-loop transition in $E$. 
The state changing stochastic processes modelled by PEFSMs in this study are very general. The distributions of transition service times are not as restricted as in previous work. Only the message arrival times will have some restrictions imposed on them to qualify the stochastic processes as CSMPs which are easy to compute. These restrictions will be introduced in the next three chapters.

One of the major differences between our models and those in earlier work is that the waiting times for incoming messages on each transition of an FSM are defined as a component in the transition time. By doing so, arrival models (or workload models) can be integrated in our models. In addition, the queues of a PEFSM can also be modelled and associated with each incoming message class. Details are discussed in the later chapters.

In short, this model makes our approach more realistic and accurate. More performance indices can be obtained and even the properties of these indices such as the probability distributions can be derived through analysis.

In the next three chapters, we show how to derive the performance metrics for synchronous, asynchronous and hybrid PEFSMs, respectively. While the hybrid model is the most useful and realistic model for communication protocols, the other two models are also presented for completeness, and as a way to explain the hybrid model.
Chapter 4

Performance Evaluation of Synchronous PEFSMs

Abstract

A synchronous PEFSM (SyPEFSM) is a PEFSM where the arrivals of all incoming messages are synchronized. This chapter analyzes statistical properties of a SyPEFSM and derives some performance measures for the SyPEFSM. While practical examples of SyPEFSMs are not common, SyPEFSMs are interesting in their own right and useful in understanding HyPEFSMs.

4.1 Introduction

A synchronous PEFSM (SyPEFSM) is a type of PEFSM where the incoming messages arrive synchronously. When a SyPEFSM enters a state, it has to wait for a firable message of this state. There is no queue wait of any incoming message.

The notations used to define PEFSM and SyPEFSM in the previous chapter
are also used in this chapter. Formally, a SyPEFSM is denoted by $\Psi$, and

$$\Psi = (M, \mathcal{P})$$

where

$$\mathcal{P} = (\mathcal{P}, \mathcal{H}, \mathcal{H}).$$

$M$ and $\mathcal{H}$ of a SyPEFSM are primitive data and are presumably given by the performance evaluator. $\mathcal{P}$ and $\mathcal{H}$ are related to the message arrival model of a SyPEFSM. A message arrival model is important to a SyPEFSM from the performance evaluation point of view, because the arrival model characterizes the traffic pattern of the incoming messages. The arrival model is in fact a workload model which directly influences the performance of a SyPEFSM. The arrival model is therefore needed in computing any performance measures.

As long as $M$, $\mathcal{H}$, $\mathcal{P}$ and $\mathcal{H}$ are known, the state changing stochastic process of the SyPEFSM can be analyzed and the performance of the SyPEFSM can be evaluated.

The performance metrics of a SyPEFSM include the state holding time, the recurrence time of a state, the throughput of the outgoing messages of a specific class and the utilization. The derivation of these performance metrics is presented in this chapter. Furthermore, the bounds of the performance measures of a SyPEFSM are also obtained by analyzing Model $S$.

The rest of this chapter is organized as follows. Section 4.2 provides a message arrival model for SyPEFSMs. Section 4.3 presents the derivations of some performance metrics with the given message arrival model. Section 4.4 discusses the related work on performance modelling and analysis by treating an FSM as a semi-Markov process. Section 4.5 summarizes the chapter.
4.2 Message Arrival Model

According to the definition of a SyPEFSM in Chapter 3, the SyPEFSM always has to wait for a firable message at every state since any message firable at a state arriving before the state is entered will be discarded. In general, neither the class of the next firable message nor the time when the next firable message will arrive is deterministic. Therefore, it is useful and natural to assume an arrival model which consists of the probabilities of each class of the firable messages of a state and the p.d.f.s of the wait times for each class of firable messages in each state. These are the transition probabilities, $P$, and the p.d.f.s of the transition waiting times of the SyPEFSM, $^{\text{w}}H$, respectively.

We define the arrival model of a SyPEFSM as $(P, ^{w}H)$. These data are directly given by the performance evaluator. We further assume $(P, ^{w}H)$ is independent of the past state occupancies of the SyPEFSM. With this assumption, the state changing process of a SyPEFSM is a CSMP. The performance of the SyPEFSM can be derived by applying the theory of CSMPs given in Appendix A. Some results are shown in this chapter.

4.3 Performance Metrics of SyPEFSMs

When all the data of $M$, $^{H}$, $P$ and $^{w}H$ of a SyPEFSM are known, the state changing stochastic process of the SyPEFSM is completely specified: the single-step transition probability matrix is $P$; the transition times, $^{w}r_{ij}$ ($i,j \in Q$), are computed as

$$^{w}r_{ij} = w_{ij} + ^{s}r_{ij}$$  \hspace{1cm} (4.1)
where \( w_{ij} \) and \( r_{ij} \) are the waiting time and the service time of transition \( ij \), respectively.\(^1\) The p.d.f.s of \( w_{ij} \) and \( r_{ij} \) are given in \( wH \) and \( RH \), respectively.

With this information about a SyPEFSM, we first derive the performance metrics from Model \( S \) of the SyPEFSM and then investigate the metrics from Model \( \Psi \).

### 4.3.1 Performance metrics computed from Model \( S \)

As discussed in the previous chapter, Model \( S \) of a SyPEFSM models the stochastic process of state change of the SyPEFSM under an extremely heavy workload so that there is no wait time for any firable message. Firable messages are always immediately ready when the SyPEFSM needs them. Models \( \Psi \) and \( S \) have the same states, transitions and even single-step transition probabilities but each transition time of \( \Psi \) is longer than that of the corresponding transition of Model \( S \). Therefore, the statistical and performance measures below obtained from Model \( S \) of a SyPEFSM can be viewed as the upper bounds of the SyPEFSM.

**Upper bound of mean overall service time**

From Chapter 3, it is seen that the steady state probabilities of Model \( E \), Model \( S \) and \( \Psi \) are the same. Let \( \pi = [\pi_i] \) be the vector of the steady state probabilities in a SyPEFSM. \( \pi \) can be computed from the single-step transition probability matrix \( P \) of the SyPEFSM. We shall assume \( \pi \) is known hereafter.

Let \( \bar{\tau}_{ij} \) be the mean service time of transition \( ij \) of a SyPEFSM, \( \bar{\tau}_i \) be the mean service time of all the transitions out of state \( i \), and \( \bar{\tau} \) be the mean service

\(^1\)The definitions of transition waiting time and transition service time are given in Chapter 3.
time of all the transitions in the SyPEFSM.

According to the definition of Model $S$ of a PEFSM, $\tau_{ij}$ is the transition time of Model $S$, $\tau_i$ is the mean transition time of all the transitions out of state $i$ in Model $S$, and $\tau$ is the mean transition time of all the transitions in Model $S$. We have

$$\tau = \sum_{i \in Q} \pi_i \cdot \tau_i = \sum_{i \in Q} \sum_{j \in Q} \pi_i \cdot p_{ij} \cdot \tau_i,$$

where $p_{ij}$ is the element at $(i, j)$ of the transition probability matrix $P$.

According to the message classification in the previous chapter, the relationship between the transitions and the incoming message classes of a PEFSM is a one-to-one mapping. Therefore, $\tau$ is also the mean overall service time of the incoming messages to the SyPEFSM.

**Upper bound of mean throughput rate**

We define a class $ij$ outgoing message is generated (if any) each time transition $ij$ is served. Let $\check{e}_i$ be the mean entrance rate of state $i$ in Model $S$ of a SyPEFSM, and $\eta_{ij}$ be the mean throughput rate of the outgoing messages of class $ij$ of the SyPEFSM.

Since class $ij$ outgoing messages are associated with transition $ij$, therefore, $\check{\eta}_{ij}$ is also the mean entrance rate of transition $ij$ in Model $S$. Applying Theorem A.4 and Equation (A.2), we can derive $\check{\eta}_{ij}$ as

$$\check{\eta}_{ij} = \check{e}_i \cdot p_{ij} = \frac{\pi_i \cdot p_{ij}}{\tau}.$$

$\check{\eta}_{ij}$ in fact is an upper bound of the mean throughput rate of class $ij$ outgoing messages in the SyPEFSM because $\check{\eta}_{ij}$ is computed without taking any
message arrival delays into account. The corresponding lower bound of the mean recurrence time of class $ij$ outgoing messages is equal to the reciprocal of $\tau_{ij}$, i.e., $\frac{s_p}{\tau_{ij}}$.

### 4.3.2 Performance metrics computed from Model $\Psi$

According to the definition of PEFSM in the previous chapter, $w_{\tau_{ij}}$ (the delay of waiting for a firable message) as well as $s_{\tau_{ij}}$ (the service time of the transition) is included as part of the transition time of transition $ij$ in a SyPEFSM.

Let $\psi_{\tau_{ij}}$ and $w_{\tau_{ij}}$ be the mean values of $\psi_{\tau_{ij}}$ and $w_{\tau_{ij}}$, respectively.

From Equation (4.1), we have

$$\psi_{\tau_{ij}} = w_{\tau_{ij}} + s_{\tau_{ij}} \quad (i, j \in Q).$$

Mean holding time at a specific state of $\Psi$

Let $\psi_{\tau_i}$ be the mean holding time at state $i$ of a SyPEFSM. It is the mean value of the time that the SyPEFSM stays in state $i$ before finishing a transition starting from state $i$.

Therefore, $\psi_{\tau_i}$ can be computed as

$$\psi_{\tau_i} = \sum_{j \in Q} p_{ij} \psi_{\tau_{ij}} = \sum_{j \in Q} p_{ij} (w_{\tau_{ij}} + s_{\tau_{ij}}) = w_{\tau_i} + s_{\tau_i}$$

where $w_{\tau_i}$ is the mean holding time at state $i$ of Model $W$ of the SyPEFSM.
Mean overall holding time at a state of $\Psi$

Let $\psi_\overline{\tau}$ be the mean overall holding time at an arbitrary state of a SyPEFSM.

Since the steady-state state probabilities of each state of the SyPEFSM are known, $\psi_\overline{\tau}$ can be computed:

$$
\psi_\overline{\tau} = \sum_{i \in Q} \pi_i \psi_i
$$

(4.6)

$$
= \sum_{i \in Q} \pi_i \sum_{j \in Q} p_{ij} (\psi_{\tau_{ij}} + s_{\tau_{ij}})
$$

(4.7)

$$
= \sum_{i \in Q} \sum_{j \in Q} \pi_i p_{ij} \psi_{\tau_{ij}} + \sum_{i \in Q} \sum_{j \in Q} \pi_i p_{ij} s_{\tau_{ij}}
$$

(4.8)

$$
= w_{\overline{\tau}} + s_{\overline{\tau}}
$$

(4.9)

where $w_{\overline{\tau}}$ is the mean overall holding times at an arbitrary state of Model $W$ of the SyPEFSM.

Limiting lodging probability of a state in $\Psi$

Let $\psi_\phi_i$ be the probability that a SyPEFSM is seen at state $i$ by an arbitrary observer when the SyPEFSM is in equilibrium. $\psi_\phi_i$ is also called the limiting lodging probability of state $i$ of the SyPEFSM.

By Theorem A.2, $\psi_\phi_i$ can be computed as

$$
\psi_\phi_i = \frac{\pi_i \cdot \psi_{\tau_i}}{\psi_\overline{\tau}},
$$

where $\psi_{\tau_i}$ and $\psi_\overline{\tau}$ are computed using Equations (4.5) and (4.9), respectively.

State recurrence time in $\Psi$

We define the recurrence time of state $i$ in a stochastic process is the interval between two consecutive entrances of state $i$. Let $\phi_{\overline{\tau}_{ii}}$, $w_{\overline{\tau}_{ii}}$, and $s_{\overline{\tau}_{ii}}$ be the mean
recurrence time of state \( i \) of Models \( S \), \( W \) and \( \Psi \) in a SyPEFSM, respectively.

By Theorem A.5, \( \psi \bar{q}_i \) can be computed as

\[
\psi \bar{q}_i = \frac{\psi_r}{\pi_i} \quad \text{(4.10)}
\]

\[
= \frac{\sum_{u=0}^{Q} \pi_u \cdot \psi_{ru}}{\pi_i} \quad \text{(4.11)}
\]

\[
= \frac{\sum_{u=0}^{Q} \pi_u \cdot \sum_{v=0}^{Q} p_{uv} \cdot \psi_{uv}}{\pi_i} \quad \text{(4.12)}
\]

\[
= \frac{\sum_{u=0}^{Q} \pi_u \cdot \sum_{v=0}^{Q} p_{uv} \cdot (w_{ru} + s_{uv})}{\pi_i} \quad \text{(4.13)}
\]

\[
= \psi \bar{q}_i + s \bar{q}_i. \quad \text{(4.15)}
\]

We know that as long as \( \pi_u, p_{uv}, w_{ru} \), and \( s_{uv} \) \((u, v \in Q)\) are known, the mean recurrence time of state \( i \) in \( \Psi \) can be computed using Equation (4.14). And from Equation (4.15), it is seen that this mean recurrence time is the sum of the mean recurrence time of state \( i \) in the waiting for arrival model plus that of in the service model. This result coincides with the intuition because \( \Psi \) is a model which counts both waiting and service.

Utilization

It is not difficult to see from Equation (4.15) that in a period of \( \psi \bar{q}_i \), the protocol specified by a SyPEFSM is busy (i.e. executing) only an amount of time equal to \( \psi \bar{q}_i \) on average.

Let \( \rho_i \) be the ratio of the busy time and the recurrence time for state \( i \). Then

\[
\rho_i = \frac{\psi \bar{q}_i}{\psi \bar{q}_i}
\]
Obviously, $\rho_i$ is the mean utilization of the SyPEFSM in the recurrence time of state $i$. From the above result, $\rho_i$ is seen to be independent of state $i$ of the SyPEFSM. Therefore, it is equal to the overall utilization of the SyPEFSM.

Let the overall utilization be denoted as $\rho$. Then

$$\rho = \rho_i = \frac{s_{i}}{\psi_{i}}.$$

Since $\rho$ is the probability that the SyPEFSM is busy, so $(1 - \rho)$ is the probability that the SyPEFSM is idle waiting for incoming messages.

**Throughput**

We define the *limiting entrance rate of transition* $ij$ in a PEFSM is the ratio of the number of transition $ij$ being served over a very long time. Let $\psi_{ij}$ be the limiting entrance rate of transition $ij$ of a SyPEFSM.

Similar to $\theta_{ij}$ in (4.2), $\psi_{ij}$ can be computed as

$$\psi_{ij} = \psi_{e_i} \cdot p_{ij} = \frac{\pi_i}{\psi_{i}} \cdot p_{ij} = \frac{\pi_i p_{ij}}{s_{i} \psi_{i} + s_{i}}.$$

Class $ij$ outgoing messages are associated with transition $ij$. Each time transition $ij$ is completed, an outgoing message of class $ij$ is output. For this reason,
\( \psi \tilde{\eta}_{ij} \) is also the mean throughput rate of class \( ij \) outgoing messages under the workload specified by \((P, \mathcal{H})\).

The mean recurrence time of class \( ij \) outgoing messages is the reciprocal of the mean entrance rate of transition \( ij \), \( \psi \tilde{\eta}_{ij} \), i.e., \( \frac{\psi \tilde{\eta}_{ij} + \psi \tilde{\eta}_{ij}}{\psi \tilde{\eta}_{ij}} \).

Let \( \tilde{\eta} \) be the mean overall throughput rate of the outgoing messages in a SyPEFSM. It is the reciprocal of the overall mean transition time of the SyPEFSM, which is equal to the mean overall holding time at a state of the SyPEFSM. Thus,

\[ \tilde{\eta} = \frac{1}{\psi \tilde{\eta}}. \]

\( \tilde{\eta} \) can also be computed from the weighted sum of the mean throughput rates of all the outgoing message classes. Both the results are the same:

\[
\tilde{\eta} = \sum_{i \in Q} \sum_{j \in Q} \psi \tilde{\eta}_{ij} \\
= \sum_{i \in Q} \sum_{j \in Q} \psi \tilde{\eta}_{ij} \cdot p_{ij} \\
= \sum_{i \in Q} \sum_{j \in Q} \frac{\pi_i}{\psi \tilde{\eta}} \cdot p_{ij} \\
= \frac{1}{\psi \tilde{\eta}}.
\]

### 4.4 Related work

A closely related work to this chapter can be found in [KrWh93]. That work also uses the theory of semi-Markov processes (CMPs) to derive performance for a given FSM of a communication protocol. The FSM of a protocol in that paper is first translated into an intermediate representation called transition relation graph. A node in the transition relation graph of an FSM represents a transition
of the FSM, and a transition in the graph corresponds to a state in the FSM. All the transitions of an FSM in [KrWh93] are assumed to be atomic and take no time to execute, but the occupancy of a state in the FSM does span over time. After the transformation from an FSM to a transition relation graph, a transition in the graph then takes the same amount of time as the corresponding state in the FSM.

Unlike [KrWh93], the transformation from an FSM to a transition relation graph is not necessary for SyPEFSMs. This is because in a SyPEFSM, transitions and state occupancies are already associated with time.

[KrWh93] did not consider the waiting time for an incoming message (i.e., \( w_{r_{ij}} \)) in any transition time. Rather, that research treats a transition time as consisting of only the transition service time, \( s_{r_{ij}} \). As a result, only the performance metrics in Section 4.3.1 of this chapter could be derived.

[KrWh93] assumes that all the transition times are discrete and have geometric distributions. In contrast, we do not impose any restriction on the distributions of transition service times or transition waiting times.

Other work whose performance evaluation methodologies are similar to [KrWh93] can be found in [GrDu87, Rico91, Mars94]. Likewise, they also have the problems discussed above.

### 4.5 Summary

An arrival model for SyPEFSMs is presented in this chapter. The arrival model is comprised of \( P \) – the transition probabilities and \( \mu H \) – the p.d.f.s of the waiting times for firable messages of each state. The p.d.f.s of transition wait times \( \mu h_{ij}(t) \)
and transition service times $h_{ij}(t)$ ($i, j \in Q$) can assume any distribution function as long as the variables of transition wait times $w_{ij}$ and transition service times $s_{ij}$ are independent of each other.

Given a SyPEFSM with this arrival model, the performance measures of the SyPEFSM such as state recurrence times and throughputs can be obtained and shown in this chapter.

The bounds of these performance measures of a given SyPEFSM are also derived by analyzing its Model S. These bounds do not depend on the arrival model of the SyPEFSM.

The results in this chapter form the basis for the evaluation of the more complex PEFSMs in the next two chapters.
Chapter 5

Performance Evaluation of Asynchronous PEFSMs

Abstract

An asynchronous PEFSM (AsPEFSM) is a PEFSM whose incoming messages arrive asynchronously. This chapter examines two types of possible asynchronous arrival models for AsPEFSMs. Performance metrics such as utilization and queue wait times of AsPEFSMs are derived for both types of arrival models.

5.1 Introduction

In the previous chapter, we have discussed the performance evaluation of synchronous PEFSMs (SyPEFSMs). The essential difference between asynchronous PEFSMs (AsPEFSMs) and SyPEFSMs is that the arrivals of incoming messages in AsPEFSMs are asynchronous. Messages may arrive at an AsPEFSM independent of the states of the AsPEFSM and therefore, may arrive before the AsPEFSM is ready to process them.
Examples of AsPEFSMs include the FSM of the service model of the asynchronous transfer mode (ATM) switch that processes Markov modulated traffic and an FSM of the token ring protocol (IEEE 802.5). The former model has been introduced in Chapter 3. The latter is an AsPEFSM because both the token and the data may arrive at the server asynchronously at any time.

An AsPEFSM has no control or influence on the arrivals of incoming messages. A message to an AsPEFSM may arrive before the AsPEFSM enters the state where the message is firable. When this happens, the messages which *arrive too early* will be stored in a buffer, commonly called a *queue*. Whenever an AsPEFSM enters a new state, the queue of the AsPEFSM is first checked to see if there are firable messages of the state. If so, one of the firable messages of that state is processed immediately; otherwise, the AsPEFSM has to wait. In the first case, the wait time of the AsPEFSM upon entering the state is zero; in the second case, the wait time depends on the arrival distributions of the firable messages of that state. The computation method for this wait time is presented in this chapter.

The wait times at states of an AsPEFSM are important indices in evaluating the performance of the AsPEFSM. These times influence the queue wait times of firable messages of the other states.

Since messages may be queued up from time to time, the queue wait times of incoming messages to an AsPEFSM are also important performance indices. Queue wait times can be more easily computed using a queueing model. For that reason, in addition to FSMs and stochastic processes, queueing models are employed in evaluating the performance of an AsPEFSM.

Queueing models are able to model *service delays* due to *resource contention*
Chapter 5: performance evaluation of AsPEFSMs

by messages. Conventional queueing models do not take into account the order of message arrivals and the order of services controlled by an FSM. However, these orders are important in an AsPEFSM and they influence the performance of the protocol specified by the AsPEFSM.

A natural way to integrate this ordering information is to construct a hybrid model consisting of both the AsPEFSM and the queueing model.

5.1.1 Queueing models

We assume the AsPEFSM of a protocol runs on a single processor computer system. This implies the queueing models of an AsPEFSM are single-server models.¹

The queueing discipline of incoming messages in each queue of an AsPEFSM is first in first out (FIFO). When there is more than one class of firable messages available at the time an AsPEFSM enters a state, the service discipline is assumed to be first come first served (FCFS). No other priority is assumed.

5.1.2 Arrival models

In contrast to SyPEFSMs, since the arrivals of incoming messages to an AsPEFSM are asynchronous and uncontrolled by the AsPEFSM, it is difficult to give a general arrival model of incoming messages for all AsPEFSMs.

In this chapter, two kinds of typical arrival models of AsPEFSMs are examined. The first assumes the arrivals of incoming messages to an AsPEFSM are uncorrelated. Incoming messages of different classes arrive independently to the

¹To extend the results to multiprocessor systems, it is necessary only to replace the single server queueing model by a multi-server queueing model.
AsPEFSM. The interarrival times of incoming messages within each class have an exponential distribution. An AsPEFSM with this kind of arrival model is called an AsPEFSM-α.

The second arrival model assumes the arrivals of incoming messages to an AsPEFSM are correlated. The order (but not the timing) of incoming messages to an AsPEFSM corresponds to one of the transition sequences in the AsPEFSM. A sequence consists of the consecutive transitions of the AsPEFSM. The interarrival times of incoming messages, regardless of classes, are assumed to be independent and identically distributed. Assume the distribution is exponential. An AsPEFSM with this kind of arrival model is called an AsPEFSM-β.

The above are the informal descriptions about the two arrival models. Formal definitions will be given in the later sections. The performance metrics of the two kinds of AsPEFSMs are derived in this chapter.

The rest of this chapter is organized as follows. Sections 5.2 and 5.3 present the performance metrics and their derivations with respect to AsPEFSM-αs and AsPEFSM-βs, respectively. Section 5.4 discusses the related work and Section 5.5 summarizes this chapter.

5.2 AsPEFSM-α

As mentioned, each class of incoming messages to an AsPEFSM-α arrives independently. An arrival model with this property is introduced below and the performance metrics of an AsPEFSM with this arrival model are derived.
5.2.1 α-arrival model

Let $A_{ik} = \{a_{ik}(0), a_{ik}(1), a_{ik}(2), \ldots\}$ be the ordered set of the arrival epochs\(^2\) of incoming messages of class $ik$ of an AsPEFSM ($i, k \in Q$):

$$a_{ik}(0) < a_{ik}(1) < a_{ik}(2) < \cdots$$

**Definition 5.1 (α-arrival model)** An arrival model of incoming messages to an AsPEFSM is called an α-arrival model if the arrival model satisfies the following conditions:

1. the elements in $A_{ik}$ are uncorrelated with those in $A_{jl}$ if $i \neq j$ or $k \neq l$ ($i, j, k, l \in Q$);

2. $(a_{ik}(n + 1) - a_{ik}(n))$ ($i, k \in Q$ and $n = 0, 1, 2, \ldots$) are exponentially distributed.

**Definition 5.2 (AsPEFSM-α)** An AsPEFSM is called an AsPEFSM-α if the arrival pattern of its incoming messages follows an α-arrival model.

5.2.2 Queueing and service model

The arrivals of incoming messages of each class of an AsPEFSM are independent of the current state as well as other incoming message classes of the AsPEFSM-α. Messages may come before they are ready to be processed in which case they are stored in queues. We assume that there is a FIFO queue for each incoming message class of the AsPEFSM-α.

\(^2\)The time is assumed to be recorded according to a global clock.
Furthmore, we assume there is only one service center for an AsPEFSM-α and the service discipline is FCFS.

Suppose that there is more than one transition out of a state in an AsPEFSM-α. Because each transition is associated with a message class, the number of incoming message classes of a state is the same as the number of the transitions starting from the state. When there are firable messages belonging to more than one class waiting in their respective queues at the time the AsPEFSM enters a state, the earliest arrived firable message of the state will be selected for service and correspondingly, the transition associated with this message will be executed.

Let $\min(\cdot)$ be a selecting function;

$\min(D)$ be equal to the smallest element in set $D$;

$\arg_k(f(x_k))$ return the index of the variable $x_k$ which satisfies the function $f$;

$X_n$ represent the $n$-th state ($n = 0, 1, 2, \ldots$) entered by the AsPEFSM-α;

$t_n$ represent the epoch when the AsPEFSM-α enters the $n$-th state.

The queueing and service model of an AsPEFSM-α is defined with the following procedure:

1. if $X_n = i$, then $X_{n+1} = \arg_k(\min(\{\min(A_{ik})\}))$;

2. if $X_{n+1} = j$, then $A_{ij} = A_{ij} - \{\min(A_{ij})\}$;

3. goto Step 1

where $i, j, k, l \in Q$. This is explained below.

In Step 1, the firable message of state $i$ which arrives the earliest among the unserved firable messages is selected and the transition associated with this
message is made. All the messages of class $ij$ whose arrival times are still in the set $A_{ik}$ are not yet served. $\min(A_{ik})$ returns the arrival time of the earliest arrived but unserved message of class $ik$. $\min(\{\min(A_{ik})\}) \ (k \in Q)$ returns the arrival time of the earliest but unserved message among the different classes of firable messages in state $i$. $\arg_k(\min(\{\min(A_{ik})\}))$ returns the index of the class to which this selected message belongs. In Step 2, the arrival time of the selected message is removed from the set in which the time is kept. This means the message is served and will not be considered again.

This type of queueing and service model for AsPEFSM-\(\alpha\) is illustrated graphically in Figure 5.1. A controller (labelled as B) controls the selection of the next message to be served based on the AsPEFSM-\(\alpha\)'s current state (labelled as A) and the arrival times of the messages in the queues.

![Figure 5.1: The queueing system for an AsPEFSM-\(\alpha\)](image)

We can also formalize the queue wait time of a firable message and the queue length of a specific class of incoming messages as follows:

1. the transition wait time

\[
 w_{ij}^T = \begin{cases} 
 0 & \text{if } \min(A_{ij}) \leq t_n \\
 \min(A_{ij}) - t_n & \text{if } \min(A_{ij}) > t_n
\end{cases}
\]
2. the queue wait time of the firable message which is selected at \( t_n \)

\[
W_{q,ij} = \begin{cases} 
  t_n - \min(A_{ij}) & \min(A_{ij}) \leq t_n \\
  0 & \min(A_{ij}) > t_n 
\end{cases}
\]

3. the queue length of class \( k \) incoming message at \( t_n \) is the number of the elements in \( A_{kl} \) at \( t_n \) which satisfy \( a_{kl}(\cdot) \leq t_n \) where \( k, l \in Q \).

### 5.2.3 Performance Metrics

Let \((M, P)\) and \(P = (P^*, H^*, \Phi)\) be the components of an AsPEFSM-\( \alpha \). The definitions are given in Chapter 3.

Let \( \lambda_{ij} \) be the mean arrival rate of class \( ij \) incoming messages of an AsPEFSM-\( \alpha \) \((i, j \in Q)\).

In this section, \( M, H^* \) and \( \lambda_{ij} \) \((i, j \in Q)\) of an AsPEFSM-\( \alpha \) are assumed to be given by the performance evaluator. The other parameters will be derived from this given information.

To obtain the performance metrics for an AsPEFSM-\( \alpha \), \( P \) and \( H^* \) of the AsPEFSM-\( \alpha \) need to be derived first.

**Computation of \( P \) and \( H^* \)**

Let \( p_{ij} \) be the single-step probability of transition \( ij \). According to the definition of \( P \) in Chapter 3, we have

\[
p_{ij} = Pr\{X_{n+1} = j | X_n = i\}.
\]

Since the transitions and the incoming message classes are one-to-one mapped, \( p_{ij} \) is the probability that a firable message of class \( ij \) is selected when the AsPEFSM-\( \alpha \) enters state \( i \).
Based on the queueing and service model of the AsPEFSM-α, when the AsPEFSM-α enters state i, the firable message to be served is the one which comes first after the arrival of the last served firable message of state i. Let \( c_{ij}(t) \) be the probability that the first firable message of state i arrives at time t after the last served firable message arrived and this message is of class \( ij \).

Applying the memoryless property of Poisson arrivals, we can derive

\[
c_{ij}(t) = Pr\{\text{no firable message of state } i \text{ arrives within time } t \\
\text{and a class } ij \text{ message arrives at time } t\}
\]

\[
= \lambda_{ij} e^{-\lambda_{ij} t} \prod_{k=0, k \neq j}^{Q} e^{-\lambda_{ik} t}.
\]

Therefore,

\[
p_{ij} = \int_{0}^{\infty} c_{ij}(t) dt = \frac{\lambda_{ij}}{\sum_{k \in Q} \lambda_{ik}}.
\]

From the above equation, it is seen that the transition probability matrix \( P = [p_{ij}] \) can be computed directly from the given mean arrival rates of incoming messages.

Furthermore, by Theorem A.1, the steady-state state probabilities \( \pi \) can be computed by solving the following equation

\[
\pi P = \pi.
\]

It means that given the arrival rates and the AsPEFSM-α, we can compute \( P \) and \( \pi \). Therefore, we assume \( P \) and \( \pi \) are known hereafter.

Next, we compute \( \psi H \).

By the definition of transition time given in Chapter 3,

\[
\psi_{\tau_{ij}} = \upsilon_{\tau_{ij}} + \xi_{\tau_{ij}} \tag{5.1}
\]
where \( \psi_{ij} \) is the transition time of transition \( ij \), \( w_{ij} \) is the time for the AsPEFSM-\( \alpha \) to wait for a firable message of class \( ij \), and \( \tau_{ij} \) is the service time of transition \( ij \) which is given by the performance evaluator. Both \( w_{ij} \) and \( \psi_{ij} \) can be computed in the following way.

Remember that we have assumed the interarrival times of incoming messages in each class to be uncorrelated and exponentially distributed. It is also well-known that if the interarrival times of the jobs are exponentially distributed, then the number of the arrivals within a certain time is Poisson distributed, and vice versa. Therefore, the arrival patterns of incoming messages for all the classes in an AsPEFSM-\( \alpha \) are independent Poissons.

As a result, the arrivals of firable messages of state \( i \) regardless of their classes are also Poisson. Denote the mean arrival rate of the messages in state \( i \) by \( \lambda_i \). We have

\[
\lambda_i = \sum_{j \in Q} \lambda_{ij}.
\]

Let NoF denote the event that the AsPEFSM finds no firable message of state \( i \) in the queue when it enters state \( i \) and \( \overline{\text{NoF}} \) denote the reverse of NoF (i.e., at least one firable message is available). Let \( p_{0,i} \) be the probability of NoF (i.e., \( p_{0,i} = Pr\{\text{NoF}\} \) and \( 0 \leq p_{0,i} \leq 1 \)) when the AsPEFSM is in equilibrium.

By the queueing and service model of an AsPEFSM-\( \alpha \), it is true that the wait time of the AsPEFSM-\( \alpha \) is equal to zero when the AsPEFSM-\( \alpha \) enters a state and finds at least one firable message of the state. Formally,

\[
Pr\{w_{ij} = 0|X_n = i, X_{n+1} = j, \overline{\text{NoF}}\} = 1
\]

or

\[
Pr\{w_{ij} > 0|X_n = i, X_{n+1} = j, \overline{\text{NoF}}\} = 0.
\]
Therefore,
\[ \psi_{ij}(t) = 1 - p_{0,i}, \]
and
\[ \psi_{ij}(t) = Pr\{ w_t = t | X_n = i, X_{n+1} = j \} \]
\[ = Pr\{ w_t = t | X_n = i, X_{n+1} = j, NoF \} \cdot Pr\{ NoF \} + Pr\{ w_t = t | X_n = i, X_{n+1} = j, NoF \} \cdot Pr\{ NoF \} \]
\[ = Pr\{ w_t = t | X_n = i, X_{n+1} = j, NoF \} \cdot p_{0,i} \]
\[ = p_{0,i} \cdot \frac{Pr\{ (X_{n+1} = j, w_t = t) | X_n = i, NoF \}}{Pr\{ X_{n+1} = j | X_n = i, NoF \}} \]
\[ = p_{0,i} \cdot \frac{\lambda_{ij} e^{-\lambda_{it}} \cdot \prod_{k=0, k \neq j}^{Q} e^{-\lambda_{ik} t}}{\lambda_{ii}} \]
\[ = p_{0,i} \cdot \sum_{k \in Q} \lambda_{ik} e^{-\lambda_{it} t} \]
\[ = p_{0,i} \cdot \lambda_{i} e^{-\lambda_{it} t} \]

where \( t > 0 \) and \( w_t \) is the wait time for a firable message.

Since \( \psi_{ij}(t) \) is the p.d.f. of \( w_{ij} \),
\[ w_{ij} = \int_0^\infty t \psi_{ij}(t) dt = p_{0,i} \cdot \frac{1}{\lambda_{i}}. \tag{5.2} \]

Apply this result in (5.1) and get
\[ \psi_{ij} = p_{0,i} \cdot \frac{1}{\lambda_{i}} + \tau_{ij} \tag{5.3} \]

where \( 0 \leq p_{0,i} \leq 1 \) and \( i, j \in Q \).

We still need to compute \( p_{0,i} \) – the probability of \( NoF \).

Focusing on state \( i \) of an AsPEFSM-\( \alpha \), we first give the following definitions.
Chapter 5: performance evaluation of AsPEFSMs

Definition 5.3 (first passage) A first passage from state $i$ to $j$ in an FSM is a sequence of consecutive transitions of the FSM which starts from state $i$, ends in state $j$ and never passes state $j$ in between.

Definition 5.4 (virtual job) A virtual job with respect to state $i$ in an AsPEFSM-$\alpha$ consists of the transitions on a first passage from state $i$ to itself in the FSM of the AsPEFSM-$\alpha$.

Definition 5.5 (virtual job class) Class $ij$ virtual jobs in an AsPEFSM-$\alpha$ are the set of the first passages from state $i$ to $i$ in the FSM of the AsPEFSM-$\alpha$. The first transition of each first passage in this set is transition $ij$ ($i, j \in Q$).

Suppose there are $n$ transitions out of state $i$. Applying the above three definitions, we transform all the first passages from state $i$ to $i$ in an AsPEFSM-$\alpha$ into $n$ virtual job classes. This transformation is illustrated in Figure 5.2.

The arrival rate of class $ij$ virtual jobs of an AsPEFSM-$\alpha$ is equal to the arrival rate of class $ij$ incoming messages $\lambda_{ij}$. Let $\zeta_{ij}$ denote the service time of a virtual job of class $ij$. If transition $ij$ is a self-loop transition from state $i$ to $i$, then the service time of a class $ij$ virtual job is equal to the service time of this transition; otherwise, the service time of the class $ij$ virtual job is the service
time of transition $ij$ plus the time spent on the first passage from $j$ to $i$ in the AsPEFSM-$\alpha$, i.e.,

$$
\zeta_{ij} = \begin{cases} 
\tau_{ii} & \text{if } i = j \\
\tau_{ij} + \theta_{ji} & \text{if } i \neq j 
\end{cases}
$$

(5.4)

where $\theta_{ji}$ denotes the first passage time from state $j$ to $i$ in the AsPEFSM-$\alpha$. $\theta_{ji}$ consists of all the transition times on the first passage and does not include any queue wait time.

We can write the mean value equation as

$$
\bar{\zeta}_{ij} = \begin{cases} 
\tau_{ii} & \text{if } i = j \\
\bar{\tau}_{ij} + \bar{\theta}_{ji} & \text{if } i \neq j \end{cases}
$$

where $\bar{\zeta}_{ij}$ is the mean of $\zeta_{ij}$; $\bar{\theta}_{ji}$ is the mean of $\theta_{ji}$.

The mean service time of class $ij$ virtual jobs, $\bar{\zeta}_{ij}$, can be computed as long as $\bar{\tau}_{ij}$ and $\bar{\theta}_{ji}$ are known. $\bar{\tau}_{ij}$, by definition, is part of the primitive data provided by the evaluator. From Theorem A.3, $\bar{\theta}_{ji}$ can be computed by solving the following recurrence equation

$$
\bar{\theta}_{ji} = \bar{\tau}_{j} + \sum_{k \in Q} p_{jk} \bar{\theta}_{ki} - p_{ji} \bar{\theta}_{ii} \quad (j \neq i)
$$

where $i, j \in Q$.

Applying the results we have obtained to the above equation, we further derive

$$
\bar{\theta}_{ji} = \bar{\tau}_{j} + \sum_{k \in Q} p_{jk} \bar{\theta}_{ki} - p_{ji} \bar{\theta}_{ii} \quad (5.5)
$$

$$
= \sum_{k \in Q} p_{jk} \bar{\tau}_{jk} + \sum_{k \in Q} p_{jk} \bar{\theta}_{ki} - p_{ji} \bar{\theta}_{ii} \quad (5.6)
$$

$$
= \sum_{k \in Q} p_{jk} \left( \frac{1}{\lambda_j} + \bar{\tau}_{jk} \right) + \sum_{k \in Q} p_{jk} \bar{\theta}_{ki} - p_{ji} \bar{\theta}_{ii} \quad (5.7)
$$

$$
= \frac{p_{0,j}}{\lambda_j} + \bar{\tau}_{j} + \sum_{k \in Q} p_{jk} \bar{\theta}_{ki} - p_{ji} \bar{\theta}_{ii} \quad (5.8)
$$
where \( i, j \in Q \) and \( j \neq i \).

This is still a recurrence equation, in which \( \lambda_j, p_{jk}, \) and \( w^r_j \) \((j, k \in Q)\) are already known from the given primitive data of the AsPEFSM-\( \alpha \). \( w^r_{ii} \) is unknown but it is the mean first passage time from state \( i \) to itself in the AsPEFSM-\( \alpha \). We also call \( \bar{\theta}_{ii} \) the mean recurrence time of state \( i \) of an AsPEFSM-\( \alpha \). \( \bar{\theta}_{ii} \) can be computed in the following way.

Let \( \bar{\zeta}_j \) be the mean service time of the overall virtual jobs from state \( i \) to \( i \) in an AsPEFSM-\( \alpha \). Then,

\[
\bar{\zeta}_i = \sum_{j \in Q} p_{ij} \bar{\zeta}_{ij}.
\]

By the definitions of \( \bar{\theta}_{ii} \) and \( \bar{\zeta}_i \), we have

\[
\bar{\theta}_{ii} = \sum_{j \in Q} p_{ij} (w^r_{ij} + \bar{\zeta}_{ij}) = \sum_{j \in Q} p_{ij} w^r_{ij} + \sum_{j \in Q} p_{ij} \bar{\zeta}_{ij} = \sum_{j \in Q} p_{ij} \frac{p_{0,j}}{\lambda_i} + \bar{\zeta}_i = \frac{p_{0,i}}{\lambda_i} + \bar{\zeta}_i.
\]

With the above construction of the virtual jobs, the queueing system serving incoming messages of an AsPEFSM-\( \alpha \) can be viewed as a queueing system serving the virtual jobs. \( p_{0,i} \) in turn is the idle probability that the transformed queueing system is waiting for virtual jobs.

Let \( \rho_i \) be the utilization of this queueing system. When \( (\lambda_i \bar{\zeta}_i) < 1 \),

\[
\rho_i = \lambda_i \cdot \bar{\zeta}_i.
\]
According to [GrHa85](p.86), for a single server queueing system when the system is in equilibrium, it is true that

\[ p_{0,i} = 1 - \rho_i \quad (\rho_i < 1). \]  

(5.14)

Substituting Equations (5.14) and (5.13) in Equation (5.12), we get

\[ \psi_{ji} = \frac{1}{\lambda_i}. \]

This result is under the condition that the system running the virtual jobs is in equilibrium, i.e., \( (\lambda_i \bar{\zeta}_i < 1) \).

Now we can rewrite Equation (5.8) with this result as:

\[ \psi_{ji} = \frac{p_{0,j}}{\lambda_j} + \zeta_j + \sum_{k \in Q} p_{jk} \psi_{ki} \frac{1}{\lambda_i}. \]  

(5.15)

Moreover, we rewrite Equation (5.14) as:

\[ p_{0,i} = 1 - \rho_i \]

(5.16)

\[ = 1 - \lambda_i \bar{\zeta}_i \]

(5.17)

\[ = 1 - \sum_{j \in Q} \lambda_{ij} \bar{\zeta}_{ij} \]

(5.18)

\[ = 1 - \left( \sum_{j \in Q} \lambda_{ij} (\zeta_{ij} + \psi_{ji}) - \lambda_{ii} \psi_{ii} \right) \]

(5.19)

\[ = 1 - \left( \sum_{j \in Q} \lambda_{ij} (\zeta_{ij} + \psi_{ji}) - \lambda_{ii} \frac{1}{\lambda_i} \right) \]

(5.20)

\[ = 1 - \left( \sum_{j \in Q} \lambda_{ij} (\zeta_{ij} + \psi_{ji}) - p_{ii} \right) \]

(5.21)

In Equation (5.15), \( p_{0,j} \) and \( \psi_{ji} \) \( (i, j \in Q) \) are unknown. In Equation (5.21), \( p_{0,i} \) and \( \psi_{ji} \) are also to be determined. Therefore, solving the equations stated...
in (5.15) and (5.21) for all \( i, j \in Q \) together, we can find \( p_{0,i} \) and \( \psi_{ji} \) if they exist.

With \( p_{0,i} \) \((i \in Q)\) known, \( \tau_{ij} \) in Equation (5.2) can be computed, and so can \( \psi_{ij} \).

In many cases, \( p_{0,i} = 0 \). The following theorem helps to identify some of these cases.

**Theorem 5.1** In an AsPEFSM-\( \alpha \), if one of the following conditions is satisfied, then \( p_{0,i} = 0 \) \((i \in Q)\):

1. \( \lambda_i \bar{\theta}_{ii} > 1 \) where \( \bar{\theta}_{ii} \) is the mean first passage time from state \( i \) to \( i \) in Model \( S \) of the AsPEFSM-\( \alpha \);

2. \( \lambda_i > \sum_{j \in Q} \lambda_{ji} \).

**Proof:**

By the definition of *virtual job*, we note that the service time of each virtual job of state \( i \) contains the service times of the transitions on a first passage from state \( i \) to state \( i \) in the AsPEFSM-\( \alpha \). Therefore, \( \bar{\zeta}_i > \bar{\theta}_{ii} \). If \( \lambda_i \bar{\theta}_{ii} > 1 \), then \( \lambda_i \bar{\zeta}_i > 1 \) and this means the queuing system serving the virtual jobs is saturated. In other words, there are always virtual jobs (i.e., firable messages of state \( i \)) waiting in the queue. Hence, \( p_{0,i} = 0 \).

To prove the second situation, we refer to Figure 5.3. From the figure, it can be seen that \( \sum_{j \in Q} \lambda_{ji} \) is the total possible arrival rate of *entry messages* which put the AsPEFSM in state \( i \), while \( \lambda_i = \sum_{k \in Q} \lambda_{ik} \) is the total possible arrival rate of *leaving messages* which make the AsPEFSM leave state \( i \).

Let \( X_{in}(t) \) be the total number of entry messages that arrive within time \( t \),
Chapter 5: performance evaluation of AsPEFSMs

Figure 5.3: Messages come to and leave state \( i \) in an AsPEFSM-\( \alpha \)

\( X_{\text{out}}(t) \) be the total number of leaving messages that have arrived within the same period, and \( X(t) \) be the total unserved messages leaving state \( i \) after the period.

We note that each entry message of state \( i \) consumes only one leaving message of state \( i \). Therefore, we can write

\[
X(t) = \begin{cases} 
(X_{\text{in}}(t) - X_{\text{out}}(t)) & \text{if } X_{\text{in}}(t) > X_{\text{out}}(t) \\
0 & \text{if } X_{\text{in}}(t) \leq X_{\text{out}}(t)
\end{cases}
\]  

(5.22)

Their mean value relationship can be expressed as

\[
E[X(t)] = \begin{cases} 
(\lambda_i - \sum_{j \in Q} \lambda_{ji}) \cdot t & \text{if } \lambda_i > \sum_{j \in Q} \lambda_{ji} \\
0 & \text{if } \lambda_i \leq \sum_{j \in Q} \lambda_{ji}
\end{cases}
\]  

(5.23)

Obviously, if \( \lambda_i > \sum_{j \in Q} \lambda_{ji} \) and \( t \to \infty \), then \( E[X(t)] \to \infty \). This means the unserved leaving messages of state \( i \) will accumulate in the queue, i.e., firable messages of state \( i \) are always available in the queue. Therefore, \( p_{0,i} = 0 \).

Utilization

The utilization \( \rho_i \) defined earlier is in fact only a virtual utilization of an AsPEFSM-\( \alpha \). This utilization is not the real (actual) utilization of the queueing system serving the AsPEFSM-\( \alpha \) because it does not take into consideration the idle periods within the first passages from state \( i \) to state \( i \).
Let $\rho$ be the (real) utilization of the queueing system of an AsPEFSM-\(\alpha\) ($\rho < 1$).

When the AsPEFSM-\(\alpha\) is in equilibrium, we have

$$\rho = \frac{\sigma_i}{\psi_i}.\nonumber$$

Applying Theorems A.5 and A.4, we get the recurrence time of state \(i\) as

$$\psi_{ii} = \frac{1}{\psi_{\infty i}} = \frac{\psi_{i\infty}}{\pi_i}.\nonumber$$

So that

$$\psi_r = \pi_i \psi_{ii} = \frac{\pi_i}{\lambda_i}. \quad (5.24)\nonumber$$

Substituting the above equation into Equation (5.2.3), we rewrite

$$\rho = \frac{\lambda_i \psi_r}{\pi_i}.\nonumber$$

The AsPEFSM is in equilibrium only if $\rho < 1$. Therefore,

$$\sigma_i < \frac{\pi_i}{\lambda_i} \quad (i \in Q)$$

is the necessary condition for equilibrium.

The following theorem follows directly from Equation (5.24):

**Theorem 5.2** When an AsPEFSM-\(\alpha\) is in equilibrium and $\rho < 1$, the following equation holds

$$\frac{\pi_i}{\lambda_i} = \frac{\pi_j}{\lambda_j} \quad \text{for all } i, j \in Q.$$
Throughputs

With $\tilde{\psi}_{ii}$ – the mean recurrence time of state $i$ of an AsPEFSM-$\alpha$, the mean throughput rate of the outgoing messages of any specific class in the AsPEFSM-$\alpha$ can be computed in a similar way given in (4.2).

When an AsPEFSM-$\alpha$ is in equilibrium, the transition service rate is equal to its entrance rate.

Let $\bar{\eta}_{ij}$ be the throughput rate of the outgoing messages associated with transition $ij$ out of state $i$.

Then,

\begin{align*}
\bar{\eta}_{ij} &= \psi_{ei}p_{ij} \\
&= \frac{1}{\tilde{\psi}_{ii}}p_{ij} \\
&= \lambda_ip_{ij} \\
&= \lambda_{ij}.
\end{align*}

The above result confirms the intuition: throughput rate = input rate for a system in equilibrium.

Queue wait times

First, we derive some queueing properties for $M/G/1^3$ queueing systems and AsPEFSM-$\alpha$s from a previously proven lemma:

**Lemma 5.3** Suppose Poisson arrivals observe a stochastic process as well as interact with it. If the process being observed cannot anticipate the future state $^3M/G/1$ is a single-server queueing system which has Poisson job arrivals and general distributed service times.
change of the Poisson process, then the fraction of arrivals that see the process in some state is equal to the fraction of time the process is in that state.

Proof:

See [Wolf82, MeWh90].

This lemma is called Poisson Arrivals Seeing Time Average (PASTA). The first proof was given in [Wolf82]. A new proof with a less restricted condition is given in [MeWh90].

According to this lemma, in an M/G/1 queueing system, the fraction of arrivals which find \( n \) customers in the system is equal to the fraction of time the system has \( n \) customers. For a M/G/1 queueing system with multiple job classes, we can derive the following theorem:

**Theorem 5.4** In an M/G/1 with multiple job classes, if the jobs in each class arrive in an independent Poisson pattern with the same mean, the mean queue wait times of each class are equal.

Proof:

Since the arrivals of the jobs of a specific class are Poisson, by Lemma 5.3, the mean number of jobs in the system at the arrival points of this class is equal to the average number of jobs in the system over time. By Little's Law \( L = \lambda W \) [Litt61], the mean queue wait time of each class is the same and is equal to the overall mean queue wait time.

Based on the above theorem, the formula to compute the mean overall queue wait time of all the messages for an M/G/1 with multiple job classes (see [Allen90](p.688)) can also be applied to compute the mean queue wait time of the messages of a specific class.
Chapter 5: performance evaluation of AsPEFSMs

There is also an analogous queueing property with AsPEFSM-αs. In the following, we prove the property also using PASTA.

Let the queueing system state of an AsPEFSM-α consist of the information on incoming messages in each queue, the current state which the AsPEFSM-α is in and the current transition which is in service.

Since we have assumed that in an AsPEFSM-α, the arrivals of incoming messages of each class are in an independent Poisson pattern, the following theorem is true for AsPEFSM-αs.

**Theorem 5.5** The probability of each queueing system state seen by the incoming messages of a specific class at their arrivals is equal to the fraction of time the system is in that state.

*Proof:* 

Direct from Lemma 5.3.

With Theorem 5.5, we can conclude:

**Theorem 5.6** The mean queue wait times of different classes of firable messages in the same state of an AsPEFSM-α are equal.

*Proof:* 

Since each class of messages arrive in Poisson, each message has the same probability of encountering a specific queueing system state as the other messages. The firable messages of the same state in the AsPEFSM-α have the same probability of waiting for a certain amount of time before the AsPEFSM-α enters this state where the firable messages can be served. Therefore, the mean queue wait times of the firable messages of different classes of the same state are the same.
Let $\bar{W}_{q,i}$ be the mean queue wait time of all the classes of the firable messages of state $i$ in an AsPEFSM-$\alpha$, and $\bar{W}_{q,ij}$ be the mean queue wait time of class $ij$ incoming messages. By Theorem 5.6, we have

$$\bar{W}_{q,i} = \bar{W}_{q,ij} \quad (i, j \in Q).$$

Therefore, only $\bar{W}_{q,i}$ of each state in an AsPEFSM-$\alpha$ need to be computed.

Recall the construction of the virtual jobs described earlier in this section. Virtual jobs from the same state, say state $i$, in an AsPEFSM-$\alpha$ are categorized into different classes with respect to the first transitions on each first passage from state $i$ to state $i$. This transforms the original queueing system serving the AsPEFSM-$\alpha$ into an $M/G/1$ with multiple job classes. Each class of jobs consists of the virtual jobs which start with a transition from state $i$.

By the definition of first passage, $\psi_{ji}$ of an AsPEFSM-$\alpha$ includes the possible idle periods when the AsPEFSM-$\alpha$ waits for firable messages during a first passage. For a specific first passage, these times are independent from the past occupancies of the states of the AsPEFSM-$\alpha$. The reason for this is that Poisson arrivals have the memoryless property.

Moreover, $\psi_{ji}$ is not related to the next arrival of any class of firable messages of state $i$ in an AsPEFSM-$\alpha$. This is because by definition, a first passage from state $j$ to $i$ does not contain any transition out of state $i$.

Therefore, the service time of a class $ij$ virtual job, $\zeta_{ij}$, does not depend on the arrival rate of any class of firable messages of state $i$ in the AsPEFSM-$\alpha$. As a result, the queueing system which serves these virtual jobs is a conventional $M/G/1$ with multiple job classes. The queueing theory of $M/G/1$ can be employed for this queueing system.
Let $\overline{\zeta_i^2}$ be the second moment of the service times of the virtual jobs from state $i$ to itself in an AsPEFSM-$\alpha$. Then, by definition,

$$\overline{\zeta_i^2} = \sum_{j \in Q} \frac{\lambda_{ij}}{\lambda_i} \cdot \overline{\zeta_j^2}.$$  

$\overline{\zeta_{ij}^2}$ can be computed from the definition of $\zeta_{ij}$ given in Equation (5.4). The detail is given in Appendix B.

By applying the solution techniques for M/G/1 (see [Allen90](p.688)), the mean queue wait time of the messages out of state $i$, $\overline{W_{q,i}}$, is

$$\overline{W_{q,i}} = \frac{\lambda_i \overline{\zeta_i^2}}{2(1 - \rho_i)} = \frac{\sum_{j \in Q} \lambda_{ij} \overline{\zeta_j^2}}{2(1 - \rho_i)}.$$  

(5.29)

The mean number of the firable messages of class $ij$ in the queue, $L_{q,i}$, can be computed using Little's Law [Litt61]:

$$L_{q,i} = \lambda_i \overline{W_{q,i}}.$$  

5.3 AsPEFSM-$\beta$

In the AsPEFSM-$\alpha$ model, incoming messages of different classes are uncorrelated. However, some communication and distributed systems, the arrivals of incoming messages are correlated. In this section, we study a simple type of correlated arrival model, called $\beta$-arrival model. An AsPEFSM with this type of input is called an AsPEFSM-$\beta$.

5.3.1 $\beta$-arrival model

We assume that the correlation of the arriving incoming messages is specified in the FSM of an AsPEFSM as follows:
Let $A = \{a(0), a(1), a(2), \ldots\}$ be the ordered set of the arrival epochs of all the incoming messages to an AsPEFSM without distinguishing their class types:

$$a(0) < a(1) < a(2) < \cdots$$

Let $C(t)$ be the function which returns the class of the message which arrives at time $t$. $C(t) = \emptyset$ if no message arrives at time $t$.

**Definition 5.6 ($\beta$-arrival model)** An arrival model of incoming messages to an AsPEFSM is called a $\beta$-arrival model if the arrival model satisfies the following conditions:

1. $a(n+1) - a(n) \ (n = 0, 1, 2, \ldots)$ are exponentially distributed with mean $\frac{1}{\lambda}$;

2. if $C(a(n))$ is class $ij \ (i, j \in Q)$, then $C(a(n+1))$ must be one of class $jk \ (k \in Q)$. The probability that $C(a(n+1))$ is class $jk$ is $p_{jk}$, and

$$\sum_{k \in Q} p_{jk} = 1.$$ 

The first condition in the above definition says the overall arrivals of incoming messages are a Poisson process with parameter $\lambda$. The second condition implies the sequence of the incoming messages corresponds to a sequence of consecutive transitions associated with the messages in the FSM of the AsPEFSM. The single-step transition probabilities of an AsPEFSM-\(\beta\), $p_{ij} \ (i, j \in Q)$, are actually given with the $\beta$-arrival model.

**Definition 5.7 (AsPEFSM-\(\beta\))** An AsPEFSM is called an AsPEFSM-\(\beta\) if its incoming messages follow a $\beta$-arrival model.
Model $A$

From the second condition of the definition of a $\beta$-arrival model, we know that a $\beta$-arrival model itself forms a stochastic process. This stochastic process is called Model $A$.

The kernel of Model $A$ of an AsPEFSM is the same as that of the AsPEFSM which is usually an FSM denoted as $M$. Graphically, Model $A$ has the same states and transitions as the FSM of the AsPEFSM, except that the contents and actions of the transitions are different. The single step transition probabilities of Model $A$ are given. The transition times in Model $A$ are the interarrival times of incoming messages which are identically distributed with mean $\frac{1}{\lambda}$. Each transition in Model $A$ sends out a message to the AsPEFSM.

Let $\pi = [\pi_i] \ (i \in Q)$ be the steady-state state probability vector of Model $A$. By Theorem A.1, $\pi$ can be computed from the given $P$. So we assume $\pi$ is known hereafter.

Let $A_{\bar{r}}_{ij}$ be the mean transition time of transition $ij$ in Model $A \ (i, j \in Q)$. Then,

$$A_{\bar{r}}_{ij} = \frac{1}{\lambda}.$$ 

Let $A_{\bar{e}}_i$ and $A_{\bar{q}}_{ii}$ be the mean entrance rate of state $i$ in Model $A$ and the recurrence time of state $i$ in Model $A$, respectively.

By Theorem A.5, we have

$$A_{\bar{q}}_{ii} = \frac{1}{A_{\bar{e}}_i} = \frac{A_{\bar{r}}}{\pi_i},$$

where

$$A_{\bar{r}} = \sum_{u, v \in Q} \pi_u P_{uv} A_{\bar{r}}_{uv} = \frac{1}{\lambda}.$$
Therefore,
\[ \lambda \theta_{ii} = \frac{1}{\lambda \pi_i} \]
and
\[ \lambda \epsilon_i = \lambda \pi_i. \]

### 5.3.2 Queueing and service model

We assume there is only one queue to store all the messages that arrive early. The queueing discipline is FIFO.

The queueing system for an AsPEFSM-\(\beta\) is shown in Figure 5.4.

![Queueing System for AsPEFSM-\(\beta\)](image)

**Figure 5.4:** The queueing system for an AsPEFSM-\(\beta\)

The order of processing the incoming messages is the same as the order of their arrivals. The trajectories of the states occupied by the AsPEFSM-\(\beta\), Models \(S\), \(W\) and \(E\) over time are identical to those of Model \(A\). Therefore, the transition probabilities of the AsPEFSM-\(\beta\), its Models \(S\), \(E\) and \(A\) are all the same.

### 5.3.3 Performance metrics

In this section, we investigate the performance metrics for AsPEFSM-\(\beta\)s.

Up to now, \(M\), \(P\) and \(H\) of an AsPEFSM-\(\beta\) are known. The mean arrival rate of incoming messages \(\lambda\) is also given. So similar to the performance evaluation
of AsPEFSM-α, we start with deriving $u^H$ of the AsPEFSM-β from the known information.

**Computation of $u^H$**

By the definition of transition time given in Chapter 3, we have

$$\psi_{x_{ij}} = u_{x_{ij}} + ^*_{x_{ij}}. \quad (5.30)$$

Let $\text{NoF}$ denote the event that the AsPEFSM-β finds no firable message of state $i$ in the queue when it enters state $i$ and $\overline{\text{NoF}}$ (i.e., at least one firable message is available) denote the reverse of $\text{NoF}$. Let $p_{0,i}$ be the probability of $\text{NoF}$.

From the definition of the queueing and service model of an AsPEFSM-β, we know that the wait time of an AsPEFSM-β at state $i$ is zero when the AsPEFSM enters a state and finds a firable message of the state. Formally,

$$Pr\{u_{x_i} = 0|X_n = i, X_{n+1} = j, \overline{\text{NoF}}\} = 1$$

or

$$Pr\{u_{x_i} > 0|X_n = i, X_{n+1} = j, \overline{\text{NoF}}\} = 0.$$

Since $u_{x_{ij}}(t)$ is the p.d.f.s of $u_{x_{ij}}$, we have

$$u_{x_{ij}}(0) = 1 - p_{0,i}.$$

When $t > 0$, we have

$$u_{x_{ij}}(t) = Pr\{u_{x_i} = t|X_n = i, X_{n+1} = j\}$$
\[ P_r = Pr\{w_r = t|X_n = i, X_{n+1} = j, NoF\} \cdot Pr\{NoF\} + \]
\[ Pr\{w_r = t|X_n = i, X_{n+1} = j, \overline{NoF}\} \cdot Pr\{\overline{NoF}\} \]
\[ = \frac{Pr\{(w_r = t, X_n = i, X_{n+1} = j)|\overline{NoF}\} \cdot p_{0,i}}{Pr\{(X_n = i, X_{n+1} = j)|\overline{NoF}\}} \cdot p_{0,i} \]
\[ = p_{0,i} \cdot Pr\{w_r = t|\overline{NoF}\} \]
\[ = p_{0,i} \cdot \lambda e^{-\lambda t} \]

where \( w_r \) is the wait time of the AsPEFSM-\( \beta \) for a firable message of state \( i \).

Therefore, we can derive
\[ w_{rij} = \int_0^\infty w_{rij}(t) \cdot t \cdot dt = \frac{p_{0,i}}{\lambda}, \]
and
\[ w_{rij} = w_{rij} + s_{rij} = \frac{p_{0,i}}{\lambda} + s_{rij}. \]

In the above equation, \( p_{0,i} \) is unknown. To compute \( p_{0,i} \) (\( i \in Q \)), we adopt the transformation introduced in Section 5.2.3 to construct virtual jobs with respect to state \( i \) of the AsPEFSM-\( \beta \). The definitions of first passage and virtual job are the same as before. The difference is that here the mean arrival rate of class \( ij \) virtual jobs in an AsPEFSM-\( \beta \) is not given. This rate can be computed as follows:

Let \( \lambda_{ij} \) be the mean arrival rate of class \( ij \) virtual jobs and \( \lambda_i \) be the mean arrival rate of all the virtual jobs with respect to state \( i \) in the AsPEFSM-\( \beta \).

From Model \( A \) of the AsPEFSM-\( \beta \) and the definition of virtual jobs, it is known that the arrival rate of class \( ij \) virtual jobs is equal to the arrival rate of the class \( ij \) incoming messages which is the entrance rate of transition \( ij \) in Model \( A \). Formally,
\[ \lambda_{ij} = \lambda \pi_i \varphi_{ij}, \]
and thus,
\[ \lambda_i = \Lambda \xi_i = \lambda \pi_i. \]

Moreover, when an AsPEFSM-$\beta$ is in equilibrium,
\[ \psi \xi_i = \Lambda \xi_i. \]

Therefore,
\[ \psi \theta_{ii} = \frac{1}{\psi \xi_i} = \frac{1}{\Lambda \xi_i} = \frac{1}{\lambda \pi_i}. \]

The rest of the work is to solve the equations stated in (5.8) and (5.19) for all \( i, j \in Q \), from which we obtain \( p_{0,i} \) and \( \psi \theta_{ji} \ (i, j \in Q) \).

With \( p_{0,i} \ (i \in Q) \) computed, \( \psi r_{ij} \ (i, j \in Q) \) can be derived and so can \( \psi \rho_{ij} \).

**Throughput**

Let \( \bar{\eta}_{ij} \) be the throughput rate of class \( ij \) outgoing messages \((i, j \in Q)\).

When an AsPEFSM is in equilibrium, the throughput rate is equal to the arrival rate. Therefore, the throughput rate of class \( ij \) outgoing messages is simply
\[ \bar{\eta}_{ij} = \lambda_{ij} = \lambda \cdot \pi_i \cdot p_{ij}. \]

**Utilization**

Let \( \rho \) be the utilization of the AsPEFSM-$\beta$.

By definition, we have
\[ \rho = \lambda \cdot \pi. \]
where $\lambda$ is the given mean arrival rate of incoming message and $\overline{\tau}$ is the mean service time of incoming messages. $\overline{\tau}$ can be computed as follows:

$$
\overline{\tau} = \sum_{i \in Q} \pi_i \cdot \overline{\tau}_i = \sum_{i \in Q} \sum_{j \in Q} \pi_i \cdot p_{ij} \cdot \overline{\tau}_{ij}.
$$

In this equation, $p_{ij}$, $\pi_i$ and $\overline{\tau}_{ij}$ ($i, j \in Q$) are already known.

Since $\rho < 1$ is the requirement for the AsPEFSM-$\beta$ to be in equilibrium, the inequality, $\lambda \overline{\tau} < 1$, must hold for the formula to work.

Queue wait times

By definition, the messages come to an AsPEFSM-$\beta$ in a Poisson pattern, and there is only one queue and one service center. As a result, the queueing system of an AsPEFSM-$\beta$ serving the incoming messages is an $M/G/1$ queueing system.

In this queueing system, the mean arrival rate is $\lambda$, and the mean service time of messages is $\overline{\tau}$. The second moment of the service times, $\overline{\tau^2}$, can be computed as follows:

$$
\overline{\tau^2} = \int_0^\infty (\sum_{i,j \in Q} \pi_i p_{ij} \overline{w_{ij}}(t)) t^2 dt \tag{5.31}
$$

$$
= \sum_{i,j \in Q} \pi_i p_{ij} (\int_0^\infty \overline{w_{ij}}(t) t^2 dt) \tag{5.32}
$$

$$
= \sum_{i,j \in Q} \pi_i \cdot p_{ij} \cdot \overline{\tau^2}_{ij}. \tag{5.33}
$$

The mean overall queue wait time of the incoming messages can be obtained using the solution technique for $M/G/1$ (see, for example [Allen90](p.688)):

$$
W_q = \frac{\lambda \overline{\tau^2}}{2(1 - \rho)}.
$$
By Little’s Law, the mean overall queue length is

\[ L = \lambda W_q = \frac{\lambda^2 \rho^2}{2(1 - \rho)}. \]

**Probability of the number of messages in the queueing system**

An AsPEFSM accepts and processes incoming messages according to its Model \( M \) sequentially. Model \( M \) is independent of the transition services of an AsPEFSM-\( \beta \), but Model \( M \) and its AsPEFSM-\( \beta \) share the same set of states. The state trajectory of the AsPEFSM-\( \beta \) always follows that of its Model \( M \).

Due to the service delay, the present state of an AsPEFSM-\( \beta \) is never ahead of that of its Model \( M \). Instead, the current state of an AsPEFSM-\( \beta \) is occasionally one or more states behind that of Model \( M \). For instance, when the AsPEFSM-\( \beta \) is in state \( j \), Model \( M \) may already be in state \( i \) and several incoming messages are buffered in the queue of the AsPEFSM-\( \beta \). The AsPEFSM-\( \beta \) enters state \( i \) when these messages in the queue have been processed.

A pair can be used to characterize this state relation of Model \( M \) and its AsPEFSM-\( \beta \) at any instant. We denote a pair for this purpose as \((i, j)\) \((i, j \in Q)\). The first element \( i \) of the pair represents the current state of Model \( M \) and the second element \( j \) represents the current state of the AsPEFSM-\( \beta \). If only the present state of Model \( M \) is of interest, \((i, \cdot)\) is used for indicating Model \( M \) is in state \( i \) at present and we do not care which state the AsPEFSM-\( \beta \) is in. Likewise, \((\cdot, j)\) states that the AsPEFSM-\( \beta \) is currently in state \( j \) and we do not care which state Model \( M \) is in.

Since we want to indicate the number of messages in the system together with this state relation, we add another element \( n \) to denote this number. This turns
the pair \((i, j)\) into a triple \((n, i, j)\). \((n, i, j)\) indicates that the system state of Model \(A\) is at state \(i\), the AsPEFSM-\(\beta\) is in state \(j\) and there are \(n\) messages in the system.

Suppose that at time \(t\), the system is in \((i, j, n)\). This state is reached from one of two possible states in one step:

1. At time \(t_1\) \((t_1 < t)\), the system state is \((n-1, i^- , j)\)\(^4\) if \(n > 0\). After a period of \(t_2\), an incoming message arrives but no message finishes its service yet. Then in the next \((t - t_1 - t_2)\) time period, no message arrives or finishes its service.

2. At time \(t_1\) \((t_1 < t)\), the system state is \((n+1, i, j^-)\). After a period of \(t_2\), a message finishes its service and leaves the queueing system. Then in the next \((t - t_1 - t_2)\) time period, no message arrives or finishes its service.

Let \(\psi_{g_{ij}}(\tau)\) be the probability that no message arrives or finishes its service within time \(\tau\) when the system is in state \((\cdot , i, j)\). We have

\[
\psi_{g_{ij}}(\tau) = (1 - \leq^{A_{h_i}}(\tau)) \cdot (1 - \leq^{\psi_{h_j}}(\tau))
\]

where \(\leq^{A_{h_i}}(\tau)\) and \(\leq^{\psi_{h_j}}(\tau)\) are the cumulative distribution functions of the state holding times of Model \(A\) and the AsPEFSM-\(\beta\), respectively.

Let \(\phi_{n, i, j}(t)\) be the probability that Model \(A\) is in state \(i\), and the AsPEFSM-\(\beta\) is in state \(j\) at time \(t\) with \(n\) messages in the queueing system.

Formally, \(\phi_{n, i, j}(t)\) can be expressed as

\[
\phi_{n, i, j}(t) = \sum_{i^- \in Q} \int_{t_2}^{t} \int_{t_1}^{t_2} \phi_{n-1, i^-, j}(t_1) \cdot p_{i^-, i} \cdot \leq^{A_{h_{i^-}}}(t_2) \cdot \psi_{g_{ij}}(t - t_1 - t_2) dt_2 dt_1
\]

\(^4\)\(i^-\) refers to a state from which there is a one-step transition to state \(i\) in the AsPEFSM-\(\beta\).
Chapter 5: performance evaluation of AsPEFSMs

\[ + \sum_{j \in \mathcal{E}} \int_0^{t_2} \int_0^{t_1} \phi_{n+1, i, j} (t_1) \cdot p_{j, -j} \cdot h_{j, j} (t_2) \cdot \psi_{ij}(t - t_1 - t_2) dt_2 dt_1. \]

Applying the exponential transformation [Howa71, Allen90] to the above equation, we get

\[ \phi_{n, i, j}^e (s) = \sum_{i' \in \mathcal{E}} \phi_{n-1, i', j}^e (s) \cdot p_{i', -i} \cdot A^e_{i', -i} (s) \cdot \psi_{ij}^e (s) \]
\[ + \sum_{j' \in \mathcal{E}} \phi_{n+1, i, j'}^e (s) \cdot p_{j, -j} \cdot h_{j', j'}^e (s) \cdot \psi_{ij}^e (s) \]
\[ = \left[ \sum_{i' \in \mathcal{E}} \phi_{n-1, i', j}^e (s) \cdot p_{i', -i} \cdot A^e_{i', -i} (s) \right] \]
\[ + \sum_{j' \in \mathcal{E}} \phi_{n+1, i, j'}^e (s) \cdot p_{j, -j} \cdot h_{j', j'}^e (s) \cdot \psi_{ij}^e (s). \]

Rewriting the above equation in Matrix form yields

\[ \Phi_n^e (s) = \left[ P \square A^e (s) \right]^T \Phi_{n-1}^e (s) + \Phi_{n+1}^e (s) \left[ P \square A^e (s) \right] \square G^e (s) \]

where \( \Phi_n^e (s) = [\phi_{n, i, j}^e (s)], \ A^e (s) = [h_{ij}^e (s)], \ H^e (s) = [\psi_{ij}^e (s)], \) and \( G^e (s) = [\psi_{ij}^e (s)]. \) \( \square \) is the matrix element multiplication\(^5\).

Dividing the elements of the matrices on both sides of Equation (5.3.3) with \( G^e (s), \) we get

\[ \Phi_n^e (s) \diamond G^e (s) = \left[ P \square A^e (s) \right]^T \Phi_{n-1}^e (s) + \Phi_{n+1}^e (s) \left[ P \square A^e (s) \right] \]

where \( \diamond \) is the matrix element division operator\(^6\).

We can rewrite the above equation as

\[ \Phi_{n+1}^e (s) = \left[ \Phi_n^e (s) \diamond G^e (s) - P \square A^e (s) \right]^T \times \Phi_{n-1}^e (s) \times \left[ P \square A^e (s) \right]^{-1} \]

---

\(^5 \ A \square B = [a_{ij} b_{ij}]. \) This matrix operation is different from the matrix multiplication.

\(^6 \ A \diamond B = [a_{ij} / b_{ij}]. \)
where \([A]^T\) is the transpose of the matrix \(A\) and \([A]^{-1}\) is the inverse matrix of \(A\).

This is a recurrence equation of the exponential transform of the probabilities of the number of messages in the queueing system at time \(t\). The limiting number of messages in the queueing system after a very long period of time, \(\lim_{t \to \infty} \phi_{n,i,j}(t)\), can be obtained from this equation by final value property of the exponential transformation [Howa71](p.697),

\[
\lim_{t \to \infty} \phi_{n,i,j}(t) = \lim_{s \to \infty} s \phi^\varepsilon_{n,i,j}(s).
\]

The initial values are known

\[
\phi_{0,0,0}(0) = 1;
\]

and

\[
\phi_{0,0,j}(t) = 0 \quad (i \neq j).
\]

Therefore,

\[
s \phi^\varepsilon_{0,0,j}(s) = 0 \quad (i \neq j)
\]

and

\[
\lim_{s \to 0} s \phi^\varepsilon_{0,0,j}(s) = 0 \quad (i \neq j).
\]

Moreover, by definition, we have

\[
\lim_{t \to \infty} \phi_{0,i,i}(t) = p_{0,i}.
\]

Therefore,

\[
\lim_{s \to 0} s \phi^\varepsilon_{0,i,i}(s) = p_{0,i}.
\]
The computations of $p_{0,i}$ ($i \in Q$) in the above equation have been presented earlier in this section. As a result, the initial matrix $\Phi_0(s)$ is known. Thereafter, the other parameters can be computed recursively.

Theoretically, the probability mass function of the number of messages in the queueing system can also be computed by finding the inverse function of the exponential transformation.

### 5.4 Bounds of Performance Measures

As long as the transition probabilities of an AsPEFSM are known, the bounds of the performance measures can be computed by analyzing its Model $S$. The computations of these bounds are the same as those in Section 4.3.1 and will not be repeated here.

### 5.5 Related work

Queueing systems with semi-Markov arrival processes or service processes have been attracting more and more attention recently.

Most of the published work focuses on studying the impact of the correlated or batch arrival jobs on a queue system. For instance, many papers examine the influence of Markov modulated arrivals on the performance of ATM networks. The results from these papers are limited and are applicable only to some simple queueing models in which the assumptions to simplify the service times are satisfied. For example, [Herr93] studies SMP/D/1$^7$ for ATM networks. SMP/D/1 is a

---

$^7$SMP is the abbreviation of *semi-Markov process.*
more general model than the models given in [DiDe90, Ding91]. The arrivals of an
SMP/D/1 are correlated input streams and are assumed to be asynchronous, be-
ing controlled by an SMP. This is similar to the queueing systems of AsPEFSM-βs
in this chapter. However, the service times of an SMP/D/1 are assumed constant
and there is no distinction of job classes. In contrast, we have fewer restrictions
on the service times which are continuous random variables.

[DsTa93] studies a more complicated queueing system than SMP/D/1. In the
queueing system of that work, the input flows are modulated by SMPs and the
service times depend on the states of the systems and the batch sizes of input
data taken from the buffer. The queueing process of such a queueing system is the
main object to be studied. The theories of point processes and semi-regenerative
processes were employed in this work to derive the steady-state distribution of
the queueing process, the mean values of idle and busy periods, etc.. A state of
the queueing process represents the size of the unserviced data (i.e., the number
of the customers) in the system.

Unlike the previous work stated above, queueing processes are not the main
processes to be studied in our performance evaluation of AsPEFSMs. We focus
mainly on the state changing process of the FSM of an AsPEFSM. The incoming
messages in the queues are distinguished on their classes. In this way, we not
only obtain the queueing stochastics but also analyze the relationships between
the queueing processes and the states of the FSM of an AsPEFSM.
5.6 Summary

In this chapter, we have analyzed AsPEFSMs with two types of asynchronous arrival models: AsPEFSM-αs and AsPEFSM-βs. In an AsPEFSM-α, incoming messages arrive independently and are uncorrelated; in an AsPEFSM-β, incoming messages come correlated in an order which can be accepted by the FSM of the AsPEFSM-β. In both cases, the transition service times $\tau_{ij} (i, j \in Q)$ are assumed to be independent of each other and independent of message arrival patterns. The service times may assume any distribution.

The methods to compute transition probabilities and transition wait times for AsPEFSM-αs and AsPEFSM-βs are given. Moreover, the derivations of the performance metrics such as utilization, throughputs and queue wait times are presented.

From a queueing theory point of view, the queueing systems of AsPEFSMs are interesting queueing systems. The services of these queueing systems are state-dependent (or more precisely, transition dependent). The arrivals of jobs to these queueing systems are probably correlated. This kind of queueing system is more realistic in computer networks.

In addition, the queueing properties of an M/G/1 with multiple job classes and an AsPEFSM-α have been discovered and proved.
Chapter 6

Performance Evaluation of Hybrid PEFSMs

Abstract

This chapter studies the third class of PEFSMs called hybrid PEFSMs (HyPEFSMs). HyPEFSMs are the most complex of all PEFSMs. We examine a specific type of HyPEFSM and derive the performance metrics such as utilization, throughput and queue wait time for the HyPEFSMs of this type.

6.1 Introduction

In the previous two chapters, we have studied the PEFSMs whose arrival models of incoming messages are either completely synchronous or completely asynchronous, respectively. In this chapter, we will study another type of PEFSM in which the incoming messages may come synchronously or asynchronously. A PEFSM of this type is called a hybrid PEFSM (HyPEFSM).
6.1.1 Hybrid transition

Recall the alternating bit protocol in Chapter 3. The FSM of the transmitter of this protocol was shown in Figure 3.1. In Chapter 3, that FSM is used as an example for a HyPEFSM. We use the same FSM for a transmitter here but with different assumptions about the running environment of the transmitter. For convenience of discussion, we repeat the figure in Figure 6.1.

![FSM diagram]

Figure 6.1: An FSM of the alternating bit protocol

We now assume that the underlying communication network may delay an ACK message. The delayed ACK may still arrive at the transmitter after a timeout but it is not discarded by the transmitter if the transmitter is in state 1. A delayed ACK usually arrives asynchronously while a non-delayed ACK arrives synchronously. Therefore, in this case, transition T10 cannot simply be classified as a synchronous transition or an asynchronous transition\(^1\). This is because the incoming messages of transition T10 consist of both synchronous and asynchronous ACK messages.

\(^1\)We have defined synchronous transition and asynchronous transition in Chapter 3.
A transition, which is associated with both synchronous and asynchronous incoming messages, is called a *hybrid transition*.

We have seen from the alternating bit protocol example that in a PEFSM, whether a transition is hybrid or not depends on the assumptions about the protocol that the FSM specifies. If we assume that the timeout values are set long enough so that no delayed ACK message will come after a timeout, then transition T10 is solely a synchronous transition.

In this chapter, we study a HyPEFSM which has only synchronous and asynchronous transitions.

### 6.1.2 Hybrid state

In a SyPEFSM or an AsPEFSM, by definition, all the transitions from a state are completely synchronous or asynchronous, respectively. If all the transitions from a state of a PEFSM are synchronous, we call this state a *synchronous state*. Likewise, if all the transitions from a state are asynchronous, we call this state an *asynchronous state*. Therefore, all states of a SyPEFSM are synchronous, and all states of an AsPEFSM are asynchronous.

However, both types of states may co-exist in a HyPEFSM. This is because both synchronous and asynchronous transitions are allowed in a HyPEFSM. In addition, a third type of state is possible in a HyPEFSM. From a state of this type, some of the transitions are synchronous and some are asynchronous. We call this state a *hybrid state*. Figure 6.2 shows the three possible situations of a state from which there are two transitions in a HyPEFSM.

A hybrid state makes it difficult to assume or determine the single-step proba-
abilities for the transitions starting from that state. For instance, in Figure 6.2 (c), transition $ij$ is synchronous while transition $ik$ is asynchronous. Class $ij$ incoming messages are associated with transition $ij$ and class $ik$ messages associated with transition $ik$. Suppose class $ij$ messages are timeouts and class $ik$ messages are data packets with acknowledgement information. Usually, the occurrence of class $ij$ messages depends on the availability of class $ik$ messages when a HyPEFSM enters state $i$. For this reason, the single-step probability of transition $ij$ depends on the probability that there is no class $ik$ firable message when the HyPEFSM enters state $i$. This latter probability is related to the previous occupancies of the states and the previous arrivals of class $ik$ firable messages. In this sense, with a hybrid state, the transition probabilities of a HyPEFSM depend on the past state occupancies. As a result, the state changing process of the HyPEFSM may not be a continuous-time semi-Markov process (CSMP). The theory for non-CSMPs which is much more complex than that of CSMPs should be used to analyze the state changing process of the HyPEFSM. Unfortunately, few results on non-CSMP have been published. Therefore, we are not going to study the HyPEFSMs with any hybrid state in this chapter.
In the following, a class of HyPEFSMs which have only *synchronous* and *asynchronous* states is examined. A wide range of communication protocols can be modelled using this class of HyPEFMS. This is because, in general, two types of data may come to a protocol in the ISO protocol stack model. One type of data is from the protocol user, the underlying system or a communication participant as requests. The arrivals of these requests can usually be modelled as asynchronous arrivals. The other type is from a communication participant as responses. The arrivals of responses can usually be modelled as synchronous arrivals.

We assume that the arrival information of a HyPEFSMs is given. Furthermore, similar to the AsPEFSM, the early asynchronous messages to a HyPEFSM are assumed to wait in queues. The performance metrics such as *throughputs*, *utilization* and *queue wait time* are derived for this class of HyPEFSMs in this chapter.

The rest of this chapter is organized as follows. Section 6.2 describes the message arrival model, the queuing model and the service model. The derivation of the performance metrics is also given in this section. Section 6.3 shows the application of the performance evaluation method to the alternating bit protocol and compares the analytical result with the simulation result. Section 6.4 summarizes this chapter and provides some suggestion on the applicability of the solution techniques given in this chapter to a HyPEFSM with hybrid states.
6.2 Models and Evaluation

Consider a HyPEFSM, denoted as $\Psi$, which has only two types of states: *synchronous* and *asynchronous*.

Let $(M, P)$ and $P = (P, H, H)$ be the components of $\Psi$. The semantics of these components are defined in Chapter 3.

6.2.1 Message arrival model

By definition, incoming messages arrive synchronously or asynchronously to $\Psi$.

The arrival models of synchronous messages in $\Psi$ are similar to those in SyPEFSMs in Chapter 4. Suppose that state $i$ of $\Psi$ is a synchronous state, and class $ij$ firable messages of state $i$ ($j \in Q$) are synchronous. Only one firable message of state $i$ will come each time $\Psi$ enters state $i$. We assume the probability that the next incoming firable message is of class $ij$ is $p_{ij}$ ($j \in Q$), and the p.d.f. of the wait time for this message is $w_{ij}(t)$. Both $p_{ij}$ and $w_{ij}(t)$ ($j \in Q$) are given by the performance evaluator, and

$$\sum_{j \in Q} p_{ij} = 1.$$ 

The arrivals of asynchronous messages for each class in $\Psi$ are assumed to be a Poisson process. Suppose that state $k$ is an asynchronous state, and class $kl$ is a class of firable messages of state $k$ associated with transition $kl$ ($l \in Q$). The arrival pattern of class $kl$ messages is a Poisson process with mean $\lambda_{kl}$. This model is the same as the $\alpha$-arrival model of an AsPEFSM-$\alpha$. 
6.2.2 Queuing and service model

We assume there is a queue for each class of asynchronous incoming messages in $\Psi$. The queuing discipline of each queue is first in first out. The service discipline at an asynchronous state is first come first served.

When $\Psi$ enters a state, if the state is a synchronous state, $\Psi$ will wait for a firable message of the state; if the state is an asynchronous state and $\Psi$ will first check the queues to see whether there is a firable message. If there is none, $\Psi$ will wait for the first firable message of this state to arrive; otherwise, it will select the earliest arrived firable message from a queue and process it.

Therefore, the service model of $\Psi$ in a synchronous state is the same as the one for a SyPEFSM. The queuing and service model of $\Psi$ in an asynchronous state is the same as the one for an AsPEFSM-$\alpha$.

Similarly, we can compute the transition probability of transition $ij$ for asynchronous state $i$ as:

$$p_{ij} = \frac{\lambda_{ij}}{\sum_{j \in Q} \lambda_{ij}}.$$ 

With this queuing and service model, the transition probability matrix $P$ of $\Psi$ can be determined completely and directly from the given arrival model. So we assume $P$ is known hereafter.

6.2.3 Performance metrics

With the arrival model and the queuing and service model stated above, the primitive data $M$, $P$, $H$ of $\Psi$ are known. $w_{ij}$ is also known if transition $ij$ is synchronous. So in the following, we first compute the transition wait times for the asynchronous transitions and then show the derivation of the performance
metrics – throughput, utilization and queue wait time.

Computation of \( \psi_H \)

Let \( \pi = [\pi_i] \) be the steady-state state probability vector of \( \Psi (i \in Q) \). By Theorem A.1, \( \pi \) can be computed from the given \( P \). So we assume \( \pi \) is known hereafter.

From the definition of transition time in Chapter 3, we have

\[
\psi_{\tau_{ij}} = w_{\tau_{ij}} + s_{\tau_{ij}}.
\] (6.1)

Therefore,

\[
\psi_{\tau_{ij}} = w_{\tau_{ij}} + s_{\tau_{ij}}.
\] (6.2)

If transition \( ij \) is synchronous, \( w_{\tau_{ij}} \) is already given in the arrival model; otherwise, \( w_{\tau_{ij}} \) is unknown and must be computed.

Suppose that state \( i \) is asynchronous. Let \( NoF \) denote the event that \( \Psi \) finds no firable message of state \( i \) in the queues when it enters state \( i \) and \( \overline{NoF} \) denote the reverse of \( NoF \) (i.e., there is firable message(s)).

Let \( p_{0,i} \) be the probability of the event \( NoF \).

Following similar derivation of \( w_{h_{ij}}(t) \) for an AsPEFSM-\( \alpha \) presented in Chapter 5, we can also get the following result for \( \Psi \)

\[
w_{h_{ij}}(t) = \begin{cases} 
(1 - p_{0,i}) & \text{for } t = 0 \\
p_{0,i} \cdot \lambda_i e^{-\lambda_it} & \text{for } t > 0 
\end{cases}
\] (6.3)

where

\[
\lambda_i = \sum_{k \in Q} \lambda_{ik}
\] (6.4)
and $\lambda_{ik}$ is the mean arrival rates of class $ik$ incoming messages ($k \in Q$).

Since $\nu h_{ij}(t)$ is the p.d.f. of $\nu r_{ij}$, therefore

$$\nu r_{ij} = \int_0^\infty t \cdot \nu h_{ij}(t) \cdot dt = p_{0,i} \cdot \frac{1}{\lambda_i}, \quad (6.5)$$

and

$$\psi r_{ij} = p_{0,i} \cdot \frac{1}{\lambda_i} + \sigma r_{ij}. \quad (6.6)$$

In the above equation, $p_{0,i}$ is unknown. To compute $p_{0,i}$ ($i \in Q$), we adopt the transformation introduced in Section 5.2.3 to construct virtual jobs with respect to state $i$ of $\Psi$. The definitions of first passage, virtual job and virtual job class in $\Psi$ are the same as those for an AsPEFSM-$\alpha$. As a result, we obtain the following equation that is similar to Equation (5.19):

$$p_{0,i} = 1 - \left(\sum_{j \in Q} \lambda_{ij} (\sigma r_{ij} + \psi \theta_{ji}) - \lambda_{ii} \psi \theta_{ii}\right). \quad (6.7)$$

Similar to an AsPEFSM-$\alpha$, when $\Psi$ is in equilibrium, the mean recurrence time of an asynchronous state, say state $i$, has the following property:

$$\psi \theta_{ii} = \frac{1}{\lambda_i}. \quad (6.8)$$

However, the mean recurrence time of a synchronous state in $\Psi$, say state $k$, still needs to be computed by using Theorem A.5:

$$\psi \theta_{ii} = \frac{\psi \tau}{\pi_i} \quad (6.9)$$

where

$$\psi \tau = \sum_{u,v \in Q} \pi_u p_{uv} \psi \tau_{uv}.$$
\( \psi_T \) is the overall mean transition time in \( \Psi \). \( \psi_{uv} \) is the transition time of transition \( uv \) which can be rewritten using either Equation (6.2) or (6.6) depending on whether transition \( uv \) is synchronous or asynchronous, i.e.,

\[
\psi_{ij} = \begin{cases} 
\frac{w_{ij}}{p_{0,i}} + \frac{a_{ij}}{\lambda_i} & \text{if transition } ij \text{ is synchronous} \\
\frac{1}{\lambda_i} & \text{if transition } ij \text{ is asynchronous} 
\end{cases} \tag{6.10}
\]

The formula for \( \psi_{ii} \) is also different for a synchronous state and an asynchronous state. This is an essential difference between a HyPEFSM and an AsPEFSM-\( \alpha \) in the derivations of performance metrics.

The mean first passage time from state \( j \) to \( i \) (\( i, j \in Q \)) in Equation (6.7), \( \psi_{ji} \), can be computed using Theorem A.3:

\[
\psi_{ji} = \psi_{i} + \sum_{k \in Q} p_{jk} \psi_{ki} - p_{ji} \psi_{ii} \tag{6.11}
\]

\[
= \sum_{k \in Q} p_{jk} \psi_{jk} + \sum_{k \in Q} p_{jk} \psi_{ki} - p_{ji} \psi_{ii} \tag{6.12}
\]

where \( \psi_{ki} \) can be rewritten using (6.2) or (6.6) depending on whether transition \( ki \) is synchronous or asynchronous, and \( \psi_{ii} \) can be rewritten using (6.9) or (6.9) also depending on whether state \( i \) is synchronous or asynchronous.

Using equations (6.7) and (6.12) \((i, j \in Q)\), we can solve \( p_{0,i} \) and \( \psi_{ji} \), and furthermore obtain \( w_{ij} \) of asynchronous transition \( ij \) \((i, j \in Q)\) and \( \psi_{ii} \) of synchronous state \( i \) \((i \in Q)\).

**Throughputs**

Given \( \psi_{ii} \) – the mean recurrence time of state \( i \) of a HyPEFSM \((i \in Q)\), one is able to compute the mean throughput rate of any specific class of outgoing messages associated with a transition in \( \Psi \).
Let $\bar{\eta}_{ij}$ be the throughput rate of the outgoing messages associated with transition $ij$.

When a system is in equilibrium, it is true that mean throughput rate = mean input rate. Class $ij$ outgoing messages are associated with transition $ij$. For that reason, the throughput rate of class $ij$ messages is equal to the entrance rate of transition $ij$, i.e.,

$$\bar{\eta}_{ij} = \psi_e i_p_{ij}$$

$$= \frac{1}{\psi\theta_{ii}} p_{ij}$$

where

$$\psi\theta_{ii} = \begin{cases} \frac{\psi_r}{\pi_i} & \text{if state } i \text{ is synchronous} \\ \frac{1}{\lambda_i} & \text{if state } i \text{ is asynchronous} \end{cases}$$

$\psi$ is computed earlier.

Utilization

Let $\rho$ be the utilization of a HyPEFSM. $\rho$ is the ratio of the service time and the transition time. We can use the following equation to compute $\rho$ when $\Psi$ is in equilibrium:

$$\rho = \frac{\psi_r}{\psi_f}.$$  

Queue wait times

In $\Psi$, there is no queue wait time for a synchronous message. However, an asynchronous message may have to wait in a queue if it arrives early. So here we only have to compute the queue wait times for asynchronous messages.
Since asynchronous messages of each class arrive in a Poisson pattern, we can draw a conclusion similar to Theorem 5.5. That is, the mean queue wait times of each class of asynchronous firable messages for an asynchronous state in $\Psi$ are the same. As a result, we only need to compute the overall mean queue wait time of firable messages for each asynchronous state of $\Psi$.

This computation of the queue wait time is similar to what we did for the AsPEFSM-$\alpha$. For each asynchronous state of $\Psi$, we construct the virtual jobs as before: a virtual job consists of the consecutive transitions on a first passage starting from the asynchronous state and ending at the same state. This approach then transforms the queue system of $\Psi$ into an $M/G/1$. The mean arrival rate of the virtual jobs of an asynchronous state, say state $i$, is $\lambda_i$. $\lambda_i$ is computed by applying Equation (6.4). The solution techniques for $M/G/1$ (see, for example, Equation (5.29)) are applied to this $M/G/1$ to obtain the mean queue wait time of the asynchronous message of the state.

### 6.3 An Application Example

In this section, we apply the performance evaluation method to the alternating bit protocol introduced at the beginning of this chapter (see Figure 6.1).

The workload and system parameters are assumed as follows:

The data packets arrive in a Poisson pattern at a mean rate of 50 packets/second. The transmission (or retransmission) of each data packet takes 0.001 second. The distribution of wait time for ACK is between 0 and 0.010 second with a mean of 0.001 second. We assume that if a timeout occurs for a data packet, no delayed acknowledge for that packet will arrive. The processing time
of ACK is assumed to be negligible. The loss probability of data or ACK is 0.1. The mean timeout value is set at 0.010 second.

The above information is translated into input data required by the two state HyPEFSM:

\[
P = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0.9 & 0.1 \end{pmatrix}
\]

and

\[
\begin{align*}
\lambda_{00} &= 0; & \lambda_{01} &= 50; \\
\sigma_{00} &= 0; & \sigma_{01} &= 0.001; \\
\mu_{10} &= 0.001; & \mu_{11} &= 0.010; \\
\xi_{10} &= 0; & \xi_{11} &= 0.001.
\end{align*}
\]

By solving Equation (A.1) with \( P \), we obtain the steady state probabilities of the states in the HyPEFSM

\[
\pi = (\pi_0, \pi_1) = \left( \frac{9}{19}, \frac{10}{19} \right).
\]

The mean state holding time of the HyPEFSM in state 1 is computed as

\[
\psi_{\tau_1} = \sum_{i=0}^{1} p_{1i} \psi_{\eta_{1i}} = \sum_{i=0}^{1} p_{1i}(\mu_{1i} + \sigma_{1i}) = 0.0020.
\]

We can construct the equations for the first passage times between two states by applying Theorem A.3:

\[
\begin{align*}
\psi_{\theta_{00}} &= \frac{\lambda_0}{\lambda_0 + \lambda_{01}}, \\
\psi_{\theta_{11}} &= \frac{\pi_1}{\pi_0 \sigma_{00} + \pi_1 \sigma_{11}}, \\
\psi_{\theta_{01}} &= \psi_{\theta_{10}} + (p_{00} \psi_{\theta_{00}} + p_{01} \psi_{\theta_{11}}) - p_{01} \psi_{\theta_{11}} = \psi_{\theta_{00}}; \\
\psi_{\theta_{10}} &= \psi_{\theta_{11}} + (p_{10} \psi_{\theta_{00}} + p_{11} \psi_{\theta_{10}}) - p_{10} \psi_{\theta_{00}} = \psi_{\theta_{11}} + p_{11} \psi_{\theta_{10}}.
\end{align*}
\]

From the above equations we can directly obtain

\[
\begin{align*}
\psi_{\theta_{00}} &= 0.0200 \text{ (second), and} \\
\psi_{\theta_{10}} &= 0.0022 \text{ (second).}
\end{align*}
\]
Let $p_{0,0}$ be the probability that a data packet arrives and sees no other data packet being served or waiting for service. Applying Equation (6.7), we have

$$p_{0,0} = 1 - \lambda_{01}(\tau_{01} + \psi_{10}) = 0.84.$$  

This means the probability that a data packet will be served immediately on arrival is 84%.

Therefore, we can further derive the mean wait time of transition 01

$$\mu_{r_{01}} = \frac{p_{0,0}}{50} = 0.0168 \text{ (second)}.$$  

It means the average time that the system is idle waiting for a data packet is 0.0168 second.

With $\mu_{r_{01}}$, we obtain

$$\psi_{01} = 0.0178 \text{ (second)}, \text{ and}$$

$$\psi_{11} = 0.0180 \text{ (second)}.$$  

To compute the mean queue wait time of the data packets, we construct the virtual jobs which consist of the transitions on the first passages from state 0 to state 0. The second moment of the service time of the virtual jobs is computed as using Equation (B.1)

$$\bar{z}_{0}^2 = \bar{z}_{01}^2$$

$$= \bar{z}_{01}^2 + \psi_{10}^2 + 2 \cdot \tau_{01} \cdot \psi_{10}$$

$$= 0.0000373.$$  

Then, the mean queue wait time is computed using Equation (5.29)

$$\bar{W}_{q,0} = \frac{\lambda_{0} \bar{z}_{0}^2}{2(1 - \rho_{0})} = 0.0011.$$
where $\rho_0 = 1 - p_{0,0} = 0.16$.

A simulation run was also conducted to obtain the performance under the same workload given above. The simulation results confirm the analytical results (see Table 6.1).

<table>
<thead>
<tr>
<th>performance metric</th>
<th>analytical result</th>
<th>simulation result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\psi_{00}$</td>
<td>0.0200</td>
<td>0.0199</td>
</tr>
<tr>
<td>$\psi_{01}$</td>
<td>0.0178</td>
<td>0.0177</td>
</tr>
<tr>
<td>$\psi_{10}$</td>
<td>0.0022</td>
<td>0.0022</td>
</tr>
<tr>
<td>$\psi_{11}$</td>
<td>0.0180</td>
<td>0.0179</td>
</tr>
<tr>
<td>$p_{0,0}$</td>
<td>0.8400</td>
<td>0.8404</td>
</tr>
<tr>
<td>system utilization*</td>
<td>0.0556</td>
<td>0.0557</td>
</tr>
<tr>
<td>mean queue wait time</td>
<td>0.0011</td>
<td>0.0012</td>
</tr>
</tbody>
</table>

*See Equation 6.16.

Table 6.1: Comparison of analytical result and simulation result

Let $\lambda_0$ be a variable while the other input data remain the same. We can analyze the relation between the mean queue wait time of the data packets and their arrival rates. Figures 6.3 and 6.4 show such a relation. It can be seen intuitively from Figure 6.4 that the mean queue wait time grows rapidly when the mean arrival rate is larger than 250 packets/second.

We can also let the packet loss probability (i.e., $p_{11}$) be a variable and analyze the relationship of the mean queue wait time as a function of the packet loss probability. Figure 6.5 illustrates that the growth of the mean queue wait time increases rapidly when the loss probability is larger than 50%.
Figure 6.3: Mean queue wait time vs. mean data arrival rate

Figure 6.4: Mean queue wait time vs. mean data arrival rate

6.4 Summary and Discussion

We have given the derivations of throughput, utilization and queue wait time for a HyPEFSM with two types of states: synchronous and asynchronous. The so-
Resolution technique for performance is basically a combination of the techniques for AsPEFSM-α and SyPEFSM. The assumptions used in HyPEFSMs comprise of the assumptions of AsPEFSM-α for asynchronous transitions and the assumptions of SyPEFSM for synchronous transitions. The HyPEFSM is more realistic in practice than either the AsPEFSM or the SyPEFSM alone.

The performance evaluation of the alternating bit protocol is shown as an application example of the HyPEFM. A more complex example describing the performance analysis of a class of multi-stage interconnection networks is given in Appendix E.

We have also briefly discussed that the main difficulty in solving a general HyPEFSM is in the treatment of hybrid states. With a hybrid state, the state changing process of a HyPEFSM is not a CSMP. A more general theory of stochastic processes needs to be employed and the computation will probably be much
more complex.

In spite of this, a hybrid state of a HyPEFSM can be modelled as a combination of synchronous and asynchronous states if we know the probability that synchronous transitions from the state are selected (or the probability that asynchronous transitions are selected). This is shown in Figure 6.6.

\begin{figure}[h]
\centering
\begin{tikzpicture}
  \node (state1) [circle, draw] {i};
  \node (state2) [circle, draw, below of=state1] {i'};
  \node (state3) [circle, draw, below of=state1] {i''};
  \draw[->] (state1) -- node[above] {synchronous transitions with the probability $p$} (state2);
  \draw[->] (state1) -- node[below] {asynchronous transitions with the probability $(1-p)$} (state3);
  \draw[->] (state2) -- node[above] {synchronous transitions} (state1);
  \draw[->] (state3) -- node[below] {asynchronous transitions} (state1);
  \draw[->] (state1) -- node[below] {$p$} (state2);
  \draw[->] (state1) -- node[above] {$(1-p)$} (state3);
\end{tikzpicture}
\caption{Modelling of a hybrid state}
\end{figure}

State $i$ is originally a hybrid state. The probability of synchronous transitions from state $i$ is known to be $p$. In the equivalent model, state $i$ becomes a synchronous state. Two new states $i'$ and $i''$ are added, and they are synchronous and asynchronous, respectively. We also add two new transitions: the one from $i$ to $i'$ is with probability $p$ and its transition time is 0; the other is from $i$ to $i''$ with probability $(1 - p)$ and its transition time is also 0. All the synchronous
transitions from state $i$ are moved to state $i'$; and all the asynchronous transitions are moved to state $i''$.

After each hybrid state of a HyPEFSM has been modelled in this way, the HyPEFSM has only two types of states: synchronous and asynchronous. The solution techniques presented in Section 6.2 can thus be applied to this HyPEFSM to obtain the performance measures.

The probability $p$ can be obtained through measurement or simulation. This approach combines analytical method with simulation result to evaluate the performance of a complex system. This concept has been used by other researchers. For example, in [KrWh93], many performance parameters such as the single-step transition probability matrix $P$ and transition times are obtained through simulation based on a formal specification.
Chapter 7
Analysis of Performance Bottlenecks using PEFSMs

Abstract

This chapter studies the problem of identifying performance bottlenecks in communication protocols. The identification method is based on the performance model PEFSM which is defined earlier in Chapter 3. Informally, a bottleneck with respect to a performance metric is defined as the transition among all the transitions in a PEFSM which would produce the largest marginal improvement of the performance metric if the time of the transitions were reduced by the same small amount. We present two methods to locate the bottleneck transitions with respect to two of the most important performance metrics: throughput and queue wait time. These methods are partially validated using simulation results.
7.1 Introduction

In the previous three chapters, we have presented techniques to compute performance measures based on PEFSMs from given arrival information. In this chapter, we are interested in how to improve system performance in an efficient way. The key step in solving this problem is to identify the performance bottleneck.

Performance bottlenecks exist in almost all computer systems in various forms. System designers, analyzers and users have worked on detection of the performance bottlenecks in a computer system for a long time. A bottleneck can be a service center of a system [Leung88, Allen90] or, at a more abstract level, a system parameter. For instance, in [ZiEt92], the sensitivities of the parameters in the throughput expression are used to determine the throughput bottleneck. There exist many definitions of performance bottlenecks and most of them are defined with respect to only throughput or utilization [Ferr78, Lock84, Leung88, Yang89, Allen90, ZiEt92].

Nevertheless, all definitions of performance bottleneck have a common characteristic: a bottleneck is referred to the component in the system which has the most significant impact on system performance. A small improvement to the bottleneck component can greatly improve system response time, throughput or utilization.

This chapter is concerned with finding performance bottlenecks in communication protocols. We have noted that it is common to specify communication protocols as interacting FSMs or EFSMs which are FSMs extended with variables. Many standardized protocols are directly given as FSMs or EFSMs. Examples
can be found in [Tane88, ISO2576, ISO7776]. At least two internationally stan-
dardized formal description techniques exist (Estelle [ISO8807] and SDL [SDL])
which provide a way to specify protocols and distributed systems based on FSMs
or EFSMs. Therefore, it is reasonable to use a PEFSM which is defined based on
EFSM (Chapter 3) as the underlying model for detection of performance bottle-
necks. PEFSMs have been shown in the previous chapters as effective models for
performance analysis.

In the FSM of a PEFSM, *state* and *transition* are the two main constructs.
*States* are conceptual while *transitions* have direct correspondence in the imple-
mentation of the protocol specified by the FSM. The time of a transition directly
affects performance. Therefore, it is natural to transform the bottleneck detection
problem to the one of identifying the *bottleneck transition* in a PEFSM. This
is useful because we then know where we should focus our efforts to improve
performance.

The execution of a transition takes time which in turn may delay the process-
ing of the following incoming messages of a PEFSM. The following messages may
be associated with other transitions in the PEFSM. Reducing the transition time
in a PEFSM will improve the performance of the PEFSM. For example, reducing
a transition time will increase the throughputs of each class of outgoing messages
since the recurrence times of each state are decreased. However, the degree of
improvement to a performance metric depends on selected transitions. The one
which can cause the most improvement to a performance metric is identified as
the *bottleneck* of this performance metric.

We use *weight* to indicate the relation between the reduction of the time of a
transition and the improvement of a performance metric. A *weight* with respect
to a performance metric is computed for each transition in a PEFSM. The higher the weight of a transition with respect to the same performance metric, the more the performance metric can be improved by reducing the same amount of time of this transition. As such, the transition with the greatest weight with respect to a performance metric is the bottleneck transition with respect to the performance metric. For instance, if the weight of transition $ij$ in a PEFSM with respect to the mean queue wait time of a message class is greater than that of any other transition, then transition $ij$ is the bottleneck transition with respect to the mean queue wait time. In other words, if each transition time is independently reduced by the same amount, the mean queue wait time will decrease the most in the case of a reduction in transition $ij$.

In this chapter, we focus on two of the most important performance metrics of a PEFSM. They are the throughput rate of a class of outgoing messages and the mean queue wait time of a class of incoming messages. The methods to compute the weights of transitions with respect to each performance metric are presented in this chapter.

The first method is to use partial derivative. In general, if a performance metric $K$ can be expressed as a function of a set of parameters, $t_1, t_2, \ldots, t_n$,

$$K = F(t_1, t_2, \ldots, t_n),$$

then the partial derivatives of $K$ with respect to $t_1, t_2, \ldots, t_n$,

$$\frac{\partial K}{\partial t_1}, \frac{\partial K}{\partial t_2}, \ldots, \frac{\partial K}{\partial t_n},$$

indicate the relative impacts of the changes of each parameter on the performance metric. The largest derivative, say $\frac{\partial K}{\partial t_k} (1 \leq k \leq n)$, indicates the corresponding
parameter, $t_k$, has the greatest impact on the change of $K$, and thus $t_k$ is the bottleneck of $K$. Therefore, the partial derivatives can be used as the weights of the parameters. In our case, we compute the partial derivatives of the throughput of outgoing messages of a specific class with respect to each transition time. These derivatives are taken as the weights of each transition with respect to the throughput.

The second method is to use an approximate approach to computing the weights with respect to the mean queue wait time of a specific incoming message class. This is because in the second case, it is difficult to compute partial derivatives.

We again assume that the transition probability matrix, $P$, and the steady-state state probability vector, $\pi$, are known. They can be obtained using the technique described in the previous three chapters.

The rest of the chapter is organized as follows. Section 7.2 presents a method to locate the bottleneck transition of the throughput rate of a class of outgoing messages. Section 7.3 presents a different method to compute the weights for each transition with respect to the mean queue wait time of a class of incoming messages. The method is partially validated with a simulation. Section 7.4 discusses related work on defining and locating performance bottlenecks, and Section 7.5 summarizes this chapter.
7.2 Throughput Bottleneck

In a PEFSM, an outgoing message of a class is generated when the transition associated with the class is processed. Therefore, the throughput rate of the outgoing messages of a class is equal to the entrance rate of the transition associated with the class.

Let $\bar{e}_i$ and $\bar{e}_{ij}$ be the mean entrance rates of state $i$ and transition $ij$ of a PEFSM, respectively. From the FSM of the PEFSM, we have

$$\bar{e}_{ij} = \bar{e}_i \cdot p_{ij}$$

where $p_{ij}$ is the probability of transition $ij$.

By Theorem A.4, one can infer

$$\bar{e}_{ij} = \frac{\pi_i}{\psi_{ij}} \cdot p_{ij} = \frac{\pi_i P_{ij}}{\sum_{u,v \in Q} \pi_u p_{uv} \psi_{uv}}.$$

Let $\bar{\eta}_{ij}$ be the throughput rate of class $ij$ outgoing messages. Then,

$$\bar{\eta}_{ij} = \bar{e}_{ij} \cdot \frac{\pi_i P_{ij}}{\sum_{u,v \in Q} \pi_u p_{uv} \psi_{uv}}.$$

The above equation illustrates the relationship of the throughput rate $\bar{\eta}_{ij}$ and the transition times $\psi_{uv} (u, v \in Q)$. For a given PEFSM, $\pi_i p_{ij} (i, j \in Q)$ is fixed. So the change of $\bar{\eta}_{ij}$ relies on the denominator of the above equation.

Using the derivatives, one can find that the coefficient $\pi_u p_{uv}$ of $\psi_{uv}$ in the denominator indicates the relative degree of the improvement of $\bar{\eta}_{ij}$ by transition $uv (u, v \in Q)$ compared to the other transitions. $\pi_u p_{uv}$ in fact is the steady-state probability of transition $uv$. Among the transitions, the one with
Chapter 7: bottleneck identification

the largest steady-state transition probability has the greatest impact on increasing the throughput rate $\bar{\eta}_{ij}$.

So we can define $\pi_0 p_{uv}$ as the weight of the transition $uv$ with respect to $\bar{\eta}_{ij}$. The bottleneck transition of $\bar{\eta}_{ij}$ is the transition which has the largest $\pi_0 p_{uv}$ ($u, v \in Q$).

Since $\pi_0 p_{uv}$ ($u, v \in Q$) is not related to $\bar{\eta}_{ij}$, we can further conclude that the bottleneck transition is the same for the throughput rates of outgoing messages of all the classes.

### 7.3 Queue Wait Time Bottleneck

In Chapter 3, PEFSMs are classified into three types: SyPEFSMs, AsPEFSMs and HyPEFSMs. The incoming messages in a SyPEFSM have no queue wait time. Only incoming messages in an AsPEFSM or asynchronous messages in a HyPEFSM may experience queue waits. Therefore, AsPEFSMs and HyPEFSMs are the two types of the PEFSMs that will be studied in this section. For simplicity, we use PEFSM to refer to an AsPEFSM or a HyPEFSM in the rest of this section.

From the derivations of mean queue wait times, we know that different incoming message classes may experience different queue wait times on average. We are interested in identifying the bottleneck transition with respect to the mean queue wait time of a specific class of incoming messages, say $\bar{W}_{q,ij}$.

First of all, we will revisit the computation of the mean queue wait time of incoming messages of a specific class in an AsPEFSM-$\alpha$. The incoming messages
Chapter 7: bottleneck identification

of a class are assumed to arrive asynchronously and in a Poisson pattern. With this assumption, it can be proved that the mean queue wait times of different classes of firable messages of a state are the same, i.e.,

$$W_{q,ij} = W_{q,i} \quad (i, j \in Q).$$

So we need only to compute the overall mean queue wait time of all the classes of firable messages of a state.

To compute the overall mean queue wait time, we construct the virtual jobs of a state and treat the queueing system of the PEFSM as an M/G/1. The mean queue wait time is computed by applying the well-known solution technique for M/G/1

$$W_{q,i} = \frac{\lambda_i \bar{\zeta}^2_{ij}}{2(1 - \rho_i)} = \frac{\lambda_i \bar{\zeta}^2_{ij}}{2(1 - \rho_i)}.$$

In the above equation, $\bar{\zeta}^2_{ij}$ is the second moment of the service time of class $ij$ virtual jobs. From Appendix B, we see that to compute $\bar{\zeta}^2_{ij} \quad (i, j \in Q)$, a set of equations must be solved. The closed-form solution of $\bar{\zeta}^2_{ij}$ is difficult to obtain. So it is infeasible to compute the partial derivatives of $W_{q,i}$ with respect to each transition time $\psi_{uv} \quad (u, v \in Q)$ for use as the weights to identify the bottleneck transition of $W_{q,i}$. Therefore, the following approximate solution is proposed to compute the weights for each transition with respect to the mean queue wait time of a class of incoming messages.

An approximate approach

As mentioned earlier, the FSM of a PEFSM provides the service order of incoming messages. This ordering affects the performance of the PEFSM and should be
taken into consideration to obtain more accurate results.

In general, we can assume the queuing system of a PEFSM consists of a single server with a single queue. Figure 7.1 shows a queuing system which serves a PEFSM. The service order of incoming messages is controlled by the FSM of the PEFSM and the incoming messages.

![Diagram showing service order implied by the FSM of a PEFSM](image)

Figure 7.1: Service order implied by the FSM of a PEFSM

We know that an asynchronous incoming message to a PEFSM may arrive before the PEFSM is ready to process this message. When this happens, the message will have to wait in a queue. The *queue wait time* of this message is the elapsed time between the moment it arrives and the moment it is processed. From Figure 7.1, it is not difficult to see that the waiting period of this message includes not only the processing times of all the messages of the same class which
arrived earlier, but also the processing times of the transitions associated with the messages of other classes. These transitions bring the PEFSM to the right state so that the target message can be processed. For example, in Figure 7.1, message $m_3$ has to wait for service until the transitions associated with $m_1$ and $m_2$ are processed.

Next we will show how the incoming messages of a class in a PEFSM wait statistically when they arrive early. First, we define transition path and transition subpath.

Definition 7.1 (transition path) A transition path of a PEFSM is a sequence of consecutive transitions in the PEFSM.

Definition 7.2 (transition subpath ij) A transition subpath ij of a PEFSM is a finite number of consecutive transitions in the PEFSM starting from state i and ending at state j. The first transition of the subpath is called the head of the subpath; the last transition is called the tail of the subpath.

Figure 7.2 shows a state-transition tree of a PEFSM. Each state in the tree is a state in the FSM of the PEFSM; each transition is a transition in the FSM. This tree includes all the possible transition subpaths to transition ij.

Suppose $\gamma$ is an incoming message of class ij of the PEFSM and $W_{q,ij}$ ($W_{q,ij} > 0$) is the queue wait time of $\gamma$. Suppose transition subpath 1 in Figure 7.2 includes the transitions which must be processed before $\gamma$.

Assume the PEFSM is in state $k_0$ when $\gamma$ arrives. Then, the transition subpath $k_0i$ includes the transitions which are seen by $\gamma$ and which will be processed before $\gamma$. Let these transitions be transitions $k_0k_1, k_1k_2, \ldots, k_{m-1}k_m$ ($k_m = i$), and $D_1, D_2, D_3, \ldots, D_m$ are their transition times, respectively.
By definition, we have
\[
\sum_{n=2}^{m} D_n < W_{q,ij} \leq \sum_{n=1}^{m} D_n. \tag{7.1}
\]
Transition \(k_0k_1\) may be already in process at the time \(\gamma\) arrives. The other transitions have not yet been processed by then.

We know that \(\gamma\) will wait until the processing of these transitions is finished whether or not the incoming messages associated with them have already arrived. The decrease of the transition time of any of these transitions will reduce the
queue wait time of \( \gamma_i, W_{q,ij} \). If some transitions appear in the transition subpath \( k_0i \) more than once, then decreasing the time of the transition which appears most often in the subpath will reduce \( W_{q,ij} \) the most.

However, different class \( ij \) messages may see different transition subpaths at their arrivals. Furthermore, a transition may be seen in different transition subpaths. So the relative frequency of each transition subpath seen by \( \gamma \) should be taken into account while computing the weights for all the transitions with respect to \( W_{q,ij} \). The relative frequency of a subpath can be computed using the transition subpath probability defined below.

**Definition 7.3 (transition subpath probability)** The probability of a transition subpath \( k_0k_m \) is defined as:

\[
Pr\{\text{subpath } k_0k_m\} = \prod_{n=1}^{m} \pi'_{k_{n-1}k_n} p_{k_{n-1}k_n}
\]

where transition \( k_{n-1}k_n \) \((n = 1, 2, \ldots, m)\) are the transitions of the subpath and \( \pi'_{k_{n-1}k_n} = \pi_{k_{n-1}k_n} p_{k_{n-1}k_n} \) is the steady-state probability of transition \( k_{n-1}k_n \). \( p_{k_{n-1}k_n} \) is the single-step probability of transition \( k_{n-1}k_n \).

Given the probabilities of transition subpaths, we can compute the relative frequency of a transition seen by a specific class of incoming messages. The frequency is the sum of the probabilities that this transition appears in all the possible transition subpaths which satisfy Inequality (7.1). It is useful to compute the relative frequency of a transition because decreasing the time of the transition with the highest frequency by the same amount will reduce the mean queue wait time of the specific class the most. Therefore, in this case, the frequency can be used as a weight to identify the bottleneck transition of the mean queue wait time of the incoming messages of the specific class.
Let \( w_{uv} \) be the weight of transition \( uv \). From the discussion above, we can define

\[
w_{uv} = \sum_{l \in \text{subpaths}} (Pr\{\text{subpath } l\} \cdot Pr\{\text{transition } uv \text{ appears in the subpath } l\}).
\]

Next, we present an algorithm to compute the weights, given the mean queue wait time of a class of incoming messages.

**Computation of weights**

Assume that the single-step transition probabilities in \( P \) of a PEFSM are given, and the steady-state state probabilities \( \pi \) as well as the transition times of each transition have been computed.

Suppose \( \bar{W}_{q,ij} \) is known either by computation or measurement. An algorithm to compute the weights with respect to \( \bar{W}_{q,ij} \) is provided in Figure 7.3.

Procedure 1 of the algorithm initializes all the weights to zero before calling Procedure 2. Procedure 2 computes the weights for all transitions in the PEFSM. Using a recursive procedure, it traverses all the transition subpaths which end at state \( i \) and satisfy Inequality (7.1). The subpath starts backwards from state \( i \). A transition is added to the current head of the subpath in each iteration. This transition becomes the new head transition of the subpath. The transition subpath grows until the sum of the mean transition times of the transitions along the subpath is larger than the given mean queue wait time, \( \bar{W}_{q,ij} \). At each step, the current subpath probability is added to the weight of the head transition.

When Procedure 2 terminates, the weights of all the transitions in the given PEFSM with respect to the mean queue wait time, \( \bar{W}_{q,ij} \), are computed. These
**Procedure 1**: compute weights given mean queue wait time

*Inputs*: \( \overline{W}_{q,ij} \) – the mean queue wait time of class \( ij \) incoming messages;

*Outputs*: the weights of all the transitions in the PEFSM, \( w_{uv} \) (\( u, v \in Q \));

*Steps*:

1. initialize the weights of all the transitions to zero, i.e., \( w_{ij} = 0 \) for \( i, j \in Q \);  
2. call Procedure 2 with arguments \((1, ij, \overline{W}_{q,ij})\).

**Procedure 2**: recursively backtrack to add the subpath probabilities to the transitions which are the heads of the subpaths

*Inputs*: 1) the current subpath probability \( p \);
            2) the reference of the current transition \( uv \);
            3) the remaining waiting time \( R_w \);

*Outputs*: weights of the transitions;

*Steps*:

1. if \( R_w \leq 0 \), return;
2. for (each transition which is immediately before the current transition \( uv \) in the FSM of the PEFSM, say transition \( ku \)) do:
   1) let \( p = p \cdot p'_{ku} \) where \( p'_{ku} \) is the steady-state transition probability of transition \( ku \);
   2) let \( w_{ku} = w_{ku} + p \);
   3) call Procedure 2 with arguments \((p, ku, (R_w - \psi_{uv}))\);  
endfor.

Figure 7.3: Algorithm for weight computation

weights reflect the relative frequency of the transitions seen by class \( ij \) incoming messages. If the transition time of each transition is reduced by the same amount one at a time, the one which has the largest weight will cause the largest improvement in the mean queue wait time of class \( ij \) messages. Therefore, the
transition with the largest weight is the *bottleneck transition* with respect to this mean queue wait time.

**Simulation results**

Simulations have been conducted to validate the accuracy of the bottleneck identification method. The architecture of the simulation experiment is shown in Figure 7.4.

![Figure 7.4: A simulation architecture](image)

The *simulator* module accepts a model description of the PEFSM and simulates the execution of transitions in the FSM of the PEFSM. The module contains an *incoming message generator* which generates the incoming messages to the PEFSM based on the given arrival model of the PEFSM.

The simulation results are fed to the *weight computations* module. This computation module also stores the description data of the PEFSM. The algorithms in Figure 7.3 are used to compute the weights for all transitions with respect to the mean queue wait time of a specific transition. Upon completion of the com-
putation of the weights for all transitions, the module identifies the *bottleneck transition*.

The *modification* module reduces the service time of each transition of the PEFSM by the same small amount one at a time. This module resends the modified data of the PEFSM to the simulation module and the simulation is rerun. All the mean queue wait times of class $ij$ incoming messages in each run are recorded so as to verify if the bottleneck transition in fact causes the largest reduction in the mean queue wait time.

Several protocols were used in our experiments which showed that the proposed technique for bottleneck transition identification works in practice. We report the result of the *alternating bit protocol* in the following and put other results in Appendix F.

The FSM of the alternating bit protocol is given in Figure 6.1. The assumptions about this protocol are the same as those given in the introduction to Chapter 6. So the FSM with these assumptions is a HyPEFSM. The input data of the HyPEFSM are given in Columns 1, 2 and 3 of Table 7.1. The incoming data packets to be transmitted are asynchronous and their arrivals follow a Poisson pattern. All other transition wait times and transition service times are exponentially distributed. Their mean values are given in Columns 2 and 3 of the table, respectively.

Columns 4, 5 and 6 are the simulation results. The steady-state transition probabilities are recorded in Column 4. These results agree with the results from computation of $p'_{ij} = \pi_i p_{ij}$ where $\pi_i$ is the steady-state probability of state $i$ and $p_{ij}$ is the single-step probability of transition $ij$. The weights are computed with
Chapter 7: bottleneck identification

<table>
<thead>
<tr>
<th>transition identifier</th>
<th>mean wait time(^a)</th>
<th>mean service time</th>
<th>transition probability(^b)</th>
<th>transition weight(^c)</th>
<th>queue wait time reduction(^d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T01</td>
<td>200.0(^e)</td>
<td>0.001</td>
<td>0.4738</td>
<td>0.3102</td>
<td>0.0069</td>
</tr>
<tr>
<td>T10</td>
<td>0.001</td>
<td>0.001</td>
<td>0.4738</td>
<td>0.6207</td>
<td>0.0090</td>
</tr>
<tr>
<td>T11</td>
<td>0.001</td>
<td>0.001</td>
<td>0.0525</td>
<td>0.0343</td>
<td>0.0010</td>
</tr>
</tbody>
</table>

\(^a\) All times are in seconds except when otherwise indicated.

\(^b\) Steady-state transition probability.

\(^c\) The weight is computed with respect to the mean queue wait time of the data packets.

\(^d\) The new average queue wait time is measured by decreasing the mean service time of the corresponding transition by 0.002 second. The reduction of the queue wait time is equal to the original value minus the new value.

\(^e\) This is the mean arrival rate (packets/second) because this transition is asynchronous.

Table 7.1: A simulation result of bottleneck identification

respect to the mean queue wait time of a class of incoming messages and recorded in Column 5. Then, in each of the following runs, we select one of the transitions and reduce its service time by a small amount. The simulation is re-run with the modified data. The reduction in mean queue wait time is recorded in Column 6. This procedure is repeated for all the transitions. From the table, we can see that reducing the service time of the transition with the largest weight causes the largest reduction in the mean queue wait time of that class of incoming messages. This result confirms the analysis given in this section.

Experiments with different workload parameters were performed for several protocols. In most cases, the results (as in Table 7.1) from simulation agreed with the analytic results. Only a few exceptions were found. However, even then, the reduction of the mean queue wait time by reducing the service time of the bottleneck transition is close to the largest reduction. The reason why the bottleneck transition occasionally is not identified correctly is that both the
queue wait times and the transition times have variance and we use only the mean value to compute the weights for simplicity.

7.4 Related Work

Performance bottleneck detection and removal has received less attention in the literature than performance prediction. This is not only because the problem itself is hard but also because there is a lack of adequate formal definitions and effective analysis methods. Only a few methods have been proposed to locate the performance bottleneck of a software system.

The concept of critical path was first introduced in the context of project management and used to manage the progress of projects [Lock84]. It was later adopted for parallel processing systems [Yang89] where there are both parallel events and synchronization points. The critical path of a parallel program is the path in the program activity graph\(^1\) which determines the program performance (e.g. shortest execution time). The scope of the potential bottleneck is reduced to within the critical path. Other techniques are used for locating the bottleneck within the critical path.

Lockyer's critical path analysis [Lock84] is often used to identify bottlenecks of parallel or distributed systems which are modeled as acyclic directed graphs [Yang89, Wagn93].

However, only one transition in a PEFSM can happen at a time. The execution of the transitions in a PEFSM are sequential and in a certain order.

\(^1\)A program activity graph is a directed graph which depicts the synchronization points of the whole system.
There is no synchronization with other transitions in a single PEFSM. Therefore, the method of critical path analysis cannot be directly applied to PEFSMs in identifying bottlenecks.

Although intuitively we all know what a bottleneck is, historically, the term bottleneck has had various definitions. They can be classified into the following two categories according to their usage:

- definitions based on analytic approaches
- definitions based on measurements

Using a derivative is an analytical approach to identify a performance bottleneck. For example, in [Ferr78], the derivatives of the mean throughput rates with respect to the service rates of the constituent servers of the system are used to define performance bottlenecks analytically. Suppose $\bar{T}$ is the mean throughput rate of the object system, and $\mu_k$ is the service rate of server $\Sigma_i$ $(k=1,2,...,s)$. If

$$\frac{\partial \bar{T}}{\partial \mu_i} > \frac{\partial \bar{T}}{\partial \mu_j} \quad (j = 1, 2, ..., s; j \neq i)$$

then server $\Sigma_i$ is the performance bottleneck of a system with $s$ servers.

However, this definition presents some problems. It cannot be used whenever $\bar{T}$ is not differentiable. Moreover, the performance bottleneck is related to the performance metric. The bottlenecks with respect to different metrics may be different.

Utilization based techniques constitute another analytical way to determine performance bottlenecks [Leung88, Allen90]. Among the servers in a queuing network model, the one with the highest utilization or the one which first achieves
100% utilization with increasing workload on the system is considered to be the bottleneck of the system. However, this approach is not appropriate for a PEFSM because it is assumed to have only one service center.

Generally, the analytical definition is applied to a model of the system. When an implementation of the system already exists, analysis of data from measurement can be used to identify the bottleneck. In [ZiEt92], the bottleneck is defined as the performance parameter which is most sensitive to performance. The sensitivity of a parameter is defined as

$$\text{sensitivity} \triangleq \frac{\% \text{Change in Performance}}{\% \text{Change in Parameter}}$$

Intuitively, the sensitivity is similar to the weight to a certain extent. Both can be used in analytical approaches and measurement approaches.

### 7.5 Summary

We have proposed a methodology for identifying performance bottlenecks based on the PEFSM performance model. As in the case for performance evaluation, the bottleneck identification methodology can be used in the design phase given only the specification, and also on an implementation when some of the model parameters can be directly measured. In the latter case, it is still necessary to have a formal specification of the implementation.

Weights are used to measure the impact of the reduction of each transition time on the improvement of a specific performance metric. The bottleneck with respect to a performance metric is defined as the transition in the PEFSM with the maximum weight.
Chapter 7: bottleneck identification

The methods to compute the weights of the transitions in a PEFSM with respect to two performance metrics are presented. The first method makes use of the closed-form expression of a performance metric such as throughput. This requires both the closed-form expression of a performance metric and the partial derivatives of the performance metric with respect to each transition time to exist. The second method uses an algorithm to compute the weights with respect to a performance metric. This method is used when no closed-form expression of the performance metric or the derivatives exists or is easily computable.

The second method which identifies the bottleneck transition with respect to the mean queue wait time of a specific class of incoming messages is more general than the first one. It can be applied to the PEFSM in which the arrivals of asynchronous messages are not Poisson, and the mean queue wait time may be obtained either by measurement or through computation.

The mean transition time of the bottleneck transition can be reduced in two ways: reducing the mean transition wait time or the mean transition service time. To reduce the mean transition wait time, one may increase the arrival rate of the incoming messages associated with that transition. For example, we can increase the throughput rate of messages or decrease the queue wait time of messages in a specific workstation by shortening the token turnaround time for this workstation in a token ring network. To reduce the transition service time, one may try to improve the software implementation of that transition or use faster hardware to process that transition.
Chapter 8

On Transition Time Testing based on FSMs

Abstract

A number of methods have been developed to evaluate communication protocol performance based on finite state machines (FSMs). One of the examples is to use PEFSMs. This is introduced in the previous chapters. The service times of the transitions are one of the key parameters in the performance analysis and are assumed to be known a priori. This chapter presents a methodology to estimate transition service times by testing when the protocol implementation is given as a black box. An approach to deriving test sequences for transition service time testing is also proposed.

8.1 Introduction

In the previous chapters, we have discussed how to use PEFSMs to evaluate the performance of communication protocols and to identify bottlenecks. In our analysis as well as in the work by others, the service times of the transitions in a PEFSM are primitive data and assumed to be known a priori. To our knowledge, no work has addressed the issue of how the transition times can be obtained,
particularly when the protocol implementation is given as a black box.

In this chapter, we propose an approach to obtain the transition service times of a communication FSM through *performance testing*. Since performance testing has different meanings depending on the focuses of the studies, we shall classify performance testing for communication protocols into three categories:

1. transition time testing
2. saturation testing
3. benchmark testing

Each of the above tests different performance aspects of the protocol implementation under a different workload condition. The objective of *transition time testing* is to measure the transition service times with respect to the FSM on which the protocol is based; *saturation testing* aims at measuring the peak performance of a communication protocol under the condition that a specific service center is saturated; and *benchmark testing* is used to predict the system’s performance under a specific workload and hardware/software platform. In this chapter, only *transition time testing* is discussed.

A major problem that makes *transition time testing* difficult is that the implementation under test (IUT) of a protocol is often given as a black box so that inserting internal measurement points is almost impossible. Moreover, some transitions in the FSM of a protocol can be *invisible* in that only their inputs or outputs (but not both) are observable. To make matters worse, often a transition can produce a variable number of output messages and the transition can still be in progress at the time its first output message is seen from the outside. On
the other hand, the transition may have finished before its first output message is observed; or the transition has to wait in an internal queue until the previous transitions finish execution. Therefore, measured response times are often inaccurate, especially when the time period is measured in milliseconds or microseconds. It should be obvious from the above discussion that measuring the transition times accurately is not a straightforward matter.

This chapter presents a methodology for measuring/computing the transition service times effectively under certain circumstances. We assume the measurement is done under the same conditions as conformance testing as described in ISO 9646 [ISO9646a]. The testing method (or architecture) used can be remote method [ISO9646a], and the IUT and the tester can be in different machines. We do not require the availability of globally synchronized clocks as long as there is a way to timestamp all the output data from the IUT and send the log file to the tester. Furthermore, we assume the IUT has already passed conformance testing, i.e., it is correct with respect to the specification.

The rest of the chapter is organized as follows. Section 8.2 discusses the relationship between service times and response times and proposes a technique which we shall call the t-test (standing for transition termination test) to estimate the service times based on response times. Section 8.3 studies techniques dealing with invisible transitions. Section 8.4 presents a framework for generating test sequences for transition time testing. Finally, Section 8.5 concludes the chapter.
8.2 T-test

An IUT is an implementation of an FSM (or an EFSM). Since we have assumed that the IUT has passed conformance testing, it is considered to be correct with respect to the FSM. Furthermore, according to ISO 9646, the IUT is given as a black box for conformance testing purpose. The tester may only send messages to an IUT and measure their response times. The measured data are used to analyze the performance. An abstract architecture for performance testing is shown in Figure 8.1, where a tester simulates the peer entity which communicates with the IUT.

![Figure 8.1: A sample conceptual testing architecture](image)

In FSMs, *transition* and *state* are the two basic constructs. A state is rather a logical notation which represents some specific conditions of the system. A transition usually consists of input and output events and data processing.
Conceptually, the current state of an IUT changes after the IUT has received an incoming message from the environment and finished execution of the transition associated with the message. Therefore, executing an IUT can be viewed as executing a sequence of transitions of the corresponding FSM while moving from state to state.

The intervals of sending messages to an IUT are crucial in performance testing. This is further explained in the following.

Let

\[ t_i \] the service time of message \( i \);
\[ \hat{t}_i \] the response time of message \( i \);
\[ D_i \] the transmission delay of message \( i \) from the tester to the IUT;
\[ W_i \] the queue wait time of message \( i \);
\[ \delta_i \] the latency of message \( i \).

\( t_i \) is the service time of the transition associated with message \( i \) which includes only the CPU time but no queue wait time as defined in Chapter 3. \( \hat{t}_i \) is defined as the interval between the time when the message is sent and the time when the first response (usually an outgoing message) is seen. For simplicity, we also use the response time of a transition to refer to the response time of the message that invoked the transition. Note that some transitions may not produce any outgoing messages and thus the response times of these transitions cannot be measured directly. This will be discussed in the next section.

Normally, \( \hat{t}_i \) can be measured directly and accurately if the transition associated with message \( i \) produces any output. However, \( t_i \) cannot be accurately measured because the transmission delay and the internal queue wait time for
service are difficult to obtain precisely when the IUT is given as a black box. For this reason, we propose a technique using the next message's response time to estimate the \textit{transition service time}.

As shown in Figure 8.2, $D_i$ is the interval between the moment that message $i$ is sent by the tester and the moment that the message arrives at the IUT. $W_i$ is the time message $i$ spends on waiting for service\textsuperscript{1}. $\delta_i$ is defined as the interval from the time when message $i$ is sent by the tester to the time when the corresponding transition begins to be executed. Their relationship can be expressed as

$$\delta_i = D_i + W_i$$

where $\delta_i$ is the latency of message $i$, $D_i$ is the transmission delay of message $i$ and $W_i$ is the queue wait time of message $i$.

\hspace{1cm}

In general, $\delta_i$ will decrease if $\delta_i$ decreases. When the message sending interval $\tau_i$ increases, $W_i$ will decrease (if $W_i > 0$). This in turn will reduce $\delta_i$ and $\hat{t}_i$.

\textsuperscript{1}As defined in Chapter 3, to service an incoming message means to execute the transition associated with the message.
In order to obtain the service time of transition $i$ accurately, the interval $\tau_i$ of sending message $i$ from the tester should be increased until $\hat{t}_i$ will not reduce any further. This means the waiting time $W_i$ is approximately zero (i.e., processing of the previous transition is completed by the time when message $i$ arrives). In this case, $\delta_i \approx D_i$. This situation is illustrated in Figure 8.3.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{time_diagram.png}
\caption{A sample time diagram where there is no queue wait time}
\end{figure}

$D_i$ can be obtained accurately if the test environment is controllable. $D_i$ can be assumed to be negligible if the tester and the IUT are on the same computer and there is no other job with higher priority running concurrently with the IUT.\footnote{In this case, the value is usually very small compared to the other times.} This means that the technique is most useful when the tester and the IUT are on the same computer or in the same location connected by a short and dedicated link where network load variation is not a problem. Under this condition, and assuming the response time and the service time end at the same time (we shall relax this assumption later on), we have

\[ \hat{t}_i \approx t_i. \]
From the above analysis, we see that the tester should control not only what to send to the IUT but also when to send it. When there is no queue wait time, the response time of a message provides a rough estimate of the service time of the corresponding transition.

The above discussion also shows that the response time of a message can be used to determine whether or not the processing of the previous transition has completed. This is useful when the response time of the previous transition cannot be directly observed externally, or when we would like to have a more accurate estimate of the service time but the response time and the service time do not end at the same time. The technique is called *t-test* and is explained in detail in the following.

Suppose message *i* is sent at time *A* in Figure 8.4, and at time *B* the first response due to message *i* is seen. Assume that the processing of the transition associated with message *i* finishes at time *C*. *C* may be before or after *B*, and *C* may not be observable externally to the IUT because of the *black box* assumption of the IUT. Our objective is to determine *C* which will allow us to derive the service time of the transition associated with message *i* (i.e., *AC*).

![Figure 8.4: Selecting the message sending intervals](image-url)
We select a time $E'$ to send the next message (i.e., message $(i+1)$) to the IUT. $E'$ must be chosen after the last response \(^{3}\) to message $i$ is seen. To determine whether or not the IUT has finished processing the previous transitions at $E'$, we repeat the sequence of message transmissions except that in the next round, message $(i + 1)$ is sent at time $E''$ where the interval between $B$ and $E''$ is twice as long as the one between $B$ and $E'$, i.e.,

\[(E'' - B) = 2 \cdot (E' - B).\]

It is known from the prior discussion that increasing the sending interval will reduce or eliminate the queue wait time of message $(i+1)$. Therefore, the response time of message $(i + 1)$ can only decrease (or remain unchanged). If there is no change in the response time in this round of test, it means the IUT has finished processing message $i$ and its associated transition by time $E'$. If not, the previous step is repeated using the same test data but moving $E'$ to $E''$ and selecting a new value of $E''$ as before. This is continued until there is no change in response time or the change is smaller than a predefined small threshold.

Without loss of generality, let us call the time instant $E'$ that was determined in the round before the last one $E$, and the smallest response time that we obtained as $R_{i+1}$. The value of $E'$ in the second last round is used because the response times in that round and the last round are the same. Taking the value of the second last round will reduce the overhead in the next phase of testing (see below). The interval between $A$ and $E$ is recorded as $\tau_{i+1}$. Based on the previous discussion, it is obvious that $R_{i+1}$ is the smallest response time for message $(i+1)$.

This phase is called the *expanding phase* because $E'$ is moved further and

---

\(^{3}\)The transition associated with message $i$ may generate more than one outgoing message.
further from A. This is achievable because the testing environment is completely under our control so that no other jobs will interfere with the testing. The queue wait time will be zero when the message sending interval is larger than the service time of the previous transition.

Now we are in a position to determine the instant C. We know from the expanding phase that

$$(B - A) \leq (C - A) \leq (E - A).$$

The idea of the next testing phase is to repeat sending messages i and (i + 1), but in each round bring E closer to C until the time difference between two successive sendings of message (i + 1) is less than a predetermined margin. As a result, the interval between C and E is less than this difference which is the error margin that we are willing to tolerate. Note that all times in each round are measured relative to A.

This procedure is formalized in the following algorithm. The function $\tau$ is defined as $\tau(x_2, x_1) = |x_2 - x_1|$ $(x_1, x_2 > 0)$. The parameters $\tau_{i+1}$ and $R_{i+1}$ are initialized with the values obtained in the expanding phase.
Algorithm: determining transition completion time after the expanding phase has been completed

Inputs: 1) $pr_i$ — the preamble\(^{a}\) of transition $i$;
2) $ps_{i+1}$ — the postamble\(^{b}\) of transition $(i+1)$;
3) $\tau_{i+1}$ — the interval between sending message $i$ and message $(i+1)$;
4) $R_{i+1}$ — the smallest response time of message $(i+1)$;
5) $\epsilon$ — the predetermined error margin;

Outputs: $t_i$ — the service time of transition $i$;

Procedure:

1. set $\tau''_{i+1} = \tau_{i+1}$ and $\tau'_{i+1} = \frac{1}{2} \tau_{i+1}$;
2. send $pr_i$ to set the protocol at the starting state of transition $i$;
3. send message $i$ and label the sending instant as $A$;
4. send message $i+1$ at time $D$ where $\tau(D,A) = \tau'_{i+1}$;
5. measure the response time of message $i+1$ and call it $R'_{i+1}$;
6. if $(R'_{i+1} = R_{i+1})$ then
   set $\tau_{i+1} = \frac{1}{2} \tau'_{i+1}$;
   go to Step 2;
else
   continue Step 7;
7. set $\tau_{i+1} = \tau'_{i+1} + \frac{1}{2} \tau(\tau''_{i+1}, \tau'_{i+1})$;
8. send $pr_i$ to set the protocol at the starting state of transition $i$;
9. send message $i$ and label the sending instant as $A$;
10. send message $i+1$ at time $D$ where $\tau(D,A) = \tau_{i+1}$;
11. measure the response time of message $i+1$ and call it $R'_{i+1}$;
12. if $(R'_{i+1} = R_{i+1})$ then
   set $\tau''_{i+1} = \tau_{i+1}$;
else
   set $\tau'_{i+1} = \tau_{i+1}$;
13. if $(\tau(\tau''_{i+1}, \tau'_{i+1}) <= \epsilon)$ then
   set $t_i = \tau_{i+1}$;
   stop;
else
   send $ps_{i+1}$ to set the protocol back at the idle state;
   go to Step 7.

\(^{a}\)The preamble of a transition is a sequence of consecutive transitions starting from the idle state and leading to the start state of the transition.

\(^{b}\)The postamble of a transition is a sequence of consecutive transitions bringing the protocol from the end state of the transition back to the idle state.

\(^{c}\)Note that $R'_{i+1}$ cannot be less than $R_{i+1}$ because $R_{i+1}$ is the smallest response time obtained in the expanding phase.
Chapter 8: transition time testing

The first 6 steps of the algorithm try to find a lower bound on the service time. An upper bound on the service time has been obtained in the expanding phase. In each round from Steps 7 to 13, the difference between the lower bound and the upper bound is reduced by half to form the new upper bound. So this algorithm is actually a type of binary search. As a consequence, the distance between $C$ and $D$ can be made arbitrarily small. In this sense, the service time of message $i$ (i.e., time interval $AC$ in Figure 8.4) can be determined accurately. This procedure can be automated and shall be referred to as the shrinking phase because the upper bound on the service time is gradually reduced in this phase. The shrinking phase implicitly assumes that the message sending intervals can be made smaller than the minimum transition service time.

The expanding phase and shrinking phase together constitute what we call the $t$-test.

As shown above, the complete $t$-test needs to execute the same test subsequence a number of times. The number can be reduced if we are willing to accept a larger error in service time estimation. Moreover, if it is adequate to use the response time of a transition as an estimate of its service time in performance analysis, then only the expanding phase of the $t$-test is needed to make sure no queue wait time occurs for the specific transition. The expanding phase of the $t$-test is much easier to achieve and in the best case it needs only two rounds.

In the rest of the chapter, we assume only the expanding phase of the $t$-test is used in the transition time test whenever the response time of a transition can be measured directly.
8.3 Testability

The \textit{t-test} requires at least two consecutive transitions. It assumes that the input of the first transition is controllable and both the input and the output of the second transition are observable. However, not all transitions in an FSM have both observable input and output events. Transitions without any input are known as \textit{spontaneous transitions} [Sari84]. We shall define a transition without any input (i.e., \(-/0\)) or a transition without any output (i.e., \(I/-\)) as an \textit{invisible transition}. Conversely, a \textit{visible transition} is defined as a transition in which both its input and output are observable (i.e., \(I/O\)). A transition dealing with the timeout event is a typical invisible transition because timeout is usually generated internally.

Invisible transitions make performance measurement difficult. Their response times cannot be obtained directly and thus these transitions cannot be used to detect the completion of previous transitions. Fortunately, sometimes this problem can be solved by combining an invisible transition with other transition(s). This will be discussed in detail later in this section.

Figures 8.5 and 8.6 list all the possible situations of two consecutive transitions involving invisible transitions. Each of the cases is analyzed in the following:

1. The simplest case is when both transitions are visible (Figure 8.5 (1)). The measured response time can be used to estimate the service time of the corresponding transition. Furthermore, as explained in Section 8.2, the response time of the second transition can also be used to determine the service time of the first transition accurately by the \textit{t-test}.
Figure 8.5: Two consecutive transitions with at least one visible transition

Figure 8.6: Two consecutive transitions without any visible transition

2. In Figure 8.5 (2), the second transition is invisible and its response time cannot be measured directly. Therefore, it cannot be used directly to determine the service time(s) of its preceding transition(s). However, if the first transition can be combined with other visible transitions following it as shown in path 2 of Figure 8.7 (1), then we have case 1 and the service time of the first transition can be accurately determined. Furthermore, the second transition can also be treated as a visible one by combining it with a visible transition that follows it (path 1 in Figure 8.7 (1)). Path 1 then
becomes case (3) in Figure 8.5 and can be analyzed using the technique described below.

3. The second transition in Figure 8.5 (3) is visible. T-test can be used to determine the service time of the first transition (which is invisible) as discussed in Section 2.

Alternatively, if the two transitions in Figure 6 (3) are treated as one transition, it is visible and its response time can be obtained directly. Suppose the response time is \( t \) and the response time of the second transition is \( t_2 \). Therefore, the response time of the first transition, \( t_1 \) can be obtained using the simple relation : \( t_1 = t - t_2 \). \( t_1 \) can be used as a rough estimate of the service time of the first transition.

4. Figure 8.5 (4) can be handled in a way similar to that of Figure 8.5 (3). The second transition is invisible because it can be fired without any input. However, we can combine these two transitions and treat them as a single visible transition as discussed in case 3.

5. The first transition of Figure 8.5 (5) is not useful and neither is the combination of the two transitions because it is still invisible. However, it may be possible to calculate the service time of the first transition with the help of the second visible transition and some simple computation as discussed later in this section.

6. In cases (1), (3) and (4) of Figure 8.6, like the case in Figure 8.5 (5), the two transitions are not very useful either individually or as a unit. Other transitions in the FSM are needed to compute the service times of these
two transitions (see below).

7. The two transitions in Figure 8.6 (2) can be combined into one visible transition if necessary. However, the individual service time for each transition can only be obtained if at least one of them can be measured through other subpaths.

![Diagram of transitions](image)

Figure 8.7: Example of test subpaths for invisible transitions

As we have seen, sometimes it is necessary to combine two or more consecutive transitions in the FSM as one transition for use in the test. We shall refer to such a transition as a compound transition. A compound transition is visible if the input of its first transition and the output of its last transition are both observable. For example, $I_1/-, \cdots, -/O_n$ is a visible compound transition.

The response time of a transition is the response time of the associated incoming message which is the interval between the time when the incoming message is sent and the time when the first outgoing message of the transition is seen. A transition is defined as measurable if its response time can be measured. A visible
(compound) transition is always measurable. Its service time can be estimated by measuring its response time.

We define a (compound) transition to be testable in transition time testing if its service time can be obtained by either direct or indirect measurement. An example of indirect measurement is the t-test. By this definition, a visible transition (either single or compound) is always testable. An invisible (compound) transition may become testable under certain circumstances as discussed earlier.

It is reasonable to assume that in a communication protocol, there exists at least one observable input event and one observable output event. Furthermore, since a correct FSM is always strongly connected, the following lemma is true.

**Lemma 8.1** An invisible transition can always be included either at the head or at the tail of a visible compound transition.

**Proof:**

An invisible (single) transition has the form I/- or -/O. If the invisible transition is of the form I/-, the visible compound transition is the transition path starting with I/- and ending with the transition having an observable output event. If the invisible transition has the form -/O, the visible compound transition is the transition path that starts with an observable input event, and ending in -/O.

To tackle invisible transitions, we present the following theorems.

**Theorem 8.2** An invisible transition is testable if it can be included as the head or tail of a testable compound transition and the rest of the compound transition is testable.

**Proof:** This follows directly from the definition of testable transition.
Theorem 8.3 (1) An invisible transition $I/-$ is testable in an FSM if there is at least a transition immediately following it which has the form of $I'/O'$ or $I'/-$, or which is a testable $-/O'$. (2) An invisible transition $-/O$ is testable if there is at least a transition immediately before it which has the form of $I'/O'$ or $-/O'$, or which is a testable $I'/-$.

Proof:

(1) The testable (compound) transition succeeding $I/-$ may begin with a transition of the form $I'/O'$, $I'/-$ or $-/O'$. If $I'/O'$ is the first transition in the testable (compound) transition, then we have case 3 of Figure 8.5 and the service time of $I/-$ can be derived using the t-test. So $I/-$ becomes testable in this case.

Otherwise, if $I'/-$ follows $I/-$, according to Lemma 8.1, $I'/-$ can always form a visible compound transition with other transitions in the FSM and the compound transition is testable. So does $I/-$. The service time of $I/-$ is the difference of the service times of these two compound transitions. Therefore, $I/-$ is testable in this case.

The last case is that a testable $-/O'$ follows $I/-$. $I/-$ can form a visible compound transition with $-/O'$ and the compound transition is testable. The service time of $I/-$ is the service time of the compound transition minus that of $-/O'$. $I/-$ is also testable in this case.

(2) Similar analysis can be applied to prove (2) of the theorem.

In the next section, we present a method for selecting test sequences for transition time testing based on the above discussion of testability.
8.4 Test Sequence Generation

The objective of transition time testing is to obtain the service times of all the transitions in the FSM of the protocol. The test data that are used to achieve this are called test sequences. Here a test sequence is defined as a sequence of consecutive transitions which takes the FSM from its beginning (usually idle) state back to the same state. A test subsequence is a segment of a test sequence, but not necessary starting from the beginning state or ending at the beginning state.

Since the service times of all the transitions in an FSM are primitive (i.e., required) data for performance analysis, all the transitions should be covered in the test sequences. Furthermore, the test sequences should be able to check that there is no queue wait time for the visible (compound) transitions so that

\[ \hat{t}_i \rightarrow t_i, \]

as discussed before. For this reason, a test subsequence should include at least one visible (compound) transition. As explained earlier, the visible (compound) transition can be used to estimate its own service time and/or to determine the service time of its previous transition in the t-test.

The framework for generating test subsequences for transition time testing consists of two phases. In phase one, all the visible transitions in the FSM are selected to be tested and their service times obtained. In phase two, the other transitions left (mostly invisible) are considered in the way presented in the previous section. This is discussed in detail below.

The FSM of the protocol is considered as a directed graph, denoted as \( G \),
where the nodes stand for the states and the directed edges for the transitions. 

$G$ is a strongly connected graph because, for any correct communication protocol (specified by an FSM), there should be at least one transition sequence leading to any state from the beginning state, and at least one transition sequence bringing the protocol back to the beginning state.

In the derivation of the test sequences, a subgraph of $G$ is first constructed, denoted as $G^{(0)}$. It includes all the nodes and all the visible transitions of $G$ (excluding the invisible transitions). The set $S^{(0)}$ initially contains all the invisible transitions of the FSM. $G^{(0)}$ may not be a strongly connected graph any more as shown in Figure 8.8. However, all the edges in $G^{(0)}$ are visible and testable. So each transition in $G^{(0)}$ is treated as a subsequence by itself.

Phase two begins after the test subsequences for all the visible transitions have been generated. Select any invisible transition from $S^{(0)}$ and put it into $G^{(0)}$. If the selected invisible transition is of the form $I/-$, check if there is any transition in the current graph immediately after it. If not, put it back into $S^{(0)}$ and pick another transition. If the following transition is of the form $I_1/O_1$, the test subsequence $(I/-, I_1/O_1)$ is generated; otherwise, if the following transition is of the form $-/O$, select the test subsequence $(I/-,-/O)$. If the above cases are all false, the following transition must be of the form $I_1/-$. In this case, two visible compound transitions are generated, one starting with $I_1/-$ and the other with $(I/-, I_1/-)$. Note this is always possible since all transitions in $G^{(0)}$ are testable. The difference in service times of the two compound transitions gives an estimate of the service time of $I/-$.

Likewise, if the selected invisible transition is of the form $-/O$, check the graph to determine if there is any transition in the current graph immediately before it.
If not, put it back into $S^{(0)}$ and pick another one. If the transition preceding $-/0$ is of the form $I_1/O_1$, the test subsequence $(I_1/O_1, -/0)$ is generated; otherwise, if the preceding transition is of the form $I/-$, the compound transition $(I/-, -/0)$ is selected as a test subsequence. If both cases are false, the preceding transition must be of the form $-/O_1$. In this case, two visible compound transitions are generated, one ending with $-/O_1$ and the other with an $-/O_1, -/0$.

If the newly selected invisible transition becomes testable, it remains in $G^{(0)}$, and the graph becomes $G^{(1)}$ with one more edge while $S^{(0)}$ becomes $S^{(1)}$ with one fewer transition.
This process is repeated until the set of invisible transitions is empty or none of the invisible transitions in the set can be made testable using this procedure.

Finally, we reconsider all the transitions remaining in $S^{(n)}$. First, for each transition of the form $I/-$, check in $G$ if there is another transition of the form $I_k/-$ following it, which may still be in $S^{(n)}$. If so, this transition can become testable according to Theorem 8.3. Two visible compound transitions are then generated, one starting with $I_k/-$ and the other with $(I/-, I_k/-)$. Record these two subsequences for the transition, remove $I/-$ from $S^{(n)}$ and put it in $G^{(n)}$. Similarly, each invisible transition of the form $-/O$ in $S^{(n)}$ is checked. The procedure ends when all the transitions in $S^{(n)}$ have been checked or the set is empty.

If any invisible transition has become testable in this phase, Phase two of the procedure is repeated until no remaining transitions can be made testable.

Note that it is possible that some transitions in the FSM are untestable. Figure 8.9 shows an example FSM with two untestable transitions ($I/-$ and $-/O$). How to deal with these transitions is left as future work. A simple method is to examine the source codes of these transactions to arrive at an estimate of the execution times.

![Figure 8.9: A sample FSM with untestable transitions](image)

Each test subsequence derived above is a visible compound transition. To form the test sequences, a *preamble* which is a transition sequence starting from
the beginning state of the FSM and ending at the starting state of the compound transition, and a postamble which is a transition sequence starting from the ending state of the compound transition and ending at the beginning state must be added.

Since a test subsequence may have to be tested many times, the shortest preamble and postamble of an FSM should be used to reduce the testing cost. An algorithm to find the shortest path between two vertices in a directed graph can be found in [RoWr88].

**An example**

To illustrate, we use an abstract FSM shown in Figure 8.10.

```
Suppose that Figure 8.10 is graph G. The initial subgraph \( G^{(0)} \) of G in the test sequence generation procedure is given in Figure 8.11 (a). The initial invisible transition set is \( S^{(0)} = \{ -/O3, -/O4, I5/-, I6/- \} \). Two test subsequences are generated from \( G^{(0)} \): I1/O1 and I2/O2.

Next, the first invisible transition in \( S^{(0)} \), \(-/O3\), is selected and put in \( G^{(0)} \). It can be made testable using the subsequence I2/O2, \(-/O3\). \( G^{(1)} \) is given in
```
Figure 8.11 (b) and $S^{(1)} = \{-/O4, I5/-, I6/-\}$.

Then, $-/O4$ is selected because $-/O3$ is now testable. The test subsequence for $-/O4$ is $I2/O2$, $-/O3$, $-/O4$. $G^{(1)}$ is given in Figure 8.11 (c) and $S^{(2)} = \{I5/-, I6/-\}$.

$I5/-$ can also become testable using $t$-test with the test subsequence $(I5/-, I1/O1)$. As a result, $S^{(3)}$ becomes $\{I6/-\}$.

Finally, $(I6/-, -/O4)$ is used to derive the service time of $I6/-$.

In summary, the test subsequences derived are:
(1) I1/01;
(2) I2/02;
(3) I2/02, -/03;
(4) I2/02, -/03, -/04;
(5) I5/-, I1/01;
(6) I6/-, -/04;

8.5 Experimental Results

A tester which implements t-test was built to validate the methodology. This tester uses the architecture shown in Figure 8.12.

This architecture is a realization of the abstract one introduced in Figure 8.1. It consists of an IUT module and a tester module. The IUT and the tester may reside on two different computers. The clocks of the two computers are
not required to be completely synchronized as long as their time differences are kept reasonably constant. This is because t-test uses the sending interval of two consecutive messages to approximate the service time of the first message. The sending interval is computed using the clock of the tester. Nevertheless, if the clock of the IUT is behind the clock of the tester, then the response time of a message which involves the two clocks may be close or equal to zero. This situation can be avoided easily by setting the clock of the IUT a little bit ahead of the clock of the tester.

When the lower or upper service interfaces of the IUT are not directly accessible, then the IUT implementor or the test evaluator must add some codes shown in the shadow boxes of Figure 8.12 outside the IUT to record the times of the output data or service primitives. This is similar to the ferry clip testing method [Chan89, Chan93].

The tester sends messages to the IUT to trigger transitions. This is implemented using UNIX's datagram sockets (UDP). The timestamps of each of the outgoing messages from the IUT are recorded in a log file. The tester also recorded in its memory the timestamps of each message it sends to the IUT. At the end of each round of test, the tester reads the log file and recomputes the message sending intervals for the next round according to the algorithms presented in Section 8.2.

Generally speaking, the IUT can be any protocol implementation. We implemented a simple protocol whose FSM is illustrated in Figure 8.13. The service time of each transition in this FSM consists of two parts: the first part is before the output event (if any) of the transition; the other is after. Both times are generated randomly but only once for an IUT. Therefore, the output event of a
transition is somewhere in the middle of the service of a transition. This is done so as to make the response time of a message an inaccurate estimation of the service time of the associated transition.

![FSM of an abstract protocol](image)

*Figure 8.13: The FSM of an abstract protocol*

We used this implementation as an IUT, treated it as a black box and conducted the t-test.

First, we derived a test case using the method introduced in Section 8.4. This test case is \((L:11/L:O1, U:12/-, L:13/U:03, L:11/L:O1)\). By the definition of t-test, the service times of all three transitions can be obtained only with this test case.

Secondly, we ran the t-test and got the results which are listed in Table 8.1. For each row in the table, the first column contains the transition identifier labelled with their input and output events. The second column is the service times of the transitions which were randomly generated. The third column is the measured
response times. The forth column is the result of the *shrinking phase* of the t-test.

<table>
<thead>
<tr>
<th>Transition</th>
<th>Service Time(^a)</th>
<th>Measured Response Time(^b)</th>
<th>T-test Result(^c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1/O1</td>
<td>216118(^d)</td>
<td>136378</td>
<td>197133</td>
</tr>
<tr>
<td>I2/-</td>
<td>839081</td>
<td>N/A(^e)</td>
<td>856853</td>
</tr>
<tr>
<td>I3/O3</td>
<td>1208362</td>
<td>1090068</td>
<td>1211972</td>
</tr>
</tbody>
</table>

\(^a\) The service time of each transition is generated randomly in two parts: one is before the output of outgoing messages; the other is after the output.

\(^b\) The response time is the difference between the message sending time and the output seeing time.

\(^c\) The error tolerance is 5%.

\(^d\) All times are in microseconds.

\(^e\) Not applicable because this transition has no output.

Table 8.1: Simulation and Testing Results

The results show that the estimations of the transition service times (Column 4) using t-test are much closer to the actual service times (Column 2) than the measured response times (Column 3). Accurate results are obtained even for some invisible transitions.

We also apply the t-test to a real protocol implementation which contains a subset of LAPB. The test results are presented in Appendix G.

### 8.6 Summary

A number of papers have been published on performance analysis of communication protocols directly from their formal descriptions using analytic techniques or simulation (see for example, [Krit87, BoVa88, LiLi88, GuRu89, AFF90, FVV91, KrWh93, ZhCh93]). Most of the work is based on FSMs and with the assumption that the transition service times are somehow known in advance.
Chapter 8: transition time testing

In this chapter, we have shown that it is possible to obtain an accurate estimate of the service time of a transition in an FSM using the $t$-test, even though the protocol implementation is given as a black box. The $t$-test uses the same environment as an ISO conformance test and can be performed right after conformance testing. A $t$-test consists of two phases: expanding phase and shrinking phase. In the expanding phase, the $t$-test needs to execute the same transition subsequence more than once in order to make sure the sending interval is long enough for the previous transitions to finish processing. Then, in shrinking phase, the $t$-test uses binary search technique to determine rapidly the finishing point of the service time of a transition. The key to the $t$-test is that a tester must have control over both the test sequences and the sending intervals of the messages.

The testability within the scope of transition time testing has also been discussed. Although some transitions of an FSM are invisible, it may be possible to combine them with other transitions to form testable compound transitions which can be used to determine the service times of their own as well as other (compound) transitions.

Based on the discussion of testability, a method of test subsequence generation (TSG) from a given FSM is proposed. The procedure of the TSG consists of two iterative phases. The best case is when all the transitions of the FSM are visible. In this case, only Phase one of the generation procedure needs to be executed and its time complexity is $O(n)$, where $n$ is the number of the transitions.

Each round of Phase two will take $O(nm + m^2)$ time, where $m$ is the number of invisible transitions remaining in the set. Phase two will have to be repeated if any invisible transition becomes visible in the round. Therefore, the worst case situation occurs when only one invisible transition becomes visible in every
round so that Phase two has to be repeated the maximum number of times. So
the worst case time complexity of the TSG is $O(nm^2 + m^3)$ where $m$ is the number
of invisible transitions.

When the service times of all the transitions in an FSM are determined,
the performance of the protocol specified by the FSM can be computed using
the PEFSM method presented in Chapter 4, 5 or 6 of this thesis, queueing analysis [ZhCh93], reachability analysis [LiLi88, BaBu91], or other methods
[Krit87, KrWh93].
Chapter 9

Conclusions and Future Work

9.1 Conclusion

Formal description techniques (FDTs) have found various applications in software engineering, including specification, semi-automatic implementation, test case generation, verification and performance prediction. The main objective of this thesis is to develop models, algorithms and tools which further enhance the power of existing FDTs in performance analysis.

Each FDT has at least one embedded formal model (e.g. finite state machine (FSM)). The formal model can be used to characterize the properties of a communication protocol such as liveness, state reachability, message service order, and queueing and service disciplines. These properties are useful in analysis, design, simulation, performance evaluation and testing of the protocol. Compared to conformance testing of communication protocols, performance testing and evaluation based on FDTs are relatively under-developed; not much has been
published in this field.

In this thesis, we have proposed a new performance model called *Performance Extended Finite State Machine* (PEFSM) and the solution techniques based on PEFSMs. This framework may be used to evaluate the performance of communication protocols in the design stage (based on a formal specification alone), or on the actual implementations. In the latter case, it is still necessary to have the formal specifications of the implementations.

### 9.1.1 Contributions

The thesis has addressed three aspects of performance evaluation:

- *performance prediction*,
- *performance improvement*, and
- *performance measurement*.

The major contributions of this thesis are summarized below:

- formulation of a new performance model called PEFSM and a framework to evaluate performance based on PEFSMs;
- proposal of a methodology to identify performance bottlenecks based on PEFSMs;
- presentation of a testing methodology and architecture to obtain the *transition service times* when the protocol implementation is given as a black box;
• discovery of a queueing property with M/G/1 queueing systems with multiple job classes and a similar property with a class of PEFSMs called AsPEFSM-αs;

• survey of research work and tools on performance evaluation related to FDTs.

Unlike the previous performance models based on FSMs, a PEFSM integrates an incoming message arrival model with an FSM. The distinction of incoming message arrival models based on synchronous and asynchronous characteristics allows us to classify the PEFSMs into three categories: synchronous PEFSMs (SyPEFSM), asynchronous PEFSMs (AsPEFSMs) and hybrid PEFSMs (HyPEFSMs). Furthermore, queueing models are integrated with AsPEFSMs and HyPEFSMs to obtain the queue wait times of incoming messages. The solution techniques of performance evaluation for the HyPEFSMs in this thesis are based on the techniques for the SyPEFSMs and the AsPEFSMs.

The issue of which type of PEFSM should be used to analyze the performance of a protocol that is specified by an FSM depends on many factors. As we have seen in the thesis, some of the factors are related to the assumptions of the underlying system on which the protocol runs. In certain circumstances, a complex model such as a HyPEFSM can be transformed into a simpler model such as a SyPEFSM or an AsPEFSM for analysis. This can be achieved by obtaining the values of some performance parameters such as transition probabilities and transition wait times through simulation. This is an approach which combines analytic methods with simulation methods in performance analysis. This approach can help the performance evaluator to achieve a reasonable trade-off between the
accuracy of performance evaluation and the computation complexity.

In Chapter 7, we show that PEFSMs can be used not only to model and predict performance in a more realistic manner but also to define and identify *software bottlenecks* in terms of the transition that has the greatest impact on throughput and mean queue wait time. Properly assigning *weights* to transitions in a PEFSM with respect to a specific performance metric can help to identify and improve the performance of this metric more effectively. Two different methods to compute weights are presented in Chapter 7 with respect to the throughput of a specific class of outgoing messages and the mean queue wait time of a specific class of incoming messages.

The transition service times are important parameters for many performance models. Previous work has assumed these times are known a priori. In Chapter 8, we show how the transition service times can be measured/estimated from a protocol implementation when it is given as a black box. A testing method called *t-test* is proposed to obtain the transition service times of invisible transition. Each round of t-test involves at least two (compound) transitions. The response time of the second (compound) transition is used to identify the exact finishing time of the first (compound) transition so that the service time of the first (compound) transition can be accurately computed. This makes t-test a very effective means to obtaining the service time of an invisible transition.

Simulations and measurement experiments were conducted to validate the methodologies proposed. The results showed the proposed models and methodologies are reasonably accurate.
We have also observed and proved a queueing property for queueing systems with multiple job classes. That is, for an M/G/1 queueing system with multiple job classes, the mean queue wait times of different class jobs are the same. We further prove that in an AsPEFSM, the mean queue wait times of firable messages of the same state for different classes of jobs are also the same. With the latter property, the different classes of firable messages of the same state can be treated as one class in the computation of mean queue wait times for AsPEFSMs and HyPEFSMs. Both of these properties are proved by applying the well-known queueing property—Poisson arrivals seeing time average (PASTA).

9.1.2 Applicability of PEFSM to EFSM

In addition to FSM, EFSM is often used for modelling communication protocols. A conventional EFSM is an FSM extended with variables which are used to control various aspects of the EFSM.

We shall use the formal specification language Estelle [Chari89] as a case study. Estelle is designed to specify EFSMs for communication systems.

A variable in the EFSM of an Estelle specification may appear within a transition or in the conditional clause of a transition. In the first case, the variable may affect the transition service time or the generation of outgoing messages. An example is a variable in a while or if ... then ... statement within a transition. If the variable in a transition affects only the service time, the performance model PEFSM is still applicable. With the variable, the service time distribution of the transition may no longer be constant. Since PEFSM does not impose any constraint on the distributions of transition service times, performance metrics
can be derived as usual based on PEFSMs. Even if the influence of a variable on a transition affects the generation of outgoing messages, PEFSM is still applicable. Suppose a transition may produce one of two types of outgoing messages, say $O_1$ and $O_2$, depending on the value of a variable. We can define the class of outgoing messages generated by this transition to consist of either $O_1$ or $O_2$. The rest of the performance evaluation remains the same. However, we note that the throughput computed using Equation (4.2) is concerned with the overall outgoing messages ($O_1$ and $O_2$) with no distinction between them.

In the second case, the variable is contained in the condition of a transition which determines state changes in an EFSM. An example would be a variable in a *when* or *provided* statement of an Estelle specification. We can define the incoming message class of the transition to be both the incoming message type and the variable with the values (or range) which satisfy the condition of the transition. In this way, PEFSM can still be used to model and evaluate the performance of the system based on the EFSM.

Some EFSMs are non-deterministic, i.e., a transition with exactly the same input may end up in different states in the EFSM at different times. We can address this in the conventional manner [LiLi88, BoVa88]: additional transition probabilities are used to model these non-deterministic choices. Let, for instance,

$$p_i^{(1)} = Pr\{\delta(i, I_i) = j_1\};$$

$$p_i^{(2)} = Pr\{\delta(i, I_i) = j_2\};$$

$$\ldots$$

$$p_i^{(n)} = Pr\{\delta(i, I_i) = j_n\};$$
and

\[ \sum_{k=1}^{n} p_i^{(k)} = 1 \]

where \( i, j_1, j_2, \cdots, j_n \in Q \).

\( p_i^{(k)} \) is directly equivalent to the transition probability if \( I_i \) is the only class of fireable messages of state \( i \). Otherwise, \( p_i^{(k)} \) is used as the conditional probability in the computation to obtain the probability of the transition from state \( i \) to state \( j_k \).

9.2 Future Work

The thesis has established a foundation for performance evaluation based on PEFSMs. The work creates a new research area which can be further developed in many directions.

9.2.1 System-wise performance evaluation

An FDT specification of a communication protocol in a formal specification language such as Estelle could consist of more than one module (see for example Figure 9.1). As a result, two levels of performance analysis are needed:

- single module (module-wise)
- the entire system (system-wise)

In module-wise analyses, only the module of interest is investigated. All information regarding the other modules is given as external factors, such as workload
parameters. In this sense, the work reported in this thesis belongs to this category. To evaluate network performance, this level of analysis may not be enough since multiple modules should be considered at the same time.

System-wise analyses take all the modules explicitly into consideration in performance evaluation. To use our performance evaluation method directly, one may combine the FSMs in all the modules into one large FSM. Unfortunately, this combination usually increases the number of states in the resultant FSM exponentially. The computational complexity also increases tremendously as a consequence.

An alternative is the iteration method described below. Take the example given in Figure 9.1 for a closed queueing model consisting of only two modules A and B.

It is obvious that:

\[ \lambda_A = f_1(X_B) \]
\[ X_A = g_1(\lambda_A) \]
\[ \lambda_B = f_2(X_A) \]
\[ X_B = g_2(\lambda_B) \]
where $X_A$ and $X_B$ are throughputs from modules A and B respectively and $\lambda_A$ and $\lambda_B$ are the arrival rates for modules A and B respectively.

These parameters are dependent on each other. By substitution, for example, we have:

$$\lambda_A = \mathcal{F}(\lambda_A)$$

where $\mathcal{F} = f_1 \circ g_2 \circ f_2 \circ g_1$ and $f_1 \circ f_2(x) = f_1(f_2(x))$.

Some parameters are given initial values and the equations solved iteratively. Iterations in this way would refine the values of some performance elements. The process stops when the change in values is less than some predefined limit. Fixed point theory [Dugu82] could also be applied to check for the existence of solutions and to find the values.

The applications of system-wise analysis could range from network performance evaluation to network management.

### 9.2.2 Queueing theory

We have studied only a few message arrival patterns in AsPEFSMs and HyPEFSMs. To increase the applicability of PEFSM, more arrival models need to be analyzed. For example, the queueing systems in AsPEFSMs and HyPEFSMs may have modulated or correlated inputs. Queueing models with these types of workload are still being researched although some results have been published which are mostly related to asynchronous transfer mode (ATM) networks. It is expected that for a general arrival model, the performance computation would be much more complex. These are interesting and challenging research topics.

The distributions of outgoing messages from a single module are important for
those modules which take the messages as inputs. Therefore, another interesting topic is how to derive these distributions. Our work has focussed on certain time-averaged statistical performance metrics such as mean throughputs, mean queue wait times and mean queue length. In particular, the distributions or at least higher moments of queue length, queue wait time, busy period and idle period deserve further study.

9.2.3 New topics in stochastic processes

The stochastic processes of PEFSMs are different from those of traditional EF-SMs. The PEFSMs allow waiting for incoming messages. Unlike the conventional semi-Markov processes where the single-step transition probabilities are given, the transition probabilities of PEFSMs, with the exception of synchronous PEFSMs, are computed from the given workload descriptions. The computations involve solving matrix equations which may be very complex and sometimes insoluble. Therefore, further approximation methods are needed to deal with this problem.

9.2.4 On-line performance tuning

Bottleneck definition and identification provide insight on improving system performance. This forms the basis of on-line or real-time performance tuning. However, more studies are needed on how to remove a bottleneck efficiently. For example, the cost in reducing transition times has not been taken into account in our bottleneck definition and identification methodology.

In summary, the research reported in this thesis has identified and laid the ground work for several new research directions, some of which have practical
significance to industry.
Bibliography


BIBLIOGRAPHY


[Groo91] Jan Friso Groote. Specification and Verification of Real Time Systems in ACP. Published in Computer Notes (?).


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


Appendix A

Mathematical Background

A.1 Introduction

The semi-Markov process (SMP) has been extensively studied in the past decades and used widely in performance evaluations. Its theory is also the foundation of the analysis in this thesis. This appendix provides the definitions and some of the properties of MPs and SMPs. More detailed discussion of the material presented in this appendix is referred to [Howa71], [Moll89b] and [Allen90].

We extend their results to compute the limiting statistics of a state transition in a SMP because both states and transitions are of the same importance in our study.

The rest of this appendix is organized as follows. Section 2 gives the definition of Markov Process (MP). Section 3 provides the definition of continuous-time semi-Markov process (CSMP) and the theorems about the statistical properties of a CSMP.
A.2 Markov Process

A family of random variables \( \{X(t) | t \in T\} \) is called a stochastic process, where \( T \) is the index set of the process and \( X(t) \) is a random variable. Each value of \( X(t) \) is called a state of the stochastic process. The set of all possible values that the random variables \( X(t) \) can assume is called the state space of the process.

Definition A.1 (Markov process)
A stochastic process \( \{X(t), t \in T\} \) is a Markov process (MP), if for any set of \( n + 1 \) values, \( t_1 < t_2 < \cdots < t_n < t_{n+1} \), in the index set and any set of \( n + 1 \) states, \( s_1, s_2, \ldots, s_{n+1} \), there is

\[
Pr\{X(t_{n+1}) = s_{n+1} | X(t_1) = s_1, X(t_2) = s_2, \ldots, X(t_n) = s_n\} = Pr\{X(t_{n+1}) = s_{n+1} | X(t_n) = s_n\}.
\]

This definition means that the next state of a MP depends only on the current state.

Let \( p_{ij} \) be the single-step probability of transition \( ij \) which is defined as the probability that \( j \) will be the next state when a MP is in state \( i \) at present. By definition,

\[
p_{ij} = Pr\{X_{n+1} = j | X_n = i\}
\]

where \( X_n \) and \( X_{n+1} \) denote the current state and the next state, respectively.

Definition A.2 (Markov chain)
A Markov process is called a Markov chain if its state space is discrete.

Definition A.3 (irreducible)
A Markov chain is irreducible if every state is reachable from every other state.
Definition A.4 (positive recurrent)

A Markov chain is positive recurrent if for every state in the chain, the mean time to return to the state is finite.

Definition A.5 (aperiodic)

A Markov chain is aperiodic if for each state, there exists an integer \( k \) (\( k > 0 \)) for which there are transition sequences in the chain starting from the state and back to itself and these sequences traverse \( k, k + 1, k + 2, \ldots \) transitions respectively.

Definition A.6 (ergodic)

A Markov chain is ergodic if it is irreducible, recurrent nonnull and aperiodic.

We define the \textit{steady-state state probability} of state \( i \) in of a MP is the probability that the MP is seen at state \( i \) by an arbitrary observer after the MP runs for a very long period of time. This probability is denoted as \( \pi_i \).

Theorem A.1 (steady-state state probability)

If a Markov chain is ergodic, its steady-state state probabilities exist and they are equal to its stationary state probabilities which can be computed by solving the following matrix equation:

\[
\pi P = \pi \tag{A.1}
\]

where \( \pi = [\pi_i] \) is the vector of the steady-state state probabilities and \( \sum_i \pi_i = 1 \); \( P = [p_{ij}] \) is the matrix of the single-step transition probabilities.

\textbf{Proof:} See [Moll89b](p.132). \qed
A.3 Continuous-time Semi-Markov Process

Definition A.7 (continuous-time semi-Markov process)

If the successive state occupancies of a stochastic process are governed by the transition probabilities of a Markov process, but their stay in any state is described by a positive continuous variable which only depends on the current state, then the process is called a continuous-time semi-Markov process (CSMP). The Markov Process is called the embedded Markov process of the CSMP.

Let $\tau_{ij}$ be the time between the moment a CSMP enters state $i$ and the moment it enters the next state, say state $j$. $\tau_{ij}$ is called the holding time at state $i$ before the CSMP enters state $j$. $\tau_{ij}$ is also called the transition time from state $i$ to $j$. By the definition of CSMP, $\tau_{ij}$ is a positive continuous random variable. We denote the probability density function (p.d.f.) of this random variable as $h_{ij}(t)$.

Let $\tau_i$ denote the time the CSMP stays in state $i$ before the CSMP enters any other state. $\tau_i$ is called the holding time of the CSMP at state $i$. $\tau_i$ is also a continuous random variable. Let $h_i(t)$ be the p.d.f. of $\tau_i$. By [Howa71], $h_i(t)$ can be computed as

$$h_i(t) = \sum_j p_{ij}h_{ij}(t).$$

Therefore, the relation of the mean values of $\tau_i$ and $\tau_{ij}$ can be expressed

$$\bar{\tau}_i = \sum_j p_{ij} \bar{\tau}_{ij}.$$ 

Let $\bar{\tau}$ be the overall mean holding time of the CSMP at any state. $\bar{\tau}$ can be computed as

$$\bar{\tau} = \sum_i \pi_i \bar{\tau}_i.$$
A.3.1 Steady-state state probability

We define the steady-state state probability of state \( i \) in a CSMP, \( \pi_i \), as the probability that state \( i \) is seen at state change instants after the CSMP runs over a long period of time.

By the definition of CSMP, it is seen that the process at the state change instants in a CSMP is the embedded MP of the CSMP. Therefore, the steady-state state probabilities of the MP and the CSMP shall be the same which can be computed by solving Equation (A.1).

A.3.2 Limiting state lodging probability

For a CSMP, it is important and also very useful to know which state the process occupies at an arbitrary moment.

We define \( \phi_{ij}(t) \) is the probability that the process is seen in state \( j \) at time \( t \) given that it entered state \( i \) at time zero. \( \phi_{ij}(t) \) is called the state lodging probability in this thesis. \(^1\)

Theorem A.2 (limiting state lodging probability)

*If the embedded Markov chain of a CSMP is ergodic, the limiting state lodging probabilities exist and can be computed as

\[
\phi_{ij} = \lim_{t \to \infty} \phi_{ij}(t) = \frac{\pi_j \bar{\tau}_j}{\pi_k \bar{\tau}_k} = \frac{\pi_j \bar{\tau}_j}{\bar{\tau}} = \phi_j
\]

where \( \bar{\tau}_{ij} \) and \( \bar{\tau}_j \) are the mean values of \( \tau_{ij} \) and \( \tau_{ij} \), respectively.*

Proof: See [Howa71](p.595).

\( \square \)

From this theorem, it is seen that the limiting state lodging probability of a state is no longer related to its starting state of the CSMP. Therefore, \( \phi_j \) is

\(^1\)\( \phi_{ij}(t) \) is called an interval transition probability in [Howa71](p.694).
the probability that an arbitrary observer sees the CSMP is in state $j$ when the CSMP is in steady state.

### A.3.3 First passage time

Let $\bar{\theta}_{ij}$ be the mean first passage time from state $i$ to $j$ in a CSMP, and $\bar{\theta}_{ij}^2$ be the second moment of the first passage time from state $i$ to $j$.

**Theorem A.3 (First Passage Time Moments)**

When the CSMP is in steady-state, we can compute

$$\bar{\theta}_{ij} = \bar{\tau}_i + \sum_k p_{ik} \bar{\theta}_{kj} - p_{ij} \bar{\theta}_{jj}$$

and

$$\bar{\theta}_{ij}^2 = \bar{\tau}_i^2 + \sum_{k \neq j} p_{ik} (2 \bar{\tau}_{ik} \bar{\theta}_{kj} + \bar{\theta}_{kj}^2).$$

**Proof:** See [Howa71](p.735). $\square$

### A.3.4 State Recurrence

An instant entrance rate from state $i$ to $j$ is defined as that $e_{ij}(k|t)\Delta$, where $\Delta$ is very small, is the probability that the CSMP will enter state $j$ on its $k$th transition in the time interval $(t,t+\Delta)$ given that it made its zeroth transition into state $i$ at time zero. This instant entrance rate is denoted as $e_{ij}(k|t)$.

Let $e_{ij}$ be the limiting value of $e_{ij}(k|t)$, i.e.,

$$e_{ij} = \lim_{t \to \infty} e_{ij}(k|t).$$

**Theorem A.4 (limiting entrance rate)**

If the embedded Markov process is ergodic, $e_{ij}$ exists and is equal to

$$e_{ij} = \frac{\pi_j}{\bar{\tau}} = \pi_j$$
where \( \pi \) is the limiting state probability of state \( j \) for the imbedded Markov process and \( T \) is the overall mean time between transitions.

**Proof:** See [Howa71](p.725).

This theorem says that after a long period of time, \( e_{ij}(k|t) \) is not related to the starting state of the CSMP any more.

The recurrence time of state \( i \) is the first passage time from state \( i \) to \( i \) in a CSMP. Let \( \theta_{ii} \) denote this recurrent time.

**Theorem A.5 (mean recurrence time)**

*When the CSMP is in steady state, the mean recurrence time of state \( i \), \( \bar{\theta}_{ii} \), is equal to the reciprocal of the limiting entrance rate of state \( i \), i.e.,*

\[
\bar{\theta}_{ii} = \frac{1}{\bar{e}_i}.
\]

**Proof:** See [Howa71](p.735).

From the above two theorems, it is seen that the limiting entrance rate of a state in a CSMP is also the mean entrance rate of the state. We denote the latter using \( \bar{e}_i \).

**A.3.5 Transition recurrence**

Transitions in a CSMP are as important as states in our study. It is useful to know the statistics about the transitions of a CSMP. We extend the above results for transitions in the following.

We have defined the mean entrance rate to state \( i \) is \( \bar{e}_i \) when a CSMP in a steady-state.

At state \( i \), the CSMP may choose any state as its next state. By definition, the CSMP chooses the transition to state \( j \) with probability \( p_{ij} \). Therefore, the
mean entrance rate of the transition from state $i$ to state $j$ is $e_i \cdot p_{ij}$. Let $\bar{\eta}_{ij}$ denote this rate. We have

$$\eta_{ij} = \bar{e}_i p_{ij} \tag{A.2}$$

The *mean recurrence time of the transition* from state $i$ to state $j$ is the reciprocal of the mean entrance rate, i.e., $\frac{1}{\eta_{ij}}$.

We use Figure A.1 to illustrate these relationships.

![Figure A.1: The occupancies of state $i$ in time axis](image)

---

: the transition time from state $i$ to the next state
Appendix B

Computation of $\zeta_{ij}^2$

Suppose $X$ and $Y$ are random variables. Let $E[X]$ be the mean of $X$, $\text{Var}[X]$ be the variance of $X$, and $\text{Cov}[X,Y]$ be the covariance of $X$ and $Y$.

Lemma B.1  

2. $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y] + 2\text{Cov}[X,Y]$;

3. $\text{Var}[X] = E[X^2] - (E[X])^2$.


Proof:
See [Allen90](p52).

$E[X^2]$ is called the second moment of $X$. We use $\overline{X^2}$ to denote $E[X^2]$.

Now we can compute the second moment of the service time $\zeta_{ij}$. By definition,

$$
\zeta_{ij} = \begin{cases} 
\theta_{ii} & i = j \\
\theta_{ij} + \zeta_{ji} & i \neq j 
\end{cases}
$$
We compute \( \overline{c}_{ij}^2 \) in two cases as follows:

1. If \( i = j \), then

\[
\overline{c}_{ij}^2 = \overline{\sigma}_{ii}^2
\]

where \( \overline{\sigma}_{ii}^2 \) can be computed directly from the given probability density function (p.d.f.) of the service time of transition \( ii, \sigma_{ii}(t) \).

2. If \( i \neq j \), then

\[
\overline{c}_{ij}^2 = E[(\sigma_{ij} + \psi_{ji})^2]
\]

\[
= \overline{\sigma}_{ij}^2 + \overline{\psi}_{ji}^2 + 2 \cdot E[\sigma_{ij} \cdot \psi_{ji}].
\]

Since \( \sigma_{ij} \) and \( \psi_{ji} \) are uncorrelated for the reason stated in Chapter 5,

\[
\overline{c}_{ij}^2 = \overline{\sigma}_{ij}^2 + \overline{\psi}_{ji}^2 + 2 \cdot \sigma_{ij} \cdot \psi_{ji} \tag{B.1}
\]

where \( \overline{\sigma}_{ij}^2 \) can be computed from the provided p.d.f. of \( \sigma_{ij} \) and \( \overline{\psi}_{ji}^2 \) can be computed using Theorem A.3.
Appendix C

An Estelle specification for sliding window protocol

This appendix contains an Estelle specification of the sliding-window protocol. This specification is modified based on Tom Blumer's specification in 1986 which was ftp-ed from est-specsudel.edu. The module graph is in Figure C.1.

![Module relation graph of sliding window protocol](image)

Figure C.1: Module relation graph of sliding window protocol

The specification in Estelle is as follows:

```estelle
specification sliding_window_protocol;
  default individual queue;
```

219
Appendix: an Estelle specification of the sliding-window protocol

{---------------------------------------------------------------

Origin: November 3, 1986
Tom Blumer
Protocol Development Corporation
1330 Beacon St.
Brookline, MA 02146 USA
(617)354-8558
Modified by: Sijian Zhang, Oct. 7, 1992
---------------------------------------------------------------}

type seq_type = integer;  { sequence number type, must be >= 0 }
  user_data_type = ...;
  DTPDU_type = record
    seq: seq_type;
    msg: user_data_type;
  end;
  AKPDU_type = record
    seq: seq_type;
  end;

channel user_chan(user, provider);
  by user:
    Data_Request(data : user_data_type);
  by provider:
    Data_Indication(data : user_data_type);

channel comm_chan(DT_sender, DT_receiver);
  by DT_sender:
    DT(PDU : DTPDU_type);
  by DT_receiver:
    AK(PDU : AKPDU_type);

channel timer_chan(user, provider);
  by user:
    timer_request(seq : seq_type);
    timer_cancel(seq : seq_type);
  by provider:
    timer_response(seq : seq_type);

module user_head systemprocess;
Appendix: an Estelle specification of the sliding-window protocol

```plaintext
ip U : user_chan(user);
end;

body transmitter_user for user_head; external;

body receiver_user for user_head; external;

module timer_head systemprocess;
  ip T : timer_chan(provider);
end;

body timer for timer_head; external;

module transmitter_head systemprocess;
  ip U : user_chan(provider);
  CT : comm_chan(DT_sender);
  T : timer_chan(user);
end;

{ transmitter module body }
body transmitter for transmitter_head;

const transmitter_window_size = any integer;

state SENDING;

{ save user data in buffer until acknowledgment }
procedure buf_save(s : seq_type; d : user_data_type); primitive;

{ free user data buffer entry after acknowledgment }
procedure buf_free(s : seq_type); primitive;

{ retrieve user data entry from buffer }
function buf_retrieve(s : seq_type) : user_data_type; primitive;

{ construct a DT PDU from the user data and sequence number }
function PDU_DT(s : seq_type; d : user_data_type)
  : DTPDU_type; primitive;
```

var
Appendix: an Estelle specification of the sliding-window protocol

Lowest_Unacked : seq_type;
Highest_Sent : seq_type;
TWS : integer;

initialize
to SENDING
begin
  Lowest_Unacked := 1;
  Highest_Sent := 0;
  TWS := transmitter_window_size;
end;

trans

{ transmit while window not full }
T1: from SENDING to same
when U.Data_Request
  provided Highest_Sent - Lowest_Unacked < TWS
  begin
    Highest_Sent := Highest_Sent + 1;
    output T.timer_request(Highest_Sent);
    output CT.DT(PDU_DT(Highest_Sent, data));
    buf_save(Highest_Sent, data);
  end;

{ receive acknowledgment }
T2: from SENDING to same
when CT.AK
  provided (PDU.seq >= Lowest_Unacked) and (PDU.seq <= Highest_Sent)
  var s : seq_type;
  begin
    for s := Lowest_Unacked to PDU.seq do
      begin
        output T.timer_cancel(s);
        buf_free(s);
      end;
    Lowest_Unacked := PDU.seq + 1;
  end;

{ receive ack not in window }
Appendix: an Estelle specification of the sliding-window protocol

T3: provided otherwise begin
   { ignore this ack }
end;

{ Timer response }
T4: from SENDING to same
   when T.timer_response
       provided (seq >= Lowest_Unacked) and
       (seq <= Highest_Sent)
       var s : seq_type;
       begin
           for s := seq to Highest_Sent do
               begin
                   output T.timer_cancel(s);
                   output CT.DT(PDU_DT(s, buf_retrieve(s)));
                   output T.timer_request(s);
               end;
       end;
end;

T5: provided otherwise begin
   { ignore t response for sequence number outside the window. This can happen when an AK
   arrives just when a timer response occurs. }
end;
end; { transmr }

module receiver_head systemprocess;
   ip U : user_chan(provider);
   CR : comm_chan(DT_receiver);
end;

{ receiver module body }
body receiver for receiver_head;

   const receiver_window_size = any integer;
Appendix: an Estelle specification of the sliding-window protocol

state RECEIVING;

{ construct an AK PDU, given the sequence number }
function PDU_AK(s : seq_type) : AKPDU_type; primitive;

{ retrieve the PDU of sequence number s from the buffer. 
  If it is not in the buffer, return a PDU with seq number set to 0 }
function PDU_retrieve(s : seq_type) : DTPDU_type; primitive;

{ Save the PDU in the buffer }
procedure PDU_save(PDU : DTPDU_type); primitive;

{ returns true if the PDU is corrupted }
function corrupted(PDU : DTPDU_type) : boolean; primitive;

{ Return the user data from the given PDU }
function user_data(p : DTPDU_type) : user_data_type; primitive;

var
  Next_Required : seq_type;
  Highest_Received : seq_type;
  RWS : integer;

initialize
to RECEIVING
begin
  Next_Required := 1;
  Highest_Received := 0;
  RWS := receiver_window_size;
end;

trans

{ receive message in window }
from RECEIVING to same
when CR.DT
  provided (PDU.seq >= Next_Required) and
  (PDU.seq < Next_required + RWS) and
  not corrupted(PDU)
  var s : seq_type;
Appendix: an Estelle specification of the sliding-window protocol

\[ \text{tpdu : DTPDU\_type; done : boolean; begin} \]
\[ \text{PDU\_save(PDU);} \]
\[ \text{s := Next\_Required; done := false; \{ Retrieve each PDU from buffer and send it to the user. Stop when we reach the first gap in the buffer, i.e., the first PDU that has not been received. PDU\_retrieve returns a PDU with sequence number 0 if the desired PDU is not in the buffer \}} \]
\[ \text{repeat} \]
\[ \text{tpdu := PDU\_retrieve(s);} \]
\[ \text{if tpdu.seq = s then \{ extract user data from PDU and send to user \}} \]
\[ \text{output U.Data\_Indication(user\_data(tpdu)); s := s + 1;} \]
\[ \text{end} \]
\[ \text{else \{ reached gap in buffer \}} \]
\[ \text{done := true;} \]
\[ \text{until done;} \]
\[ \text{Next\_Required := s;} \]
\[ \text{output CR.AK(PDU\_AK(Next\_Required - 1))}; \]
\[ \text{end;} \]
\[ \{ \text{receive message not in window} \} \]
\[ \text{provided otherwise \{ receiver \}} \]
\[ \text{end; \{ receiver \}} \]

\[ \text{modvar transmitter\_instance : transmitter\_head;} \]
\[ \text{receiver\_instance : receiver\_head;} \]
\[ \text{transmitter\_user\_instance, receiver\_user\_instance : user\_head;} \]
\[ \text{timer\_instance : timer\_head;} \]
Appendix: an Estelle specification of the sliding-window protocol

initialize
begin
  init transmitter_user_instance with transmitter_user;
  init receiver_user_instance with receiver_user;
  init transmitter_instance with transmitter;
  init receiver_instance with receiver;
  init timer_instance with timer;
  connect transmitter_user_instance.U to transmitter_instance.U;
  connect receiver_user_instance.U to receiver_instance.U;
  connect transmitter_instance.CT to cm_instance.CR;
  connect transmitter_instance.T to timer_instance.T;
end;

end.
Appendix D

An Estelle specification of the hot-potato protocol

This appendix contains an Estelle specification of the *hot-potato* protocol.

```
SPECIFICATION hot_potato SYSTEMACTIVITY;

  CHANNEL toss (thrower, catcher);
    BY thrower:
      POTATO (thrower_num: integer);
    BY catcher:
      POTATO (catcher_num: integer);

  CHANNEL pass (left_hand, right_hand)
    BY left_hand:
      POTATO;
    BY right_hand:
      POTATO;

  MODULE player ACTIVITY (player_num: integer);
    IP
      LEFT_SIDE : toss(catcher) INDIVIDUAL QUEUE;
      RIGHT_SIDE: toss(thrower) INDIVIDUAL QUEUE;
    END; { MODULE player }
```
Appendix: an Estelle specification of the hot-potato protocol

BODY player_process FOR player;

STATE s0, s1;

IP { internal interact. points }
   LEFT_HAND : pass(left_hand);
   RIGHT_HAND: pass(right_hand);

VAR
   the_player : integer;
   started    : boolean;

INITIALIZE
TO s0
BEGIN
   the_player := 1;
   connect LEFTHAND to RIGHTHAND;
   the_player := 1;
END;

TRANS

PROVIDED (player_num=1 AND started = false)
BEGIN
   OUTPUT RIGHT_HAND.POTATO;
   started := true;
END;

TRANS

FROM s0
TO s1
WHEN LEFT_SIDE.POTATO
BEGIN
   { ... }
   OUTPUT LEFT_HAND.POTATO;
END;

TRANS
FROM s1
TO s0
WHEN RIGHT_HAND.POTATO;
BEGIN
  if ((the_player+1)<NumPlayer) then
    OUTPUT RIGHT_SIDE.POTATO(the_player+1);
  else
    OUTPUT RIGHT_SIDE.POTATO(1);
END;
END; { BODY player_process }

CONSTANT
  NumPlayer = ...;

MODVAR
  Players array [1..NumPlayer] of player;

VAR
  i: integer;

INITIALIZE
BEGIN
  all i: 1..NumPlayer do
  BEGIN
    INIT Players[i] WITH player_process (i);
  END;

  all i: 1..NumPlayer do
  BEGIN
    CONNECT Players[i].RIGHT_SIDE to Players[i].LEFT_SIDE;
  END;

  CONNECT Players[1].LEFT_SIDE to Players[NumPlayer].RIGHT_SIDE;
END;

END. { hot_potato }
Appendix E

An application example of HyPEFSM

The example described in this appendix is the circuit-switched *multi-stage interconnection network* (MIN) with a hold protocol [HaPa93].

A MIN is a matrix of small interconnected crossbars used in multiprocessor computers or network switches. MINs can be classified as blocking and non-blocking, and synchronous and asynchronous. Some well-known MINs are *banyan networks, Benes networks* and *Clos networks*.

The MIN to be studied in this appendix belongs to a subclass of banyan networks called *delta networks*. The MIN is asynchronous with a *hold protocol* which specifies the procedure for a processor to access a memory module. The finite state machine diagram of the hold protocol is given in Figure E.1.

State 0 is the starting state (often called the idle state) of the switch. The switch is in state 0 when there is no memory access request. Initially, the switch is in state 0, and it enters this state each time it finishes serving a request. We assume that the memory access requests from processors arrive asynchronously in a Poisson pattern with parameter $\lambda$. The requests will be queued on arrival if
the switch is not in state 0. The service policy for requests is first-come first-serve (FCFS) and the queuing discipline is first-in first-out (FIFO).

When the switch accepts a request, it enters state 1 immediately. At state 1, the switch checks if link 1 is free or not. If link 1 is free, the switch acquires the link and enters state 3 \((= 2 \times 1 + 1)\); otherwise, it enters the blocking state 2 \((= 2 \times 1)\), waiting for the release of the link. As soon as link 1 is free, the switch acquires it and enters state 3. In state 3, the switch attempts to acquire link 2 using the same procedure as before. This process is repeated until all required links have been acquired.

We assume that there are a total of \(J\) links to be acquired before a processor can access the memory. As soon as all \(J\) links have been obtained and the switch enters state \(2 \times J + 1\), the actual memory access begins. After the access is completed, the switch goes back to state 0 to serve the next request.

Therefore, the FSM of the hold protocol has a total of \((2J + 2)\) states and \((3J + 2)\) transitions. This FSM with the arrival and service model described above is a HyPEFSM that has only one asynchronous state (state 0) and one asynchronous transition (from state 0 to state 1). The other states and transitions are synchronous.

We assume that the checking of the status of link \(i\) \((1 \leq i \leq J)\) takes an
average time $h_i$. Let $b_i$ be the probability that link $i$ is not free and $1/\gamma_i$ be the mean time that the switch stays in state $2i$ waiting for link $i$. Finally, we assume that the memory access time takes $1/\mu$ on average.

We translate the above information into the input data required by the HyPEFSM:

- $P = [p_{ij}]$ where
  \[
  p_{ij} = \begin{cases} 
  1 & \text{if } (i = 0 \text{ and } j = 1) \text{ or } (i = 2J + 1 \text{ and } j = 0) \\
  b_n & \text{if } i = 2n - 1 \text{ and } j = 2n \text{ and } n = 1, 2, \ldots, J \\
  1 - b_n & \text{if } i = 2n - 1 \text{ and } j = 2n + 1 \text{ and } n = 1, 2, \ldots, J \\
  0 & \text{otherwise}
  \end{cases}
  \] (E.1)

- $s_{ij} = 0$ except for $s_{2J+1,0} = 1/\mu$.

- $w_{r_{2n-1,2n}} = w_{r_{2n-1,2n+1}} = h_n$ where $n = 1, 2, \ldots, J$ and $w_{r_{2J+1,0}} = 0$.

$w_{r_{01}}$ is unknown and it can be computed using the method introduced in Chapter 6.

We are interested in the performance metrics such as the probability that a request is served on arrival without waiting (i.e., $p_{0,0}$), the mean queue wait time of memory access requests (i.e., $\bar{W}_{r,0}$), and the mean overhead for a request before memory is accessed (i.e., $\bar{W}_{r,0} + \psi_{1,2J+1}$).

The first passage times from state $i$ ($i = 2J + 1, 2J, \ldots, 2, 1$) to state 0 are computed in the following using Formula (A.3):

\[
\psi_{\theta_{2J+1,0}} = \psi_{\theta_{2J+1}} + \sum_k p_{2J+1,k} \bar{\theta}_{k0} - p_{2J+1,0} \bar{\theta}_{00} \quad (E.2)
\]

\[
= \frac{1}{\mu} \quad (E.3)
\]

\[
\psi_{\theta_{2J,0}} = \psi_{\theta_{2J}} + \sum_k p_{2J,k} \bar{\theta}_{k0} - p_{2J,0} \bar{\theta}_{00} \quad (E.4)
\]

\[
= \frac{1}{\gamma_J} + \frac{1}{\mu} \quad (E.5)
\]
Appendix: an application example of HyPEFSM

\[ \psi_{2J-1,0} = \psi_{2J-1} + \sum_{k} p_{2J-1,k} \theta_{k0} - p_{2J-1,0} \theta_{00} \]  
(E.6)

\[ = h_{J} + \frac{b_{J}}{\gamma_{J}} + \frac{1}{\mu}; \]  
(E.7)

\[ \ldots \]  

\[ \psi_{10} = \sum_{i=1}^{J} h_{i} + \sum_{i=1}^{J} \frac{b_{i}}{\gamma_{i}}. \]  
(E.8)

\[ \psi_{10} \] is in fact the mean holding time of the switch by a request after service on the request has started.

The mean time spent on obtaining all the links required is

\[ \psi_{1,2J+1} = \psi_{10} - \psi_{2J+1,0} = \sum_{i=1}^{J} h_{i} + \sum_{i=1}^{J} \frac{b_{i}}{\gamma_{i}}. \]

Applying Equation (6.7), we get

\[ p_{0,0} = 1 - \lambda_{01} (\psi_{01} + \psi_{10}) \]  
(E.10)

\[ = 1 - \lambda \left( \sum_{i=1}^{J} h_{i} + \sum_{i=1}^{J} \frac{b_{i}}{\gamma_{i}} \right), \]  
(E.11)

where \( \lambda_{01} = \lambda \). \( p_{0,0} \) is the probability that a memory access request will be served immediately upon its arrival.

To compute the mean queue wait time, we need to obtain \( \overline{\psi_{10}} \) first. Applying Theorem A.3, we derive

\[ \overline{\psi_{2J+1,0}} = \overline{\psi_{2J+1}} + \sum_{k \neq 0} p_{2J+1,k} (2 \cdot \psi_{2J,k} \cdot \overline{\theta_{k0}} + \overline{\psi_{k0}^{2}}) \]  
(E.12)

\[ = \overline{\psi_{2J+1,0}}; \]  
(E.13)

\[ \overline{\psi_{2J,0}} = \overline{\psi_{2J}} + \sum_{k \neq 0} p_{2J,k} (2 \cdot \psi_{2J,k} \cdot \overline{\theta_{k0}} + \overline{\psi_{k0}^{2}}) \]  
(E.14)

\[ = \overline{\psi_{2J}} + 2 \cdot p_{2J,2J+1} \cdot \overline{\psi_{2J,2J+1}} \cdot \overline{\theta_{2J+1,0}} + \overline{\psi_{2J+1,0}} \]  
(E.15)

\[ = \overline{\psi_{2J}} + 2 \cdot p_{2J,2J+1} \cdot \frac{1}{\gamma_{J}} \cdot \overline{\psi_{2J+1,0}} + \overline{\psi_{2J+1,0}}; \]  
(E.16)
Appendix: an application example of HyPEFSM

\[ \bar{\varphi_{2J-1,0}} = \bar{\psi_{2J-1}} + \sum_{k \neq 0} p_{2J,k} (2 \cdot \bar{\psi_{2J-1,2J}} \cdot \bar{\varphi_{k0}} + \bar{\psi_{k0}}) \quad (E.17) \]

\[ = \bar{\psi_{2J-1}} + p_{2J-1,2J} \cdot (2 \bar{\varphi_{2J-1,2J}} \cdot \bar{\psi_{2J,0}} + \bar{\psi_{2J,0}}) \quad (E.18) \]

\[ + p_{2J-1,2J+1} \cdot (2 \bar{\varphi_{2J-1,2J+1}} \cdot \bar{\psi_{2J+1,0}} + \bar{\psi_{2J+1,0}}); \quad (E.19) \]

\[ \bar{\varphi_{10}} = \bar{\varphi_{11}} + p_{12} \cdot (2 \bar{\psi_{12}} \cdot \bar{\varphi_{20}} + \bar{\psi_{20}}) \quad (E.20) \]

\[ + p_{13} \cdot (2 \bar{\psi_{13}} \cdot \bar{\varphi_{30}} + \bar{\psi_{30}}). \quad (E.21) \]

\[ \bar{\psi_{10}} = \bar{\psi_{11}} + p_{12} \cdot (2 \bar{\psi_{12}} \cdot \bar{\varphi_{20}} + \bar{\psi_{20}}) \quad (E.22) \]

It can be seen that \( \bar{\varphi_{10}} \) can be computed recurrently using the above recurrent equations. The mean queue wait time is obtained as

\[ \bar{W}_{q,0} = \frac{\lambda \bar{\varphi_{10}}}{2(1 - \rho_0)} = \frac{\lambda \bar{\varphi_{10}}}{2 \cdot p_{0,0}}. \quad (E.23) \]

With (E) and (E.23), the mean overhead of a request in the switch, i.e., \( \bar{W}_{q,0} + \bar{\psi_{1,2J+1}} \), can be obtained.

In the following, we examine a simple case with only one link (i.e., \( J = 1 \)) in the switch. We assume that the checking time, the blocking times and the memory accessing time are all exponentially distributed.

In this case,

\[ \bar{\psi_{10}} = h_1 + \frac{b_1}{\gamma_1} + \frac{1}{\mu} \]

and

\[ \bar{\varphi_{30}} = \frac{2}{\mu^2}; \]

\[ \bar{\varphi_{23}} = \frac{2}{\gamma_1^2}; \]

\[ \bar{\varphi_{22}} = \frac{2}{\gamma_1^2}; \]

\[ \bar{\varphi_{13}} = \frac{2}{h_1^2}; \]

\[ \bar{\varphi_{12}} = \frac{2}{h_1^2}; \]
Therefore,
\[
\overline{w_{r_1}}^2 = \frac{2}{\gamma_1^2}.
\]

\[
\overline{w_{\theta_{20}}^2} = \frac{2}{\mu^2} 
\]

\[
\overline{w_{\theta_{20}}^2} = 2\left(\frac{1}{\gamma_1^2} + \frac{1}{\mu} + \frac{1}{\mu^2}\right) 
\]

\[
\overline{w_{\theta_{10}}^2} = 2\left(h_1^2 + \frac{h_1}{\mu} + \frac{1}{\mu^2}\right) + 2\frac{b_1}{\gamma_1}(h_1 + \frac{1}{\mu} + \frac{1}{\gamma_1}) 
\]

Using the information obtained above, we rewrite the mean queue wait time as

\[
\overline{W_{\pi,0}} = \frac{\lambda \overline{w_{\theta_{10}}^2}}{2 \cdot p_{0,0}} 
\]

\[
= \frac{\lambda((h_1^2 + \frac{h_1}{\mu} + \frac{1}{\mu^2}) + \frac{b_1}{\gamma_1}(h_1 + \frac{1}{\mu} + \frac{1}{\gamma_1}))}{1 - \lambda(h_1 + \frac{b_1}{\gamma_1} + \frac{1}{\mu})} 
\]

With the above formula, we can study the mean queue wait time vs. the arrival rate. Figure E.2 shows the relation for \( h_1 = 0.001 \text{second}, \gamma_1 = 0.001 \text{second}, \mu = 0.01 \text{second} \) and \( b_1 = 10\% \). It can be seen from the figure that the mean queue wait time grows rapidly after the mean arrival rate reaches 80 requests/second.

Figure E.3 shows the case when \( h_1 = 0.001 \text{second}, \gamma_1 = 0.001 \text{second}, \mu = 0.01 \text{second} \) and \( \lambda = 50 \text{requests/second} \). It can be seen from the figure that the growth of the mean queue wait time vs. the blocking probability is approximately linear.
Appendix: an application example of HyPEFSM

Figure E.2: Mean queue wait time vs. arrival rate

Figure E.3: Mean queue wait time vs. blocking probability
Appendix F

Bottleneck identification results

In this appendix, we report the results of the FDDI MAC Transmitter. The FSM of the transmitter is shown in Figure F.1.

![Figure F.1: The FSM of FDDI MAC Transmitter](image)

We assume that all the incoming messages are asynchronous and the arrivals of each class follow a Poisson pattern. Therefore, the FSM with these assump-
The distributions of the service times of the transitions associated with each class of incoming messages are exponential. The mean service times are given in Column 2 of Table F.1. The mean arrival rates of each class are given in Column 3 of the table. Columns 4, 5 and 6 are the simulation results.

<table>
<thead>
<tr>
<th>transition number</th>
<th>service time (sec)</th>
<th>arrival rate (msgs/sec)</th>
<th>transition probability</th>
<th>transition weight</th>
<th>average queueing time reduction (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0614</td>
<td>0.0274</td>
<td>0.01871</td>
</tr>
<tr>
<td>1</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0613</td>
<td>0.0053</td>
<td>0.01905</td>
</tr>
<tr>
<td>2</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0613</td>
<td>0.0063</td>
<td>0.01779</td>
</tr>
<tr>
<td>3</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0614</td>
<td>0.0128</td>
<td>0.02032</td>
</tr>
<tr>
<td>4</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0613</td>
<td>0.0163</td>
<td>0.02290</td>
</tr>
<tr>
<td>5</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0613</td>
<td>0.0108</td>
<td>0.01954</td>
</tr>
<tr>
<td>6</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0104</td>
<td>0.01619</td>
</tr>
<tr>
<td>7</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0046</td>
<td>0.00580</td>
</tr>
<tr>
<td>8</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0046</td>
<td>0.00534</td>
</tr>
<tr>
<td>9</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0046</td>
<td>0.00766</td>
</tr>
<tr>
<td>10</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0046</td>
<td>0.00748</td>
</tr>
<tr>
<td>11</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0102</td>
<td>0.0027</td>
<td>0.00856</td>
</tr>
<tr>
<td>12</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0204</td>
<td>0.0091</td>
<td>0.01092</td>
</tr>
<tr>
<td>13</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0204</td>
<td>0.0021</td>
<td>0.00191</td>
</tr>
<tr>
<td>14</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0204</td>
<td>0.0043</td>
<td>0.00799</td>
</tr>
<tr>
<td>15</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0204</td>
<td>0.0054</td>
<td>0.01014</td>
</tr>
<tr>
<td>16</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0503</td>
<td>0.0225</td>
<td>0.01933</td>
</tr>
<tr>
<td>17</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0503</td>
<td>0.0225</td>
<td>0.02091</td>
</tr>
<tr>
<td>18</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0504</td>
<td>0.0134</td>
<td>0.01664</td>
</tr>
<tr>
<td>19</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0691</td>
<td>0.0309</td>
<td>0.02417</td>
</tr>
<tr>
<td>20</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0692</td>
<td>0.0145</td>
<td>0.01690</td>
</tr>
<tr>
<td>21</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0692</td>
<td>0.0122</td>
<td>0.01999</td>
</tr>
<tr>
<td>22</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0592</td>
<td>0.0291</td>
<td>0.02366</td>
</tr>
<tr>
<td>23</td>
<td>0.025</td>
<td>10.0000</td>
<td>0.0653</td>
<td>0.0173</td>
<td>0.01773</td>
</tr>
</tbody>
</table>

a the steady-state transition probability.

b The new average queue wait time is measured by decreasing the mean service time of the corresponding transition by 0.005 second. The reduction is equal to the original value minus the new value.

Table F.1: A simulation result

From the table, we can see that reducing the service time of the transition
with the largest weight causes the largest reduction in the mean queue wait time of that class of incoming messages. This result confirms the analysis given in this section.
Appendix G

T-test results of a LAPB implementation

This appendix reports the t-test results for a LAPB implementation.

The finite state machine (FSM) of LAPB is referred to in Figure 3.2. However, the implementation under test (IUT) is only a subset of the LAPB protocol. The IUT is treated as a black box in testing. The testing architecture is the same as the one shown in Figure 8.12. The interface modules are also similar but a little more complicated than those used to test abstract protocols (see Chapter 8). Here we have to use the service primitives such as \texttt{lapb\_down()} and \texttt{lapb\_up()} provided by the LAPB implementation in the interface modules.

Table G.1 shows the testing results of some transitions. The first three columns are starting states, ending states and I/O events of transitions. The last three columns record the results from measurement and testing. Columns 4 and 5 are the measurement results and column 6 are the t-test results. All the times in Columns 4, 5 and 6 are in milliseconds (ms).

The total overhead of network transmission and the interface with the IUT is about 0.2 ms. Deducing this overhead from each result in columns 5 and 6,
### Appendix: T-test results of a LAPB implementation

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>I/O</th>
<th>Measured Service Time&lt;sup&gt;a&lt;/sup&gt;</th>
<th>Measured Response Time&lt;sup&gt;b&lt;/sup&gt;</th>
<th>T-test Result&lt;sup&gt;c&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISCONNECTED</td>
<td>SABM.SENT</td>
<td>L.CONreq/SABM.CTRL</td>
<td>54.4</td>
<td>49.6</td>
<td>54.1</td>
</tr>
<tr>
<td>DISCONNECTED</td>
<td>DISCONNECTED</td>
<td>SABM/L.CONInd</td>
<td>49.4</td>
<td>39.6</td>
<td>49.9</td>
</tr>
<tr>
<td>DISCONNECTED</td>
<td>DATA.TRANSFER</td>
<td>L.CONresp/UA.CTRL</td>
<td>56.8</td>
<td>49.5</td>
<td>56.3</td>
</tr>
<tr>
<td>DATA.TRANSFER</td>
<td>DATA.TRANSFER</td>
<td>LCTRL/L.DATind</td>
<td>78.8</td>
<td>69.6</td>
<td>80.9</td>
</tr>
<tr>
<td>DATA.TRANSFER</td>
<td>DISCONNECTED</td>
<td>DISC.CTRL/UA.CTRL</td>
<td>39.4</td>
<td>29.5</td>
<td>38.2</td>
</tr>
</tbody>
</table>

<sup>a</sup>The service time of each transition is measured when the implementation is treated as a white box.

<sup>b</sup>The response time is obtained by computing the difference between the message sending time and the response time. When there are two or more responses for one input such as DISC/(UA,L.DISC.ind), only one of them is chosen throughout the test.

<sup>c</sup>The error tolerance is 5%.

Table G.1: Testing results of the LAPB implementation

we observe that the t-test results in column 6 are closer to the measured service times of the same row in column 4 than the measured response times of the same row in column 5.