Dynamic routing in circuit-switched non-hierarchical networks

Eshragh, Nadereh

Dynamic Routing in Circuit-Switched Non-hierarchical Networks

Nadereh Eshragh

B. Sc., M. Sc.

School of Engineering and Applied Science

University of Durham

November 1989

A thesis submitted for the degree of

Doctor of Philosophy of the University of Durham

Abstract

Dynamic Routing in Circuit-Switched Non-hierarchical Networks

Nadereh Eshragh

This thesis studies dynamic routing in circuit-switched non-hierarchical networks based on learning automata algorithms. The application of a mathematical model for a linear reward penalty algorithm is explained. Theoretical results for this scheme verified by simulations shows the accuracy of the model.

Using simulation and analysis, learning automata algorithms are compared to several other strategies on different networks. The implemented test networks may be classified into two groups. The first group are designed for fixed routing and in such networks fixed routing performs better than any dynamic routing scheme. It will be shown that dynamic routing strategies perform as well as fixed routing when trunk reservation is employed. The second group of networks are designed for dynamic routing and trunk reservation deteriorates the performance. Comparison of different routing algorithms on small networks designed to force dynamic routing demonstrates the superiority of automata under both normal and failure conditions.

The thesis also considers the instability problem in non-hierarchical circuit-switched networks when dynamic routing is implemented. It is shown that trunk reservation prevents instability and increases the carried load at overloads. Finally a set of experiments are performed on large networks with realistic capacity and traffic matrices. Simulation and analytic results show that dynamic routing outperforms fixed routing and trunk reservation deteriorates the performance at low values of overload. At high overloads, optimization of trunk reservation is necessary for this class of networks. Comparison results shows the improved performance with automata schemes under both normal and abnormal traffic conditions. The thesis concludes with a discussion of proposed further work including expected developments in Integrated Service Digital Networks.

Acknowledgement

I would like to express my thanks and appreciation to my supervisor Professor Philip Mars for his expert guidance, encouragement and interest throughout this project. I would also like to thank the Science and Engineering Research Council (SERC) for financial support. My thanks to Mr Brian Lander and the staff at Durham computer centre for their efficient service without which the preparation of this thesis would have been much harder. Finally I would like to thank my husband Mehdi for his encouragement and understanding while I was involved with this work.

Declaration

I hereby declare that this thesis is a record of work undertaken by myself, that it has not been the subject of any previous application for a degree, and that all sources of information have been duly acknowledged.

Nadereh Eshragh, November 1989

Copyright

To Sima

Contents

List of Figures

List of Symbols

A	traffic intensity
AAR	Automatic Alternate Routing
AT&T	American Telephone and Telegraph
α	automaton's action set
a	reward parameter
B	link blocking probability
BT	British Telecom
b	penalty parameter
β	automaton's input set
c	link capacity
C	carried proportion of the offered traffic per link
CCS	Common Channel Signaling
DAR	Dynamic Alternative Routing
DNHR	Dynamic Nonhierarchical Routing
DCR	Dynamically Controlled Routing
FC	Free Circuits
FDM	Frequency Division Multiplexed

FR	Fixed Routing
ϕ	state set
GOS	Grade Of Service
K	carried load on a link
L	blocking probability of a two link path
LD	Load
LA	Learning Automata
LBA	Least Busy Alternative
L_{R-I}	Linear Reward Inaction
L_{R-P}	Linear Reward Penalty
$L_{R-\epsilon P}$	Linear Reward ϵ-Penalty
λ	average arrival rate
M	number of two link alternate paths
m	number of trunks allowed to be accessed by overflow traffic
N	number of nodes
NAC	number accepted calls
NL	number of lost calls
$\frac{1}{\mu}$	call holding time
ν	total offered load to a link
O-D pair	Origin Destination pair
OVL	Overload
PDC	Proportion of Directly routed Calls

PRO	Proportional Routing
P	probability
Q	link availability
RR	Random Routing
SRI	Stochastic Residual Index
TRP	Trunk Reservation Parameter
TDM	Time Division Multiplexed
TSMR	Trunk Status Map Routing
TL	total load
U	a random number in the range (0,1)
Z	overal blocking probability
z	penalty probability

Chapter One

Introduction

In this work we study dynamic routing strategies in circuit-switched nonhierarchical networks using both analytical and simulation models. Recent advances in telecommunications such as the introduction of stored program control networks consisting of electronic switching centers interconnected by common channel signaling links allows the implementation of dynamic routing strategies. These strategies are intended to take into account real-time conditions in order to provide better network performances.

1.1 - Circuit-switching

Circuit-switching is the technology used worldwide over the years for telephony. This kind of transmission requires an end to end path to be set up from source to destination before any communication can take place. The end to end physical path consists of a number of transmission links or circuits connected by intermediate circuit

switches. The links could consist of time slots in a time-division multiplexed (TDM) system or frequency division multiplexed (FDM) system.

While circuit-switching technique is mainly used to transmit voice, packet-switching is used for data transmission. In this case data messages are blocked into shorter units called packets, which are then transmitted from source to destination along some routing path. Two techniques are used for this purpose: virtual circuit or connection oriented transmission and datagram or connectionless transmission. In virtual circuit transmission a path is first set up end to end through the network. Packets then traverse the network through the chosen path and arrive at the destination in the sequence in which they were transmitted. In the datagram method, no initial connection is set up and packets may follow independent routes between the source and the destination.

One characteristic of circuit-switched traffic is that no queuing or waiting is allowed at any intermediate nodes along the circuit. As a result, if an incoming call cannot find a connection to its destination node, it will be blocked and it has to re-try again for a connection. Therefore an important performance measure in circuit-switched networks is to have a low blocking probability which is defined as the probability that a call attempting to access the network, is turned away. Blocking probability or Grade Of Service (GOS) depends upon a number of different functions, specifically the network topology, offered traffic, link capacities and routing policies implemented. In a packet switching network, packets queue at each node. In the case of packet switching traffic, the used performance measure is packet-time-delay which

is the average time delay that takes for a packet to reach destination.

1.1.1 - Signaling messages in circuit-switched networks

The operation of a circuit switched network can be described as follows. If a source party or *caller* at one node wishes to speak with a destination party at another node, this is signaled to the network control. Prior to any transmission, the control then attempts to reserve a route of trunks (channels) connecting the caller to the destination party. If the attempt is successful, the route is handed over to the caller and the call is established or carried. If the attempt is unsuccessful, the caller is turned away from the network and the call is termed *lost*.

Figure 1.1 shows signaling messages in a circuit-switched network. Before the start of the transmission, the circuit must be set up in a connect phase. After transmission terminates, a disconnect phase takes place. The source initiates the call-set up phase by sending a *send request* signal to the switch to which it is connected. This signal carries the information needed to set up the connection. In practice, an $off - hook$ signal plus dial digits carry this information. After processing at the switch a *connect* message is sent to the next switch on the path to the destination. When a connect message reaches the final switch in the network, a ringing signal is delivered to the destination. If the destination station is ready to accept the call, it replies with an answer or response message. The answer message travels back to the source indicating that transmission can begin. The time between the initiation of the $send - request$

message to the receipt of the *start to send* message is the call set up or connection time. When the transmission terminates, the source station sends a *clear – down* signaling message to the switch to which it is connected. This switch in turn sends a *disconnect* message, releasing the trunks along the complete circuit. Figures 1.2 and 1.3 show the components of call set up and call holding times for a simple two node example. The call holding time begins with the transmission of the first connect message and terminates once the destination exchange receives the final disconnect message.

There are two methods of sending the various signaling messages, inband signaling and common channel signaling (CCS). In inband signaling, signaling messages are sent over the same trunks as the calls themselves. In CCS, control messages or signals are carried over separate channels or networks using packet-switching techniques. Therefore a telephone network that adopts CCS, uses circuit switching for the calls and packet switching for the control messages.

1.2 - Hierarchical Routing

The traditional method of routing in telephone networks is hierarchical routing. Typically in networks using hierarchical routing, telephone switching offices or exchanges are classified into different levels to form a hierarchy. Fig 1.4 shows the North American hierarchy which consists of five basic classes as shown. Alternate routing is used in hierarchical networks, (Fig 1.5) as follows. When a call arrives, it

first tries the path at the bottom level of the hierarchy, which is called a direct trunk group. If all paths in the direct trunk group are busy, the call tries the paths on the next level up in the hierarchy until either it finds a free path or it is blocked after failing to find a free path in the top level of the hierarchy (the final trunk group). This type of routing refered to as fixed rule alternate routing has been used in the Bell telephone network for many years.

The main limitation to hierarchical routing particularly in a relatively large country is that traffic patterns tend to change in different offices and different parts of the network as a function of time of day or season of the year. Hierarchical routing is fixed and must be designed for peak traffic in the network. Some network capacity will therefore remain idle during non-peak intervals. Besides, changes in traffic patterns or major failure causes major local congestion. To solve these problems a range of network management controls are provided at the exchanges, for example traffic can be re-routed within the hierarchy or it can be stopped from entering the network for a particular destination, also trunk reservation levels can be set in order to reserve a certain number of trunks for direct traffic only. These are, in general manually activated by telephone company staff once they become aware of an overload situation. As a result there will be time delays while these are being manually implemented, causing some loss of traffic. Studies [1] have shown that the rigid hierarchical structure of the network results in higher network cost compared with that to one without hierarchical structure.

1.3 - Non-hierarchical Routing

In non-hierarchical networks, offices (nodes) in a network can all be connected to each other (Fig 1.6). Each node can act as a source or destination for traffic and, in addition, act as an intermediate (or tandem) node to form alternative routes for overflow traffic. Dynamic routing is implemented in non-hierarchical networks. In such routing strategies the first-choice and overflow routes may be varied to adapt the overall pattern of traffic to changes in offered traffic or link failure.

An example of dynamic routing is Dynamic Nonhierarchical Routing (DNHR)[1] [2] in use by AT&T for the US inter-city toll network. DNHR is a decentralized nonhierarchical routing strategy, in which each node in the network has a prescribed series of alternate paths to be tried. The number and sequence of paths varies with the day and time of day. In the US intercity network there are three separate time zones and the busy hours will occur at different times in different parts of the network. Thus there is the possibility for a heavily loaded part of the network to use alternative routes from a more lightly loaded part. In the DNHR strategy a day is divided into ten periods and a different pattern of alternative routing is used in each time period. Fig 1.7 illustrates the DNHR routing strategy. The originating call is seen to have four possible routes to the DNHR destination exchange. The direct route (B) will always be tried first. The other three routes (A, C, and D) will be tried in different orders depending on the time of day. For example during the morning the call may try the routes in the order of B-C-A-D, while in the afternoon it may use the order

of B-A-D-C. In 1984 AT&T introduced DNHR into its toll network providing a 15 percent cost reduction over hierarchical routing [1].

Another example of dynamic routing is Dynamically Controlled Routing (DCR) used in Bell Canada. In this centralized strategy (Fig 1.8), the routes change adaptively as traffic conditions change. A central routing processor receives information periodically from switches [3] and sends a set of tandem recommendations for all call destinations to the switches every 10 seconds to update their routing tables. The recommendations are based on the available outgoing links and the load on the call processing units of exchanges. The first-choice route is fixed with the network design and would normally be the direct route in a fully connected network. A second choice determined by the central processor is attempted when the first-choice is blocked. This second-choice route is updated frequently to reflect short-term changes in the traffic pattern. Unlike in DNHR where a sequence of overflow routes is attempted, the Canadian DCR strategy uses just a single overflow route chosen by the central processor based on the information received over the past 10 seconds. Results from a routing trial in the Telecom Canada trunk network have been given in [4].

In a further paper by Ash [5] a real-time extension to DNHR is proposed called Trunk Status Map Routing (TSMR). TSMR uses a central processor which receives data from the nodes and sends back real-time modifications to the ordering of alternate routes used by DNHR and in this way attempts to respond to short term traffic fluctuations. In both the AT&T and the Bell Canada cases, routes are chosen to be no more than two links long. Therefore there is only one tandem switch to be chosen

for a given alternate path.

Information passing has been presented in [6]. In this decentralized adaptive routing strategy adjacent exchanges that are connected by trunk groups, communicate congestion information using common channel signalling. This information propagates from the destination exchange to the originating exchange and allows the originating and intermediate exchanges to determine the least congested route to the destination. Routing is based on a congestion measurement called a Stochastic Residual Index (SRI) where a stochastic version of spare capacity is considered. The routing priority is to first select direct routes and then tandem routes with minimum congestion. However exchanges will make the routing choice without knowledge of the entire network status. To reduce the processing requirements associated with this approach, the best route would be calculated periodically. Other strategies using call repacking have been considered in [7]. A survey of dynamic routing strategies is presented in [8].

One of the main parts to this work is devoted to the Learning Automata (LA) routing strategies [9][10]. In these decentralized schemes action probabilities (choice of routes) are updated according to the completion or rejection of calls. In chapters 4 and 5, learning automata strategies are compared to other dynamic routing schemes such as Dynamic Alternative Routing (DAR), Least busy alternative (LBA), Random Routing (RR) and Fixed Routing (FR). The DAR, is a decentralized scheme which has recently been implemented by British Telecom [11].

1.4 - Classification of Dynamic Routing Methods

Various dynamic routing schemes may be classified with respect to the following three parameters.

1 - The basic mechanism used in the scheme - The dynamic routing procedures are divided into two basic mechanisms:

1a) Time dependent schemes, which operate according to non-coincident busy periods over the network. The routing procedures are fixed for each time period but have some ability to cope with abnormal traffic fluctuations.

2b) Network state dependent schemes, which detect the congestion patterns in the network and route traffic accordingly. These schemes are generally referred to as adaptive.

2 - The topological scope of the routing calculations - The dynamic routing algorithms are divided into three groups:

2a) Centralized methods make network-wide routing calculations.

2b) Distributed methods perform calculations relating to a limited section of the network. This could be either the origin neighbourhood, the destination neighbourhood or some combination of both.

2c) Isolated methods consider only the information available locally in each node.

3 - The frequency with which routing recommendations are updated.

3a) Hours.

3b) Minutes or less. The exchange receives a set of routing alternatives which are

modified on a regular basis.

3c) On a per call basis.

1.5 - Overview

In this chapter we reviewed the principles of circuit-switching in telephone networks. We then considered routing in circuit-switched networks and described the classical method of hierarchical routing of calls which is implemented in most telephone networks. Next we explained the recent nonhierarchical methods of routing that are being introduced in telephone networks. Nonhierarchical routing is being introduced as a result of studies showing that it will result in significant cost savings as compared with the more traditional method of hierarchical routing in circuit-switched networks.

Throughout this thesis, different dynamic routing schemes are studied and compared with each other in nonhierarchical circuit-switched networks. Emphasis is placed on Learning Automata (LA) schemes because of their ability to adapt to changes to the network conditions. Chapter two provides a review of the basic definitions and concept of LA. Linear algorithms including L_{R-P}, L_{R-I} and $L_{R-\epsilon P}$ are introduced. This is followed by a discussion of two mathematical models which predict the behaviour of linear algorithms in a changing environment.

Chapter three is concerned with computer modelling of circuit-switched networks and describes two simulation packages designed for carrying out experimental studies

of different routing algorithms. The first program written in Fortran 77 on an Amdahl 470 mainframe, allows the simulation of an N node fully connected circuit-switched network with arbitrary capacity and traffic matrices. The second package written in C on a Sun workstation is similar to the first one but with an added front end. The front end allows the user to draw a network on the screen and to enter the network's capacity and traffic matrices from the keyboard. Both packages provide simulation of six different routing strategies and are easily expandable to implement more schemes.

Chapter four performs analytical and simulation studies on the use of several routing procedures in nonhierarchical circuit-switched networks. The routing algorithms considered are: FR, RR, DAR and L_{R-P}. Mathematical models are derived for analysing these routing strategies. The principles lying behind these models are Erlang's model for a single link together with an independent link blocking assumption. Using the simulation model developed in chapter three, a set of simulations are performed to verify analytical models and to compare different routing policies to the L_{R-P} automata schemes under both steady state and failure conditions. The test network considered in this chapter is a five node fully connected from BT, designed for Fixed Routing (FR). It will be shown that FR gives the lowest blocking probabilities under both steady state and failure conditions and dynamic routing algorithms are as good as FR when trunk reservation mechanisms are employed.

In chapter five, LA is compared with DAR on small networks with link capacities and traffic matrices designed to force dynamic routings. It will be shown that in such networks LA always outperforms DAR.

Chapter six consists of two parts. In the first part we study the problem of instability in nonhierarchical networks. We use both analytical and simulation models to demonstrate the existence of network instabilities when Automatic Alternate Routing (AAR) is used in a symmetric uniformly loaded network. We then consider L_{R-P}, L_{R-I} and DAR strategies and show that no instabilities are found in the network. It will also be shown that using trunk reservation prevents instability and provides a high level of network carried load during overloads. In the second part of this chapter we study the performance of dynamic routing algorithms on two large networks with realistic capacity and traffic matrices from BT and Bell labs.

Finally conclusions and further work are presented in chapter seven. We now consider the fundamentals of stochastic learning which form the basic building block of the dynamic routing scheme which is the major concern of this thesis.

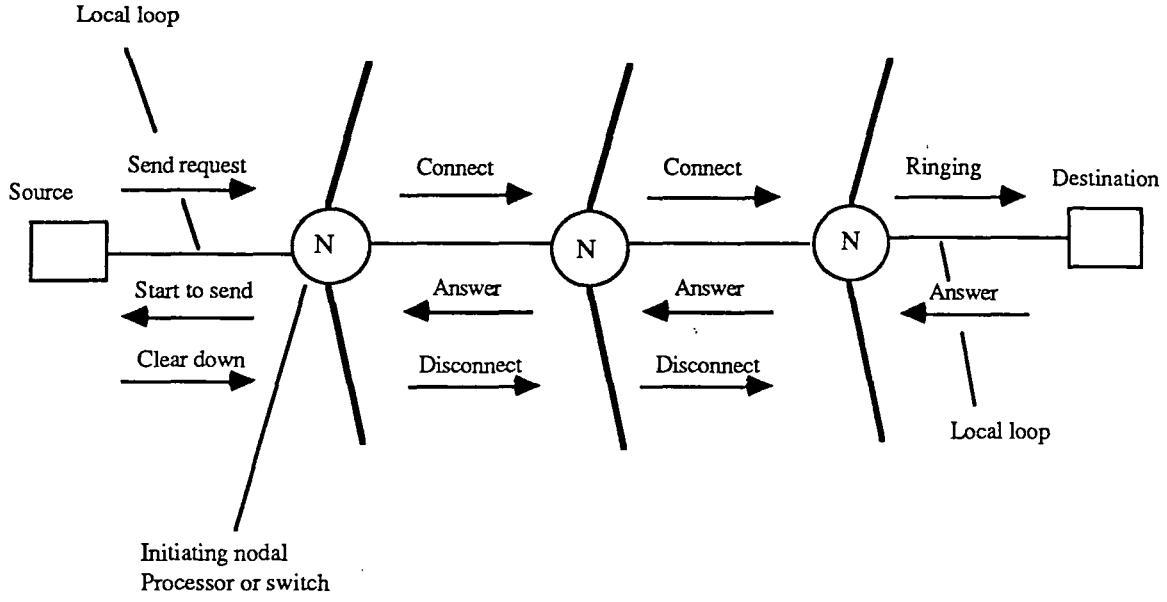


Fig 1.1 Signaling messages in a circuit-switched network.

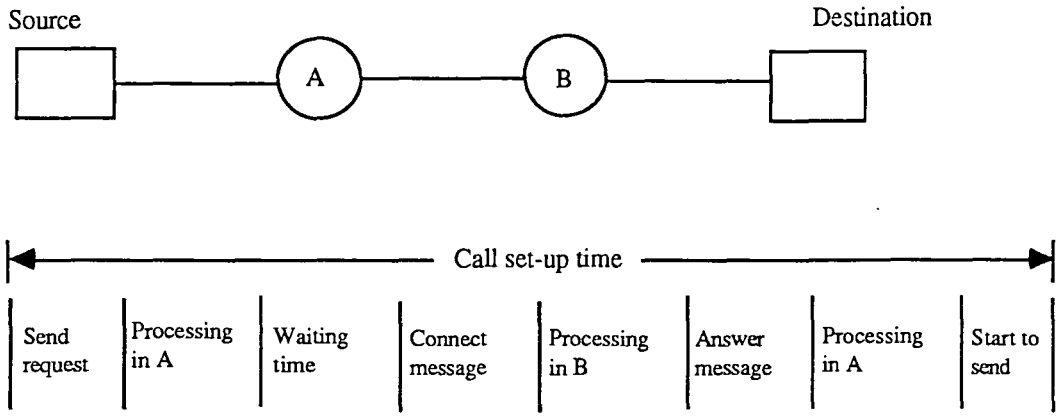


Fig 1.2 Components of call set-up time for a simple two node example.

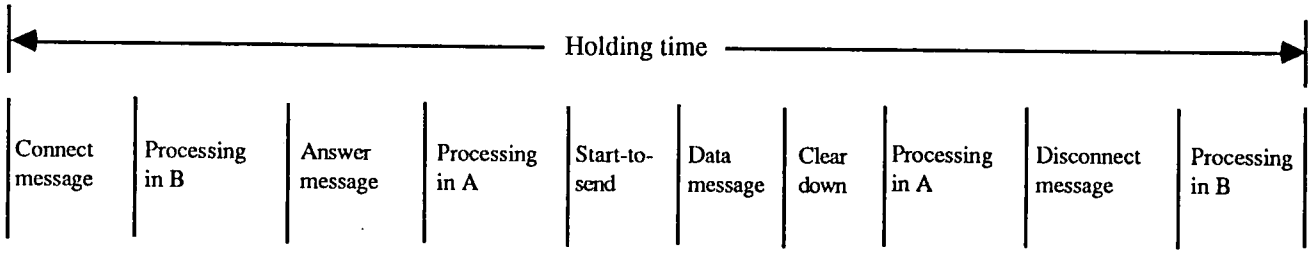


Fig 1.3 Components of holding time for the two node network in Fig 1.2.

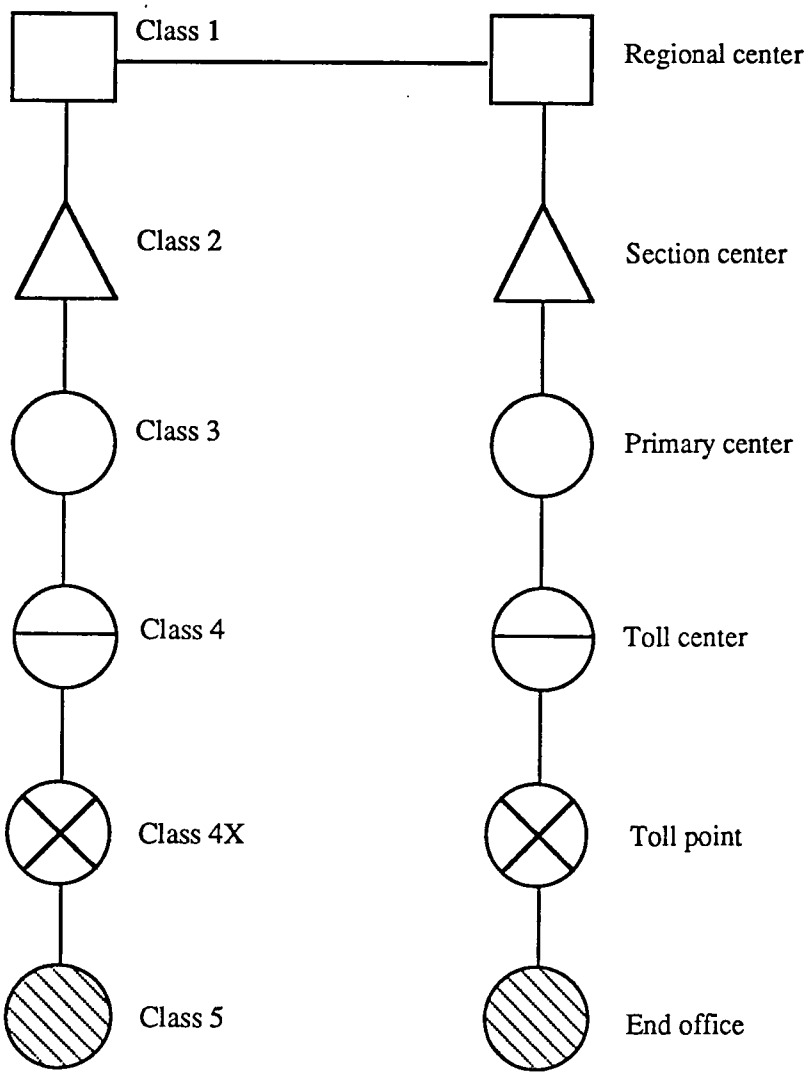


Fig 1.4 North American telephone office hierarchy.

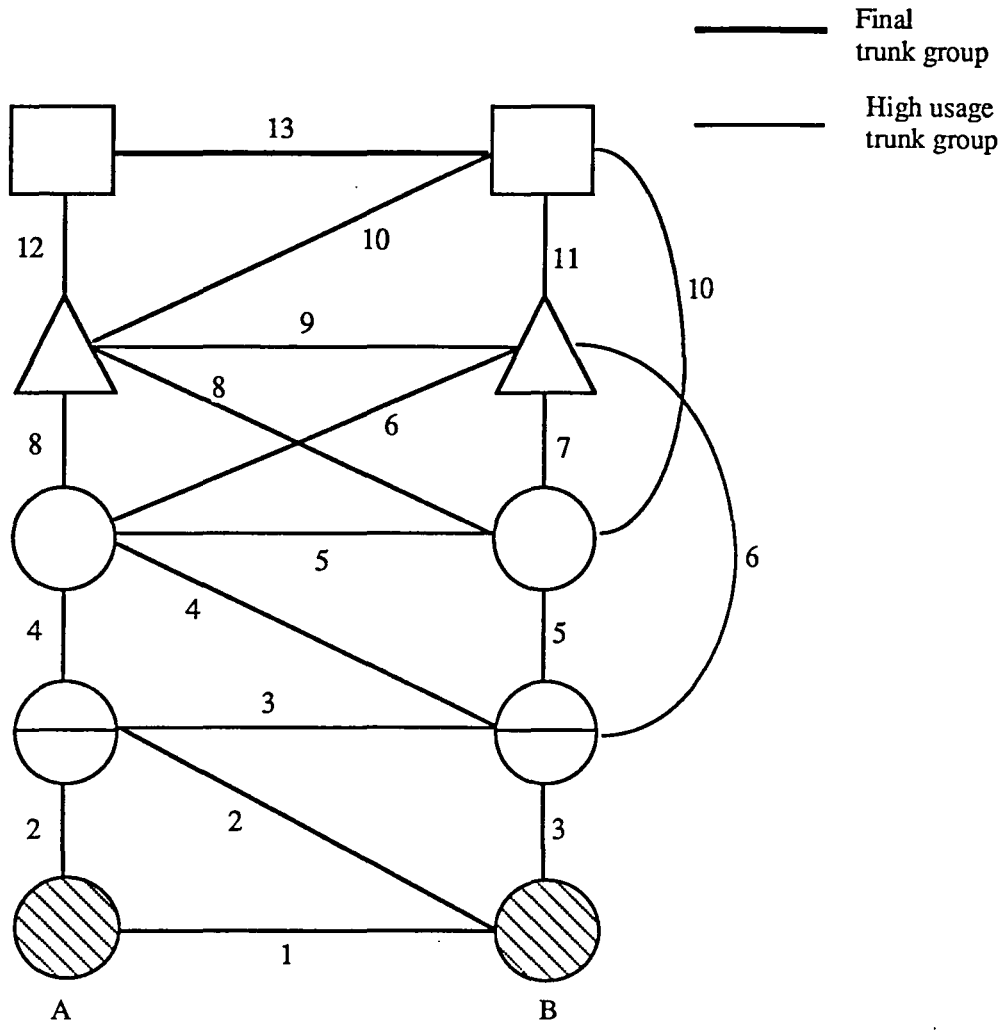


Fig 1.5 Typical hierarchical routing pattern.

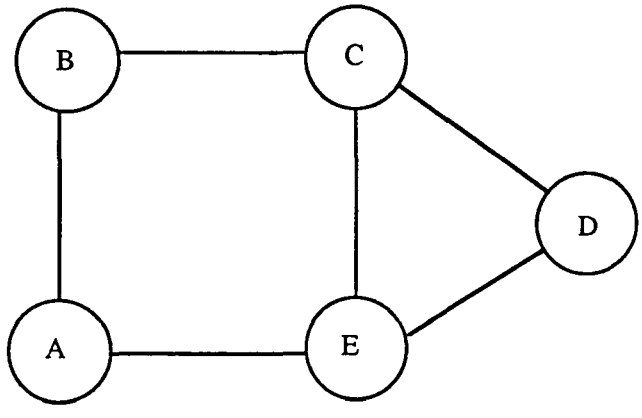


Fig 1.6 A nonhierarchical network.

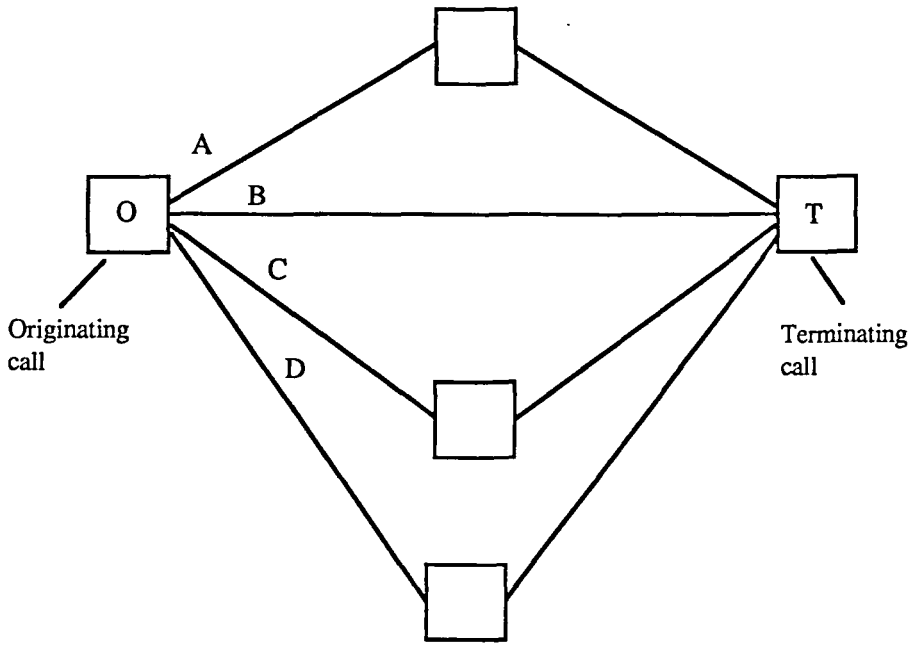


Fig 1.7 Dynamic Nonhierarchical Routing.

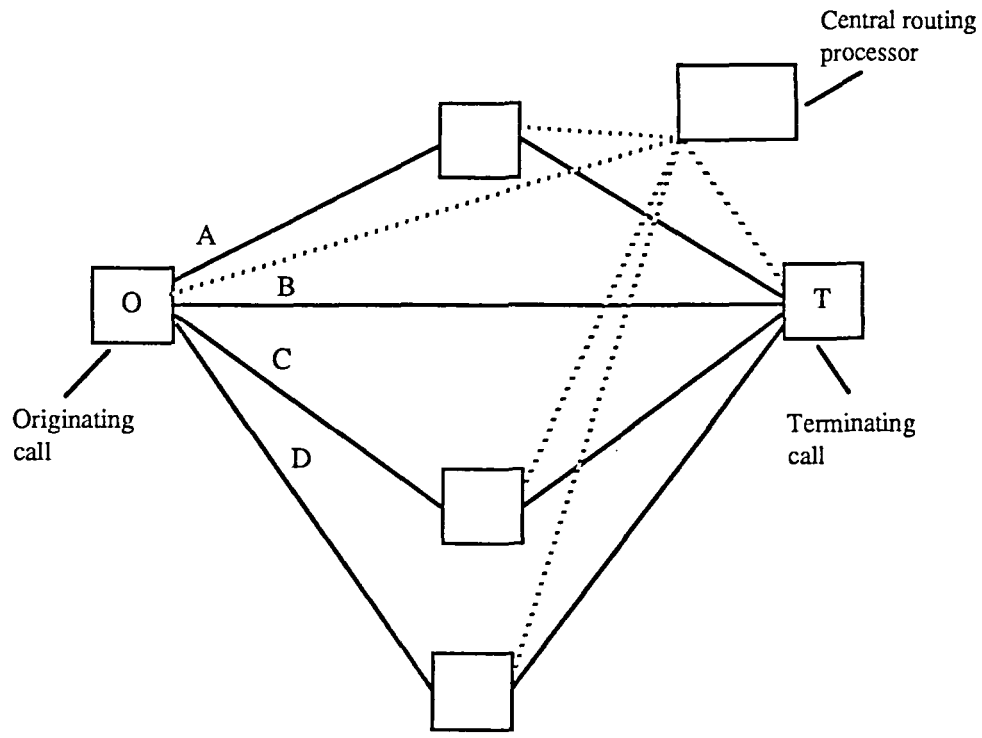


Fig 1.8 Dynamically Controlled Routing.

Chapter Two

Introduction to Learning Automata

2.1 - Introduction

The study of deterministic automata operating in a random environment was initiated by Tsetlin [12] to model the behaviour of biological systems. This was extended by Varshavskii and Vorontsova [13] to an investigation of variable structure stochastic automata. These automata are now called learning automata and their theory have been extensively developed over the last few years [14] [15] [9]. The automata choose an action from a finite set and on the basis of the response of the environment, update their strategies. In deterministic automata the state of the automaton together with the input (the output of the environment) completely determines the next action chosen. In stochastic automata the probability distribution on the action set is updated and is in turn used to generate the next action.

2.2 - The basic learning model

A learning automata consists of an automaton and an environment connected to form a feedback system, Fig 2.1. The input, at each stage n, to the environment is the action chosen by the automaton at n. The output of the environment in turn forms the input to the automaton; which influences the state transition.

2.2.1 - The Automaton

The automaton shown in Fig 2.2 is defined by a quintuple $\{\phi, \alpha, \beta, F, G\}$ where:

$\phi = \{\phi_1, \phi_2, \ldots, \phi_s\}$ is the state set $(2 \leq s < \infty)$,

$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is a finite action set $(r \leq s)$,

$\beta = \{0, 1\}$ is the input set,

$F : \phi \times \beta \rightarrow \phi$ is the state transition mapping and

$G : \phi \rightarrow \alpha$ is the output mapping.

The state ϕ represents the memory of the automaton and records the past experience. F and G can be considered as matrix operators, transforming $\phi(n)$ to $\phi(n+1)$ and $\phi(n)$ to $\alpha(n)$. Mathematically this can be expressed by the relations:

$\phi(n+1) = F[\beta(n), \phi(n)]$

$\alpha(n) = G[\phi(n)]$

If the elements of both matrices are 0 or 1, the automaton is a fixed structure deterministic automaton. If the elements of F and G lie in the interval [0,1], the automaton is a fixed structure stochastic automaton. In variable structure stochastic automata, the transition matrices corresponding to various inputs are themselves updated as

the automaton operates in the environment. In this case, the updating rule has to be specified. This type of automata will be discussed in more datail in a later section.

2.2.2 - The Environment

The random environment shown in Fig 2.3 is defined by a triple $\{\alpha, \beta, z\}$, where:

$\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_r\}$ is the input set,

$\beta = \{0, 1\}$ is the output or response set and

$z = \{z_1, z_2, \ldots, z_r\}$ is the penalty set.

The output $\beta(n) = 0$ at stage n is called a favourable response (success) and $\beta(n) = 1$ an unfavourable response (failure). z_i is the probability of failure when the input is α_i:

$$Pr[\beta(n) = 1 | \alpha(n) = \alpha_i] = z_i \tag{2.1}$$

In the basic model the environment is assumed to be stationary with z_i unknown but constant. In more complex models the environments are nonstationary and these probabilities vary with time.

The environment can further be classified into three models based on the nature of the response to automata: 1) P-model, 2) S-model, 3) Q-model. In the P-model the response to an action is of a binary nature, i.e, 1 or 0, penalty or reward. If however the response is continuous in the region (0,1), then the model is called an S-model. Similarly, the Q model is the intermediate case where the response can be one of finite set of discrete values in the range (0,1). Both P and S models are implemented

in communication network traffic routing. The P-model is used in circuit-switched networks as calls are either rejected ($\beta = 1$, penalty) or accepted ($\beta = 0$, reward) and the S-model in packet switched networks because of the continuous nature of packet delays. Actual delay values can be normalized to lie in the interval (0,1).

2.3 - Variable structure stochastic automata

Stochastic automata with variable structure (SAVS), are best described in terms of their state probability vector $p(n)$ which can also be the action probability vector. This vector relates to the probability that at instant n a certain action will be performed.

$$P(n) = \{P_1(n), P_2(n), \ldots, P_r(n)\}$$

where

$$P_i(n) = prob[\alpha(n) = \alpha_i] \tag{2.2}$$

and

$$\sum_{i=1}^{r} P_i(n) = 1 \quad \forall r \tag{2.3}$$

If an automaton chooses an action α_i which results in a success, the probability $P_i(n)$ is increased and all other components of $P(n)$ are decreased. Similarly, $P_i(n)$ is decreased if α_i results in a failure. The precise manner in which $P_i(n)$ is updated determines the learning algorithm and the asymptotic behaviour of the process. In general:

$$P(n+1) = T[P(n), \alpha(n), \beta(n)] \tag{2.4}$$

For SAVS the definition of the automata can therefore be simplified as a quadruple: $\{\alpha, \beta, P(n), T\}$ where α is the action set, β is the response set, $P(n)$ is the probability distribution over the action set at stage n, and T is the probability updating algorithm which is also referred to as reinforcement algorithm.

If $P(n+1)$ is a linear function of $P(n)$, then the scheme is termed linear, otherwise it is non-linear. In some cases, $P(n)$ is updated according to different schemes depending on the intervals in which the value of $P(n)$ lies. In such a case the combined scheme is known as hybrid.

2.3.1 - Linear algorithms

The general linear algorithm is defined as follows. If $\alpha(n) = \alpha_i$ then:

for $\beta(n) = 0$:

$$P_i(n+1) = P_i(n) + a[1 - P_i(n)]$$

$$P_j(n+1) = (1-a)P_j(n) \quad j \neq i$$

for $\beta(n) = 1$:

$$P_i(n+1) = (1-b)P_i(n)$$

$$P_j(n+1) = P_j(n) + \frac{bP_i(n)}{(r-1)} \quad j \neq i \tag{2.5}$$

Where $0 < a < 1$ and $0 \leq b < 1$ are constants called the reward and penalty parameters respectively. If $b = a$, the scheme is called a Linear Reward-Penalty (L_{R-P}) scheme; if $b \ll a$ it is a Linear Reward-ϵPenalty ($L_{R-\epsilon P}$) scheme; and if $b = 0$, it is a Linear Reward-Inaction (L_{R-I}) scheme.

2.4 - Performance measures

A useful quantity in judging the behaviour of a learning automaton is the average penalty received by the automaton. At a certain stage n, if the action α_i is selected with probability P_i, the expected penalty is:

$$M(n) = E\{\beta(n)|P(n)\}$$

$$= \sum_{i=1}^{r} P_i(n) z_i \tag{2.6}$$

If no initial knowledge is assumed and the actions are chosen with equal probabilities, the value of the average penalty probability M_0 is given by:

$$M_0 = \frac{z_1 + z_2 + \ldots + z_r}{r} \tag{2.7}$$

A learning automata is called expedient if

$$\lim_{n\to\infty} E[M(n)] < M_0 \tag{2.8}$$

When a learning automata is expedient it does better than a scheme which chooses actions in a purely random manner.

A learning automata is called optimal if:

$$\lim_{n\to\infty} E[M(n)] = \min_i z_i = z_l \tag{2.9}$$

The above equation implies that asymptotically the action corresponding to the minimum blocking probability is chosen with a probability of one.

A learning automata is called $\epsilon - optimal$ if

$$\lim_{n \to \infty} E[M(n)] < z_l + \epsilon \qquad (2.10)$$

can be obtained for any arbitrary $\epsilon > 0$ by a suitable choice of the parameters of the reinforcement scheme. ϵ-optimality implies that the performance of the automaton can be made as close as possible to the optimal as desired.

2.5 - Behaviour of Learning Automata in stationary and non-stationary environments

2.5.1 - Stationary environments

Norman [16] has shown that the Markov process associated with an L_{R-P} scheme operating in a stationary environment is ergodic. The sequence of random vectors $P(n)$ converges to a random vector P whose distribution is independent of $P(0)$. The L_{R-I} scheme on the other hand, have been shown to be absorbing barrier algorithms. This implies that the scheme chooses one of the actions almost exclusively in the limit. However by choosing the reward parameter a sufficiently small, the probability of converging to the optimum action can be made arbitrarily close to one. This probability also depends on the initial distribution $P(0)$. The $L_{R-\epsilon P}$ automaton combines the desirable features of the L_{R-P} and L_{R-I} schemes. It is ergodic like the former but can be made almost optimal by the proper choice of the parameters of the learning algorithms.

2.5.2 - Non-stationary environments

A study of the behaviour of learning schemes in telephone networks shows that since the action of automata affect the distribution of calls the network cannot be modelled as stationary environment. In a telephone network, an automaton is located at each node and the outgoing links are regarded as the automaton's actions. The automata update their action probabilities (choice of routes) according to the completion or rejection of calls. When many calls are routed through a trunk group, it becomes less attractive for subsequent calls from the point of view of call completion; on the other hand a trunk group which is not used for an interval of time, tends to get better as calls already in progress in it are released.

Methods for modelling the network as a non-stationary environment were first suggested by Narendra and Thathachar [10]. In their model when an action α_i is performed at any stage n the corresponding penalty probability z_i of the environment increases while z_j $(i \neq j)$ decreases. A mathematically more manageable model suggested by Srikantakumar and Narendra [17] assumes that the penalty probabilities z_i are functions of the action probabilities P_i. This implies that actions that are chosen more often have higher penalty probabilities. A brief discussion of the two models is presented in the next two sub-sections. For the sake of convenience the former model is called model A and the later model B.

Model A

In model A introduced by Narendra and Thathachar [10] when an action α_i is performed at any stage n the corresponding penalty probability z_i of the environment increases while z_j $(j \neq i)$ decreases. In a telephone network, this would correspond to an increase in the average penalty probability of a trunk group that is chosen to route a call and the decrease in the average penalty probability of all other trunk groups. For a two action case the environment can be described as follows:

if $\alpha(n) = \alpha_1$

$$z_1(n+1) = z_1(n) + \theta_1(n)$$

$$z_2(n+1) = z_2(n) - \phi_2(n)$$

if $\alpha(n) = \alpha_2$

$$z_1(n+1) = z_1(n) - \phi_1(n)$$

$$z_2(n+1) = z_2(n) + \theta_2(n) \tag{2.11}$$

In general $\theta_1, \phi_1, \theta_2$ and ϕ_2 can be functions of $P_1(n)$,$z_1(n)$ and $z_2(n)$ but for simplicity they are assumed to be constants.

$\theta_i(n) = \theta_i$ a positive constant if $z_i(n) + \theta_i \leq 1$ and

$\theta_i(n) = 1 - z_i(n)$ otherwise.

Similarly

$\phi_i(n) = \phi_i$ a positive constant if $z_i(n) - \phi_i \geq 0$ and

$\phi_i(n) = z_i(n)$ otherwise.

By the above definition the increase in the penalty probabilities $\theta_i(n)$ or the decrease in penalty probabilities $\phi_i(n)$ are constants except when the new penalty

probabilities lie outside the unit interval.

Some qualitative analysis of this model and simulation results are available in [10]. However as indicated in [10], the equations describing even a single automaton-environment combination are such that the mathematical tools normally used in the study of learning automata are not adequate to analyze their asymptotic behaviour. This in turn implies that the model cannot be directly extended to the case of a network with many automata operating in it. The importance of this model lies in the fact that it eventually led to model B for which such mathematical tools are applicable.

Model B

In this model the penalty probabilities are assumed to be functions of the corresponding action probabilities. In other words, each z_i is a function of P_i. It is shown that for an L_{R-P} scheme the average penalty rates for various actions are equalized in the equilibrium state [17]. The penalty rate, $f_i(P(n)) = P_i(n)z_i(n)$ is the probability that the automata receives a penalty at stage n from action α_i.

The following assumptions are made about $z_i(P)$:

a) $z_i(P)$ are continuously differentiable functions of P,

b) $\frac{\partial z_i(P)}{\partial P_i} > 0 \quad \forall i,$

c) $\frac{\partial z_i(P)}{\partial P_i} \gg \frac{\partial z_i(P)}{\partial P_j} \quad \forall i, i \neq j.$

Assumption (b) implies that the penalty probability using a specific action in-

creases monotonically with the probability P_i with which that action is chosen. Assumption (c) implies that z_i is much less influenced by the probabilities with which the other actions α_j $(j \neq i)$ are chosen.

It is shown [17] that for an L_{R-P} scheme there exists an equilibrium point $P^* = [P_1^*, P_2^*, \ldots, P_r^*]$ such that

$$f_1(P^*) = f_2(P^*) = \ldots = f_r(P^*) \tag{2.21}$$

and P^* is unique. Further the Markov process $\{P(n)\}_{n\geq 0}$ is ergodic and converges as $n \rightarrow \infty$ to a stationary probability distribution $\bar{P}$ independent of the initial distribution $P(0)$. By adjusting the learning parameter a to be small the variance of $\bar{P}$ can be made arbitrarily small and mean value close to P^*. Similar analysis for the $L_{R-\epsilon P}$ scheme[10][17], shows there exists a vector P^* such that

$$z_1(P^*) = z_2(P^*) = \ldots = z_r(P^*) \tag{2.22}$$

Therefore an L_{R-P} scheme attempts to equalize the penalty rates while an $L_{R-\epsilon P}$ automaton tends to equalize penalty probabilities at various nodes. Simulation studies have confirmed the theoretical results[9]. Furthermore it is also found that in simple networks both L_{R-P} and $L_{R-\epsilon P}$ schemes result in performance close to the optimum.

2.6 - Summary

Following the introduction a review of the basic concepts and definitions of learning automata and random environments were given. Variable structure stochastic

automata were discussed together with different linear algorithms including L_{R-P}, L_{R-I} and $L_{R-\epsilon P}$ schemes. The possible ways in which the behaviour of learning automata can be judged were defined. Section 2.5.2 considered two mathematical models to predict the behaviour of different linear algorithms in a changing environment. It was shown that an $L_{R-\epsilon P}$ scheme attempts to equalize penalty probabilities while an L_{R-P} scheme tends to equalize penalty rates. In the context of a telephone network this implies that an $L_{R-\epsilon P}$ automaton equalizes the average blocking probabilities and an L_{R-P} automaton equalizes the average blocking rates at each node. It will be shown in a later chapter, how we can implement the load equalization property of an L_{R-P} scheme to predict the theoretical overall blocking probability of a fully connected circuit-switched network.

In the next chapter we will design simulation packages to enable us to perform experimental studies on the use of the LA and several other dynamic routing strategies in circuit-switched networks.

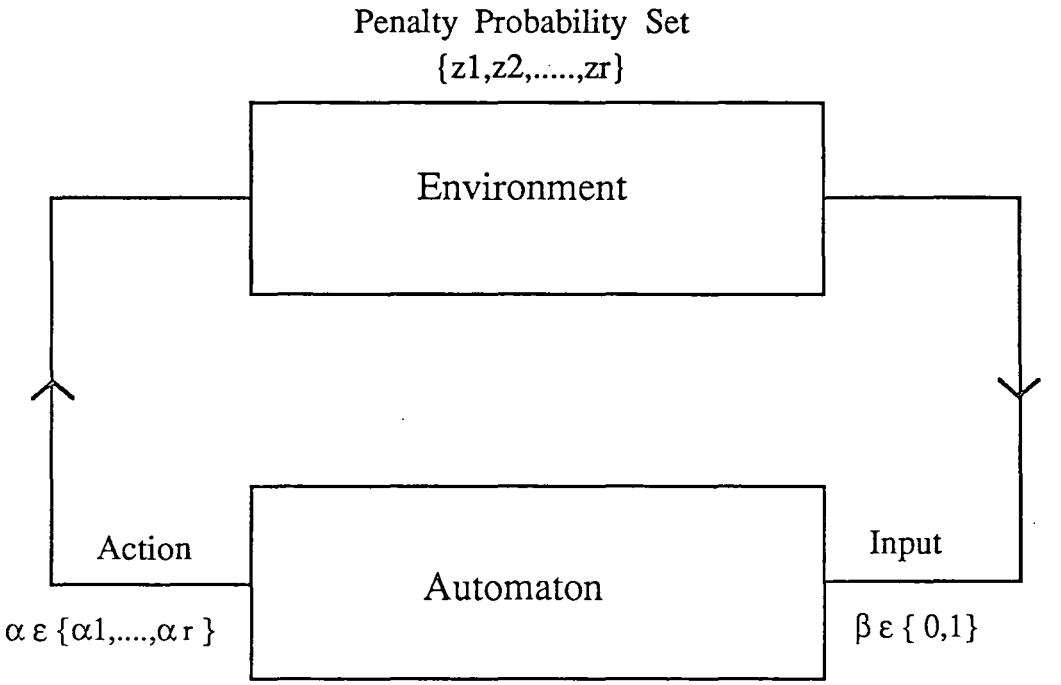


Fig 2.1 Automaton-environment feedback configuration.

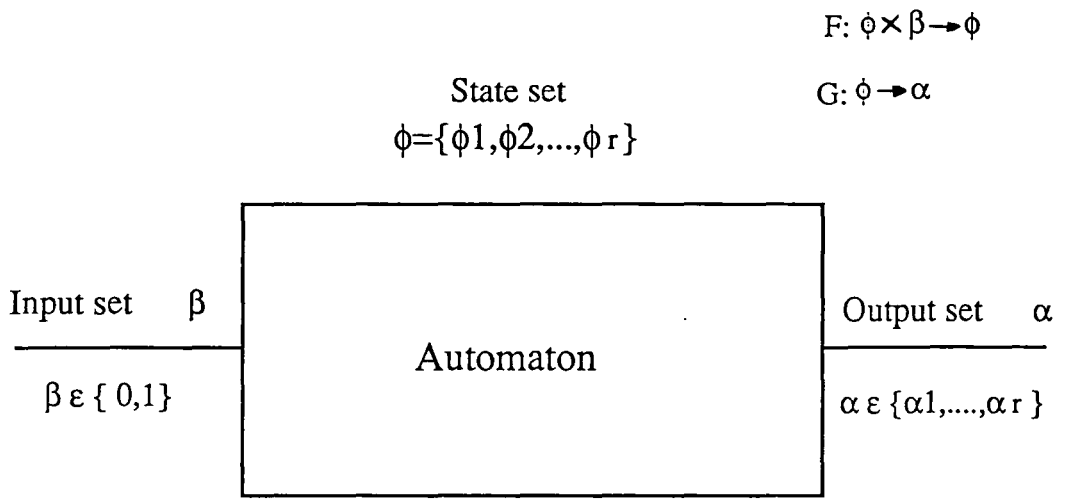


Fig 2.2 The automaton.

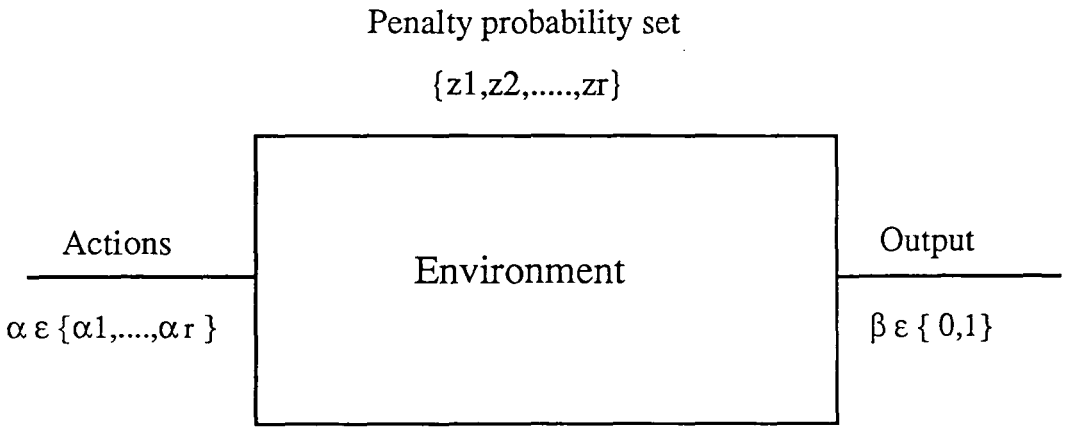


Fig 2.3 The environment.

Chapter Three

Design of a simulator for circuit-switched telephone networks

3.1 - Introduction

A simulation is the imitation of the operation of a real-world process or system over time. The behaviour of a system is studied by developing a simulation model. This model usually takes the form of a set of mathematical assumptions concerning the operation of the system. After the model is developed, it can be used to investigate many questions about the real-world system. Changes to the system can first be simulated in order to predict their impact on system performance. Simulation can also be used to study systems in the design stage, before such systems are built. Thus, simulation modelling can be used both as an analysis tool for predicting the effect of changes to existing systems, and as a design tool to predict the performance of new systems under varying sets of circumstances.

This chapter is concerned with computer modelling of circuit-switched networks

and describes two simulation packages designed for carrying out experimental studies of different routing algorithms. The first program written in Fortran 77 on an Amdahl 470 mainframe, allows the simulation of an N node fully connected circuit-switched network with arbitrary capacity and traffic matrices. The second package written in C on a Sun workstation is similar to the first one but with an added front end. The front end allows the user to draw a network on the screen and to enter the network's capacity and traffic matrices from the keyboard. Both packages provide simulation of six different routing strategies which are explained in detail in subsequent chapters.

A circuit-switched telephone network can be modelled as a set of terminals which both generate and receive calls, a connecting network which provides the physical paths or trunks over which communication takes place, and a control system that provides supervisory signals. The arriving calls generated at the terminals are modelled by a Poisson process with point-to-point traffic loads (i.e calls per unit of time originating at node i destined to node j). The number of trunks, c_{ij} in trunk group T_{ij}, as well as the calling rates λ_{ij} from node i to node j, can be represented in the form of matrices.

3.2 - Poisson process

The Poisson process is the arrival process used most frequently to model the behaviour of telephone traffic. Three basic statements are used to define the Poisson arrival process:

1 - The probability of an arrival in the interval Δt is defined to be $\lambda \Delta t$, with λ a specified proportionality constant.

2 - The probability of zero arrivals in Δt is $1 - \lambda \Delta t$.

3 - An arrival (event) in one time interval of length Δt is independent of events in previous or future intervals.

The probability P(n) of n arrivals in t is given as:

$$P(n) = (\lambda t)^n e^{-\lambda t}/n! \qquad n = 0, 1, 2, ...$$

This is called the Poisson distribution with parameter λt. The parameter λ is the average rate of Poisson arrivals.

For this study, the behaviour of calls arriving at a source node is assumed to be Poisson process with an average point to point arrival rate given by λ_{ij} calls/unit time.

The usual way of describing an arrival pattern is in terms of the inter-arrival time, defined as the interval between successive calls. For an arrival pattern that has no variability, the inter-arrival time is a constant. When the arrivals vary stochastically, it is necessary to define the probability density function of the inter-arrival times. For Poisson arrival process, the inter-arrival times are exponentially distributed random variables [18]. Let τ represent the time between successive arrivals, then the probability density function $f(\tau)$ is given by:

$$f(\tau) = \lambda e^{-\lambda \tau}$$

with mean=$1/\lambda$.

The most commonly used distribution for call duration is the exponential distribution where the probability of a call lasting t seconds is given by:

$$h(t) = \mu e^{-\mu t}$$

Where $\frac{1}{\mu}$ is the mean call holding time.

It is shown in [19] that since holding time is exponential, the call departures represent a Poisson process.

3.3 - Random numbers

Generating random numbers uniformly distributed in a specified interval is fundamental to simulation. Random numbers from almost any distribution can be obtained by transforming, (0,1) uniform random numbers. The linear congruential method is the most commonly used technique to generate random numbers [20]. A linear congruential generator, produces a sequence of integers $X_1, X_2, ...$ between zero and $m-1$ according to the following recursive relationship:

$X_{i+1} = (aX_i + c) \bmod m$ i=0,1,2,...

The initial value X_0 is called the seed, a is called the constant multiplier, c is the increment and m is the modulus. The selection of the values for a, c, m and X_0 affects the statistical properties and the cycle length [20]. Random numbers between zero and 1 can be generated using the relationship:

$$U_i = \frac{X_i}{m} \qquad i = 1, 2, ...$$

3.4 - Discrete event simulation

A discrete event simulation is the modelling of a system whose state variable(s) changes only at discrete points in time. A circuit-switched telephone network is an example of a discrete system since the state variable, the number of calls in progress changes only when a call arrives or a call is completed. A continuous simulation is the modeling of a system whose state variable(s) changes continuously over time. An example is the level of water behind a dam.

The simulation of a discrete system may be regarded as the sequencing and handling a series of events in time. To do this, two approaches may be considered:

1 - Asynchronous

2 - Synchronous

In asynchronous simulation, events such as arrivals can occur at any time. The events are simulated whenever they occur and the system clock is moved to the point of the nearest event. A synchronous simulator steps through time in constant intervals $0, \delta t, 2\delta t, \ldots$, checking if any event has occurred. However this approach has a number of drawbacks. First such models are hard to program because nonsimultaneous events may be treated as simultaneous if they fall in the same interval. This usually leads to complicated problems of priority and sequencing. The second drawback is to obtain a sufficiently accurate estimate, it may be necessary to take δt very small leading to larger simulation time. An asynchronous simulator does not have this failing. For the circuit-switched network model in this study the asynchronous simulation was

selected because it is simpler to implement.

3.5 - Simulation package I

The first simulation package written in Fortran 77 on an Amdahl 470 mainframe simulates a network of up to 10 nodes. The software can easily be extended to any number of nodes by increasing the dimension of arrays at the beginning of the program, with the rest of the program remaining unchanged. The package consists of three parts:

1 - An input file containing the initial data for simulation, for example number of nodes, capacity matrix etc.

2 - An actual simulation program for carrying out experimental runs, reading data from the input file and writing simulation results into an output file.

3 - An output file containing the data produced during the simulation run.

The simulation program consists of a number of steps which are executed cyclically. First the inter-arrival time and the origin and the destination of the next call arrival are generated. Then the calls that have been completed during the inter-arrival time are cleared down. Next a free path is searched to route the call and finally the acceptance or rejection of the call together with its origin-destination information are recorded into appropriate arrays. The simulator is run for a number of time units, the results are written into the output file every one unit of time.

3.5.1 - Call arrivals

Subroutine ARRIVE simulates the Poisson arrival process by generating exponential inter-arrival times. In an N node fully connected network there are $w = N(N-1)/2$ numbers of O-D pairs. If λ_{ij} is Poisson then $\sigma = \sum_1^w \lambda_{ij}$ is also Poisson. Let τ represent the inter-arrival time. τ is an exponentially distributed random number [20] with mean $\frac{1}{\sigma}$ and is given by:

$$\tau = \frac{-1}{\sigma} \ln U$$

Where U is a uniformly distributed random number in (0,1).

The above equation calculates the time between successive calls irrespective of identity. The identity of a call is obtained by a look-up table method. For example consider a five node network. Let IO(L) be an array representing origin nodes and ID(L) representing destination nodes. Then:

L	1	2	3	4	5	6	7	8	9	10
IO	1	1	1	1	2	2	2	3	3	4
ID	2	3	4	5	3	4	5	4	5	5

Let P(L) be the probability that the call be for O-D pair (i,j)(where i=IO(L) and j=ID(L)).

$$\mathrm{P(L)} = \frac{\lambda_{ij}}{\sigma}$$

The interval (0,1) is divided into ten ranges which correspond to the ten probabilities, P(L)(L=1,2,...10) as shown in Fig 3.1 for arbitrary values of traffic arrivals. A random number U is generated and compared to P(L). When U is in range P(L), the call is

identified for O-D pair (IO(L),ID(L)). The flow chart for subroutine ARRIVE is given in Fig 3.2.

3.5.2 - Call departures

As described previously, the call departure process is assumed to be Poisson. In what follows we explain how to use this assumption to generate a random number representing the number of calls completed during the inter-arrival time, τ. Consider a link (i,j) with capacity c_{ij} and n_{ij} lines in use. The departure rate for n_{ij} lines is $\mu_n = n_{ij}\mu$. Assuming the call holding time $\mu = 1$ call/unit time, then $\mu_n = n_{ij}$. In the simulation program the number of calls cleared down is calculated for every inter-arrival time, τ. The number of departures X, occurring in the interval τ is Poisson with a parameter $n_{ij}\tau$. To generate this random number, we use the generation procedure given in [20] :

Set $X = 0$ $P = 1$

Generate a random number U_{X+1}

$P = P \times U_{X+1}$

if $P \geq e^{-n_{ij}\tau}$ then X=X+1

Repeat until $P < e^{-n_{ij}\tau}$

In the simulation program, subroutine DEPART, simulates the call departure process. The flow chart for subroutine DEPART is given in Fig 3.3.

3.5.3 - Routing

Subroutine ROUTE performs the routing of a call. A call arriving at the network for the source destination pair (i,j) is offered to the direct path first. Let FC(i,j) be the number of free circuits on link (i,j):

FC(i,j) = c(i,j) - TL(i,j)

Where TL(i,j) is the total number of calls carried on link (i,j).

If $FC(i,j) > 0$, the direct path is available and the call is routed via that path by modifying the loading array:

LD(i,j) = LD(i,j) + 1

and accepting the call:

NAC(i,j) = NAC(i,j) + 1

If the direct path is busy ($FC(i,j) \leq 0$), a tandem node k is selected using a routing algorithm and the call is offered to the two link path (i,k,j). The call is routed via (i,k,j), if at least one free circuit exists on both links ($FC(i,k) > 0$ and $FC(k,j) > 0$). The loading array is modified and the call is accepted.

LD(i,k,j) = LD(i,k,j) + 1

NAC(i,k,j) = NAC(i,k,j) + 1

If (i,k,j) is busy, the call is lost. The number of blocked calls is saved into the array NL:

NL(i,j) = NL(i,j) + 1

The flow chart for subroutine ROUTE is given in Fig 3.4. This subroutine is programed to implement six different routing algorithms. A detailed description of the routing algorithms together with mathematical models is given in later chapters.

As an example consider the L_{R-P} scheme. Let P(i,k,j) be the probability of selecting path (i,k,j). If a call is successfully routed via (i,k,j), P(i,k,j) is rewarded or increased while P(i,l,j) (l≠k) is decreased. On the other hand if (i,k,j) is selected and the call is rejected, P(i,k,j) is penalized or decreased while P(i,l,j) (l≠k) is increased.

Function LRP performs alternate routing based on the L_{R-P} scheme. The flow chart is given in Fig 3.5. This function selects a tandem node k by calling another function named LANODE. Then it calculates the number of free circuits on path (i,k,j) which is the minimum of free circuits on paths (i,k) and (k,j). A call can be routed via an alternate path if the number of free circuits on that path is more than a threshold value TRP. TRP, a trunk reservation parameter will be explained in detail in a later chapter. If the number of free circuits on path (i,k,j) is more than TRP, the probabilities are adjusted by rewarding the path (i,k,j), the function terminates and returns the value of k. If on the other hand this number is less than TRP, the probabilities are adjusted by punishing the path (i,k,j), the function terminates and returns the value of -1 indicating that the call is blocked.

Fig 3.6 shows the flow chart for function LANODE. This function selects a tandem node k using a look-up table method similar to the one explained in section 3.5.1. For example assume that the O-D pair (i,j) has three possible tandem nodes k_1, k_2 and k_3. The interval (0,1) is divided into three ranges corresponding to P(i,k_1,j), P(i,k_2,j) and P(i,k_3,j). A random number U in the range (0,1) is generated and compared to the probabilities. When U is in range P(i,k,j), the tandem node is identified as k.

3.6 - Simulation package II

The second simulation package was written in the industrial standard language C to run on a Sun workstation. This package includes a simulation program similar to the first package and a graphics front end written to use the interesting graphics facilities provided with the Sun workstation. The front end allows the user to graphically specify a network and to enter the necessary parameters for simulation. Facilities also exist to edit the network, i.e. reposition, delete and add components. Fig 3.7 shows two windows on the Sun workstation display. The window on the left is a text window and the one on the right is the graphics front end. The menu shown on the graphics front end window is selected using the left mouse button. Fig 3.8 shows another menu selected using the middle mouse button. The right mouse button is reserved for drawing purposes.

The menu shown in Fig 3.7 provides the following options: *node, link, number node, delete node, move node, delete link, capacity , traffic, simulate.* A node can be created by selecting option *node* first and then pressing the right mouse button at the desired location. Nodes can be given numbers using option *number node* (Fig 3.9). The program saves node information into a structure array:

```
struct {
    int num;
    int x;
    int y;
```

int con [cmax]

} node[nmax]

Where num is the node number given by the user, x and y are node coordinates on the graphics window, con[cmax] is an array which saves the sequence number of other nodes connected to node[i] where i (i=1,2,...,nmax) is the node sequence number.

Nodes can be connected together using *link* option. The information about each link is saved into another structure array shown below. Structure arrays are useful features of the C language because all the information associated to a node or a link can be collected in one unit, while in Fortran separate arrays must be considered.

struct {

int origin;

int destination;

int tandem[tmax]

int capacity;

int traffic;

double probability;

int scheme;

} link[lmax]

A node can be deleted by selecting option *delete node* first and then pressing the node using the right mouse button. When a node is deleted, all the links connected to that node are removed by the software. Option *delete link* deletes a link and requires pressing two nodes at the two ends of the link. For option *move node*, the node

to be moved must be selected first and then the new position of that node pressed using the mouse. Options *capacity* and *traffic* display a text menu allowing the link capacities and traffic arrival rates to be entered from the keyboard. *simulate* initiates the simulation program and displays simulation results on the screen.

The menu shown in Fig 3.8 is selected using the middle mouse button and includes the following options:

print node connections, prints number of all nodes created and the nodes that they are connected to.

print link data, prints a list of all links and their informations eg. origin, destination etc.

simulation info, displays a menu on the text window allowing the user to alter simulation parameters.

Options *LRP, LRI, DAR, RR, AAR* and *FR* allow the user to choose a routing algorithm.

Fig 3.9 shows a network with its capacity and traffic information entered. This network can be simulated by selecting the option *simulate*.

3.7 - Summary

Two simulation packages were described. The first package written in Fortran 77, simulates a fully connected network of up to 10 nodes. The software can easily be extended to any number of nodes. The simulation program includes three subroutines

for 1) generating calls, 2) clearing down calls and 3) routing. The second simulation package written in C, includes a simulation program and a graphics front end. The front end permits the user to draw and edit a network using a menu system. After entering capacity and traffic matrices, the network can be simulated by selecting option *simulate*. The program listings for simulation packages are given in appendices 4 and 5.

The simulation packages are used in the next chapter where a series of experiments are performed to study and compare different dynamic routing algorithms in circuit-switched networks.

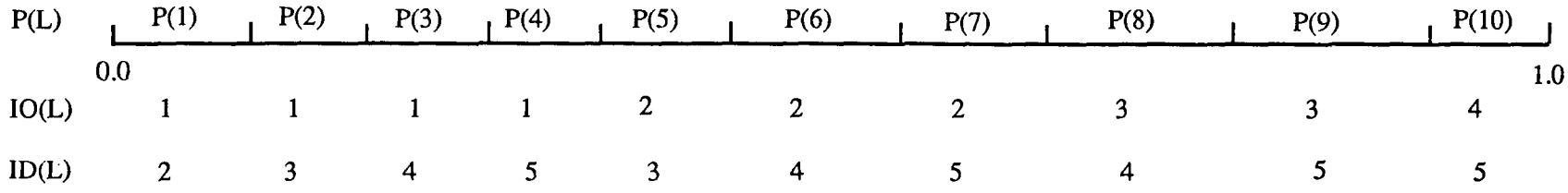


Fig 3.1 The 10 ranges in the interval (0,1), corresponding to 10 probabilities of call arrivals in a five node network.

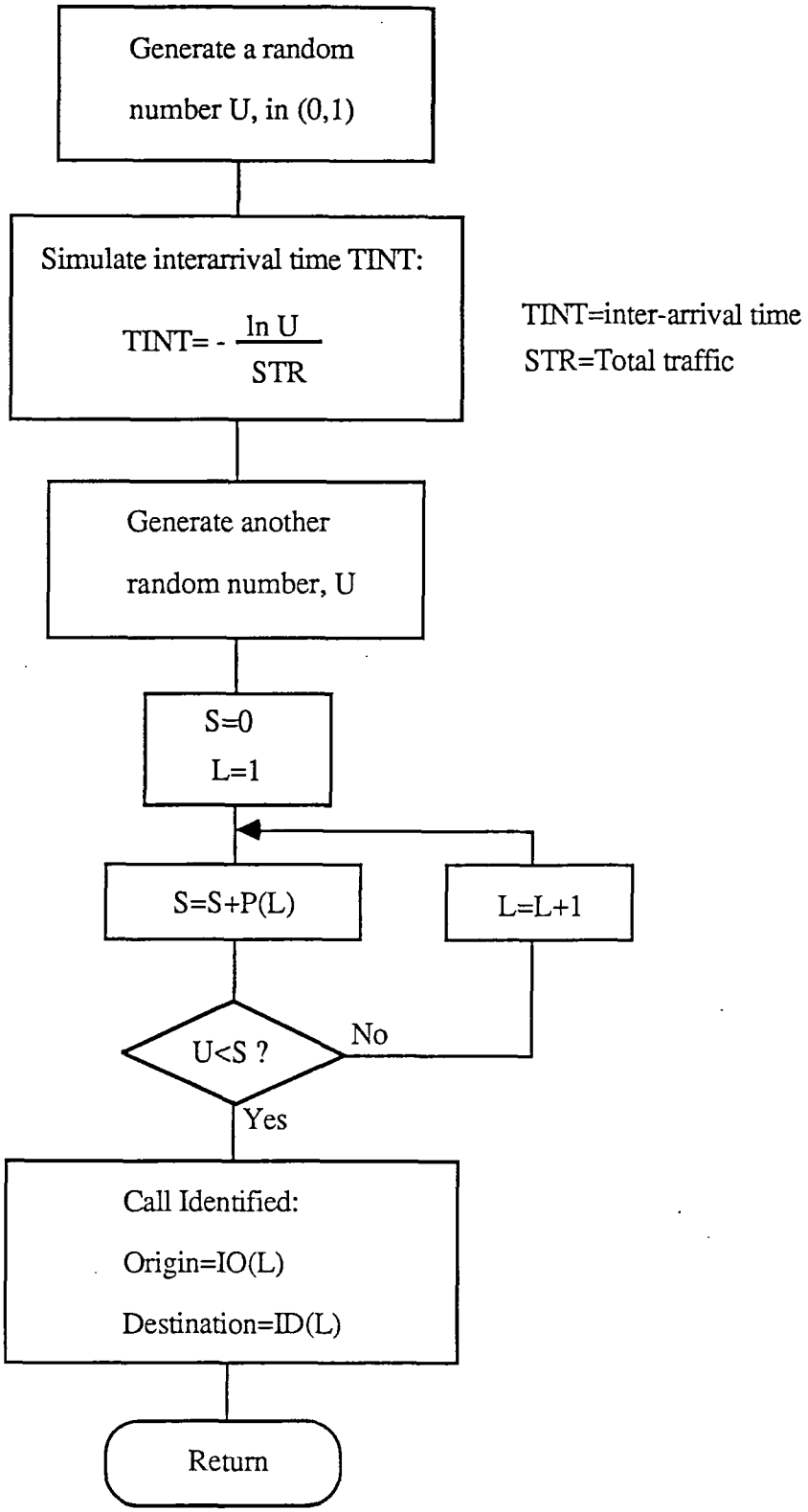


Fig 3.2 Flow chart for subroutine ARRIVE.

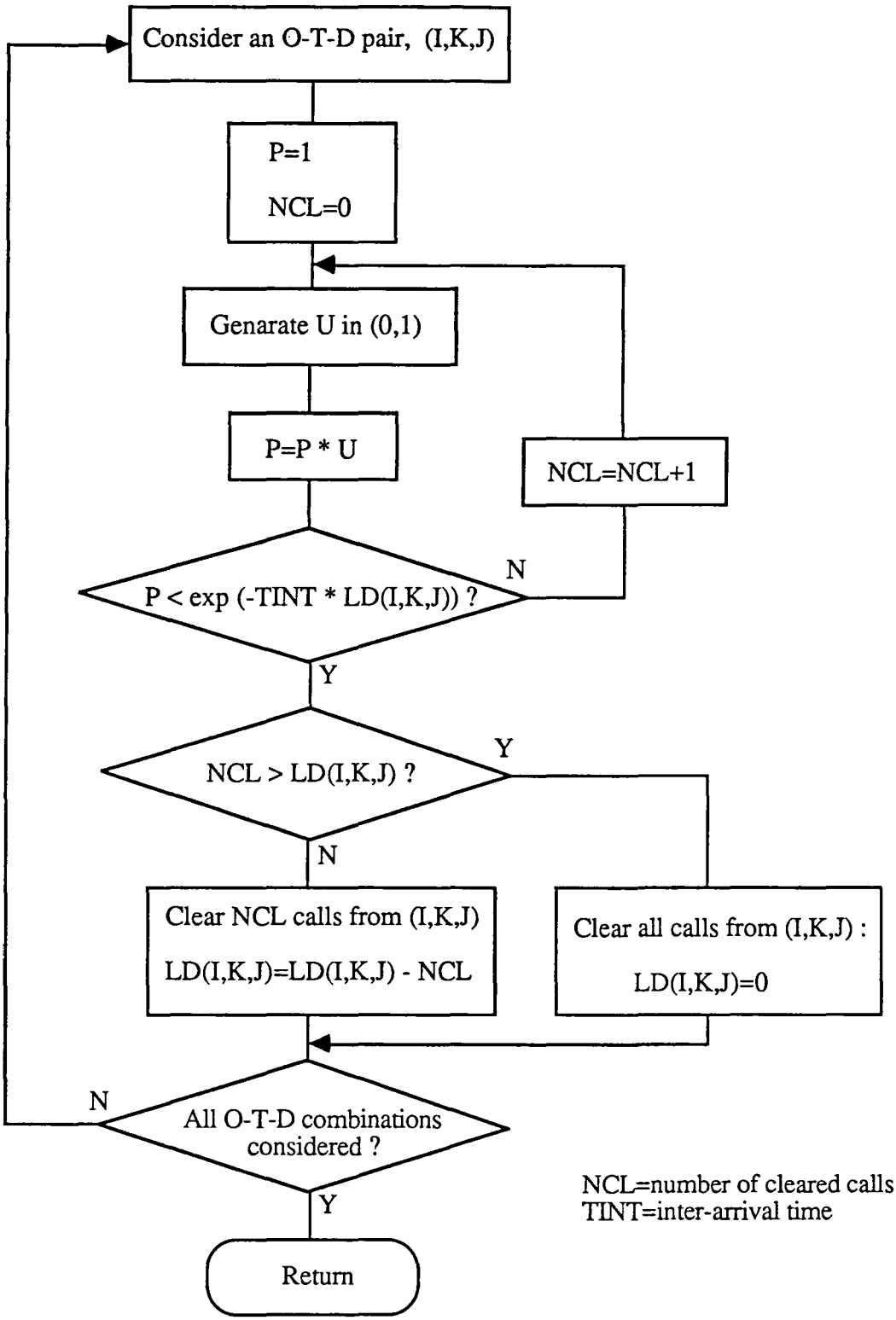


Fig 3.3 Flow chart for subroutine DEPART.

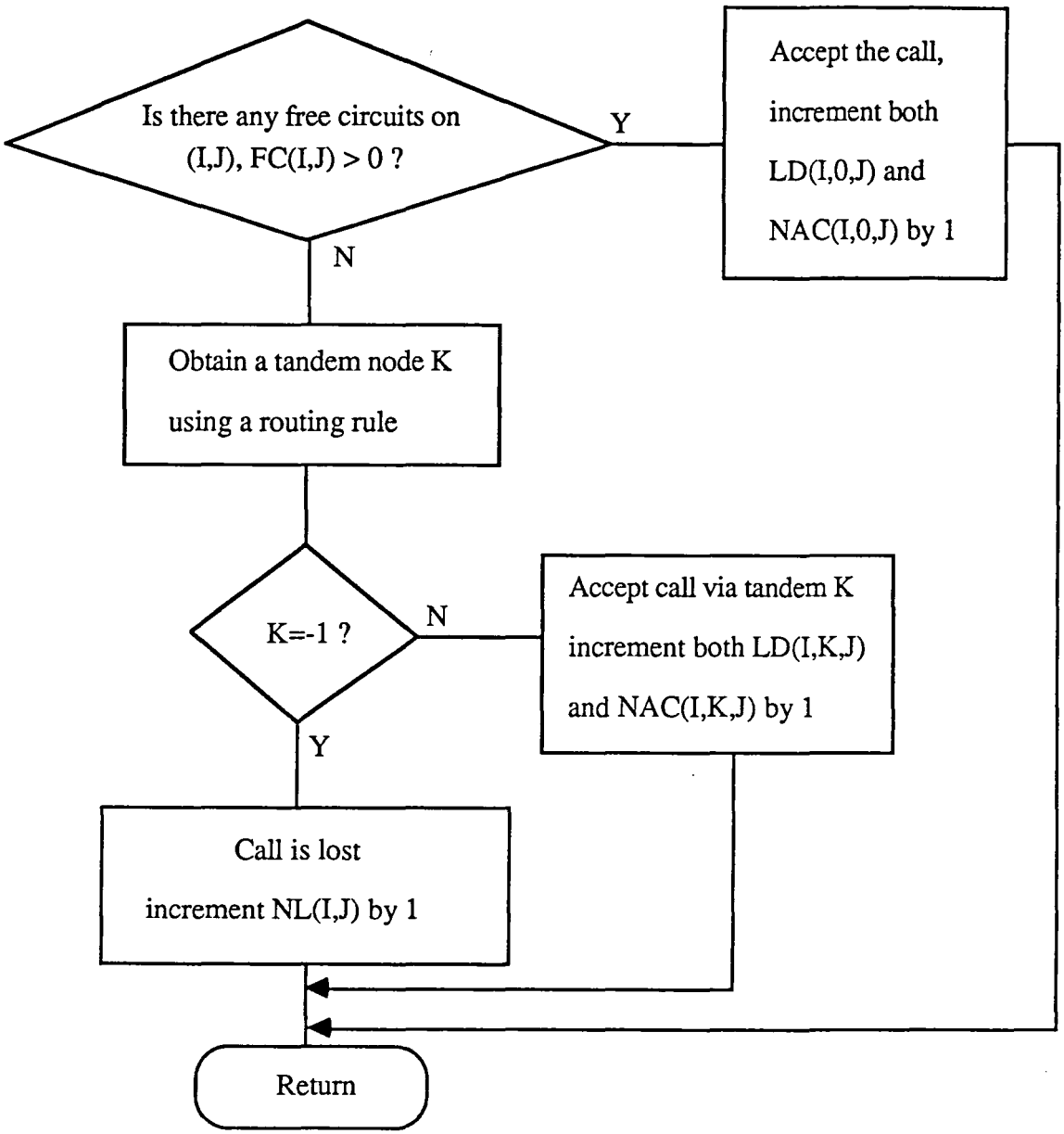


Fig 3.4 Flow chart for subroutine ROUTE.

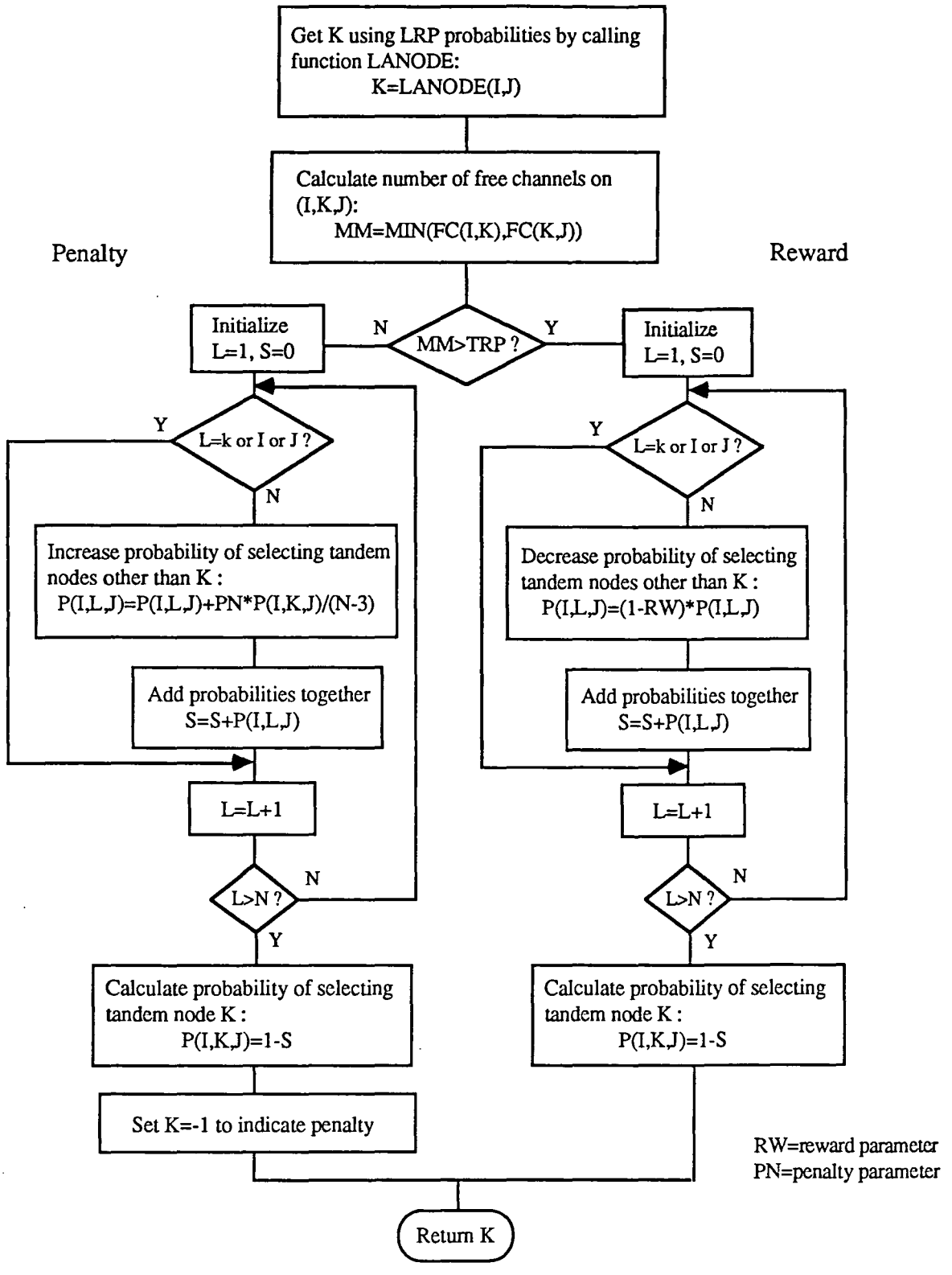


Fig 3.5 Flow chart for function LRP.

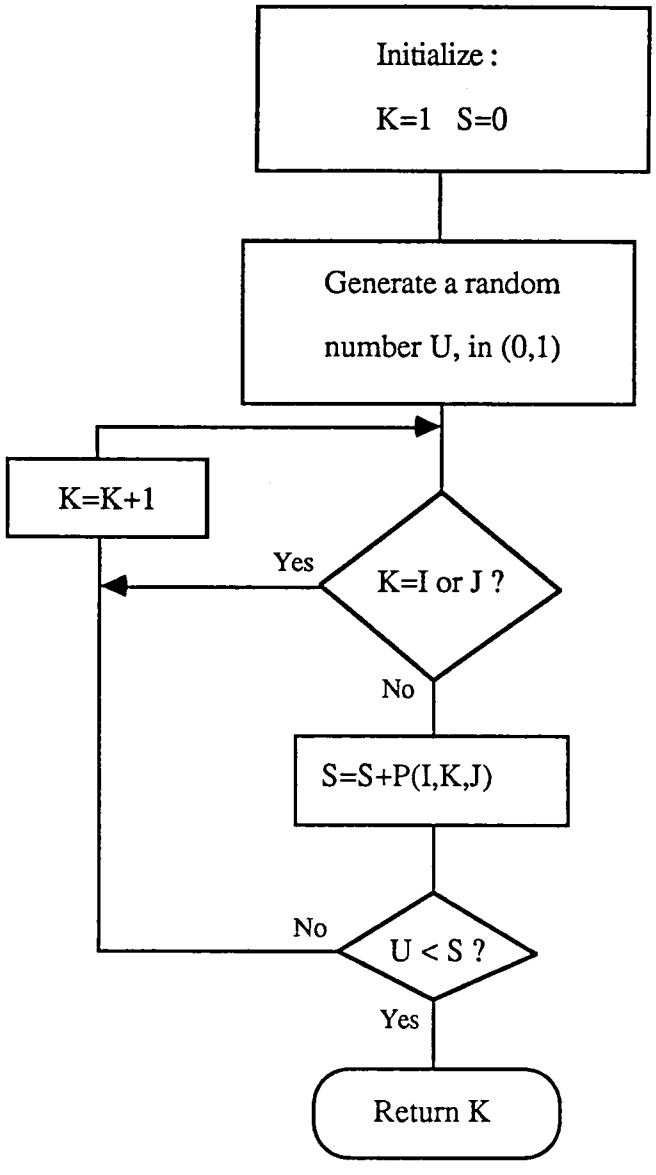


Fig 3.6 Flow chart for function LANODE.

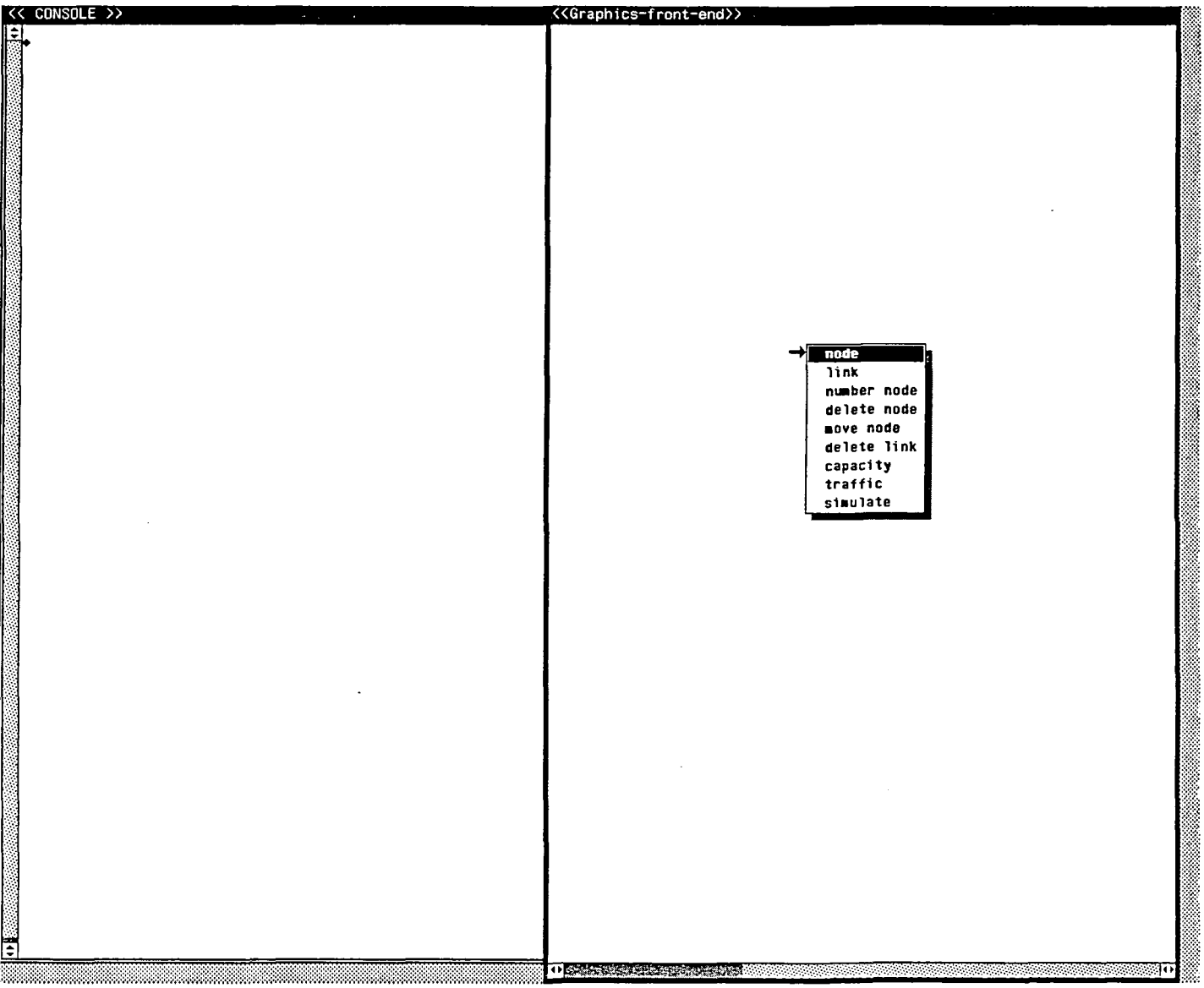


Fig 3.7 The graphics front end for the simulation pakage II, showing the first menu.

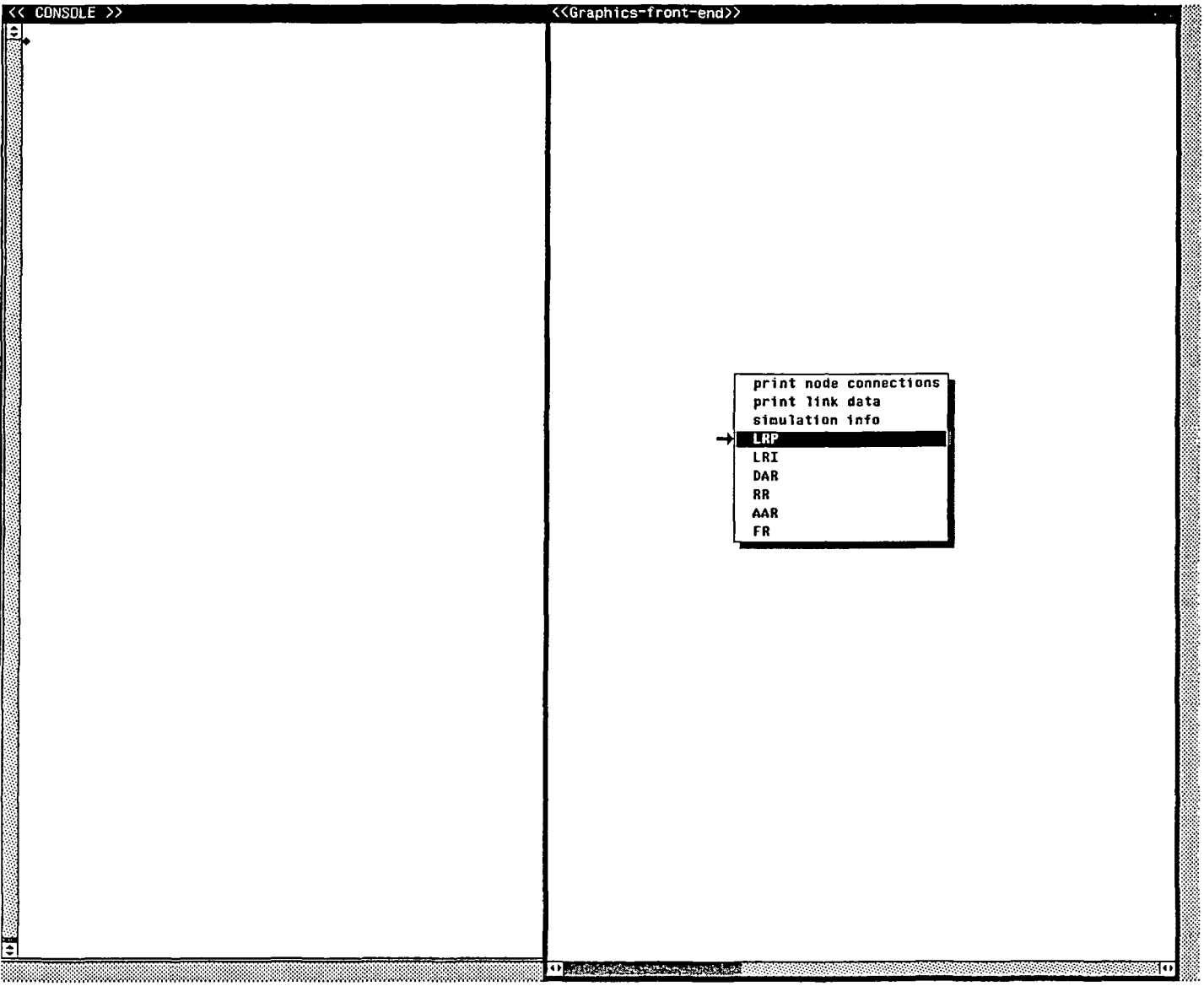


Fig 3.8 The graphics front end for the simulation pakage II, showing the second menu.

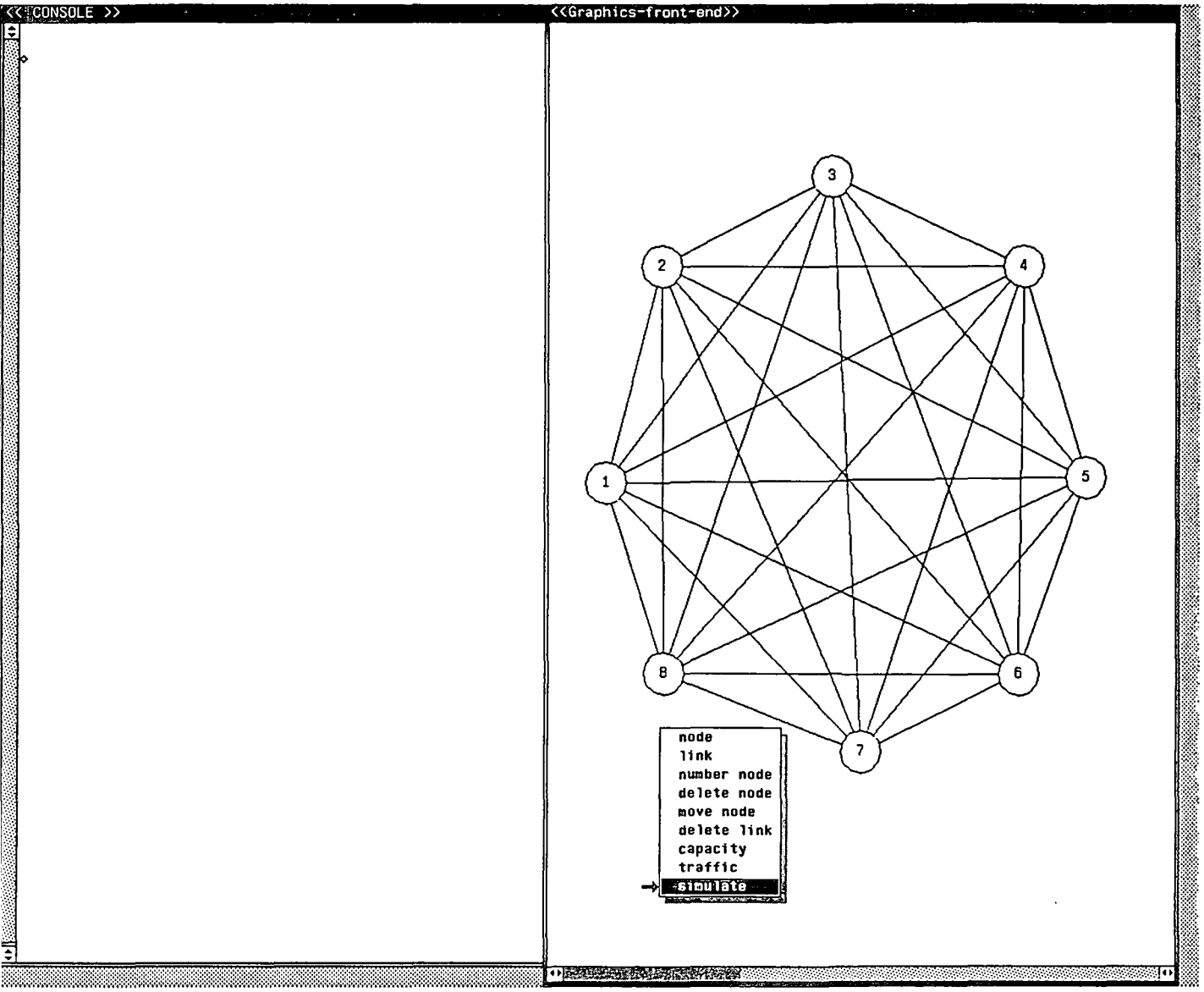


Fig 3.9 An 8 node network drawn on the graphics front end.

Chapter Four

Comparison between different routing algorithms

4.1 - Introduction

In this chapter we perform analytical and simulation studies on the use of different routing procedures in nonhierarchical circuit-switched networks. The routing policies include Fixed Routing (FR), Random Routing (RR), Linear Reward Penalty (L_{R-P}) and Dynamic Alternative Routing (DAR) [11] [21]. A series of simulations are performed to compare different routing policies to L_{R-P} automata schemes under both steady state and failure conditions.

The use of learning schemes for routing traffic in telephone networks was first suggested in [22] as a means of utilizing facilities efficiently in the face of nonstationary loads. Simulation studies in [22] revealed that the most effective use of learning automata routing occurs when overload conditions exists but additional capacity is available elsewhere in the network. In such cases the use of learning automata routing results in a lower blocking probability as compared to fixed rule alternate routing.

Other works [23][24] compared L_{R-I} and L_{R-P} schemes to fixed rule alternate routing in simple networks. It was shown that both schemes always performed as well as the optimum fixed rule. In addition a mathematical model of a simple four node telephone network was formulated. This model permitted the theoretical predictions of optimum network blocking probability. A later work [25] considered telephone traffic routing in more complex hierarchical and generalised networks using both fixed rule and L_{R-I} automata routing. The work continued with a series of experiments on a 10 node network from Bell Labs [25]. The network used has been derived using computer design techniques to obtain optimal link capacities and routing tables given the constraints of the network topology and traffic statistics. It therefore offered an optimal behaviour for fixed rule with which the Learning automata could be compared. It was shown that the L_{R-I} scheme was suboptimal under normal conditions but performed better than fixed rule under abnormal traffic conditions. Since as stated the network was deliberately designed for fixed routing, the inferior results of automata routing under normal conditions are hardly surprising. As will be shown subsequently, work by the present author has significantly improved the performance of the LA for the network.

4.2 - The network model

The network model is fully connected with N nodes labled $i = 1, 2, \ldots, N$. Fig 4.1 shows such a network with five nodes. The lines represent the transmission links

and the nodes represent the switching stations. A direct link from node i to node j is represented by the ordered pair (i, j). Each direct link in the network is composed of a number of trunks (circuits). Each trunk is a voice channel capable of supporting one voice conversation between the nodes at the two ends of the link. The number of trunks per link represents link capacity and is shown by c_{ij}. Trunks are capable of transmission in both directions. If all c_{ij} channels of a link are carrying calls, the link is said to be busy. If a path consists of more than one link and at least one of them is busy, then the path is busy.

A call is considered as a basic unit of circuit-switched traffic. The arriving calls generated at the nodes are modelled by a Poisson process as calls per time unit originating at node i destined for node j. The average arrival rate is represented as λ_{ij}. The call holding time is exponentially distributed with mean $\frac{1}{\mu_{ij}}$, for an Erlang rate of $A_{ij} = \frac{\lambda_{ij}}{\mu_{ij}}$. The call connection and disconnection times are assumed to be very short compared with the call holding time and hence they are considered to be zero.

A call between nodes i and j is offered to the direct path first. If the path is busy, then the call attempts a two link alternate path, (i, k, j). For every origin destination pair there are $N - 2$ such paths available. If the alternate path, (i, k, j) is busy then the call is blocked. Fig 4.2 gives an example of direct and alternate paths for a five node network. A call arrives at node 1 destined to node 2, the shortest path is the direct link (1,2) and alternate paths are (1,3,2), (1,4,2) and (1,5,2).

4.3 - Routing Schemes

4.3.1 - Fixed Routing (FR)

This is one of the simplest forms of routing. Each call is only allowed to attempt one path, the direct path. If the path is busy the call is blocked and is considered lost.

The main characteristic of this scheme is that its blocking performance is very stable in the sense that the blocking rate increases smoothly with the traffic rate. Such a stable blocking performance is especially essential to networks whose traffic rate fluctuates significantly. There are other advantages to using fixed routing. First since each call is only allowed to attempt one path, the total processing time needed for a call is shorter than that for dynamic routing schemes. Second due to the absence of alternately routed traffic, the blocking performance becomes relatively easy to calculate.

However the drawback to fixed routing is that it ignores the possibility of making use of free alternate paths when the direct path is busy and therefore results in a higher blocking probability. The improvement in blocking performance by making use of free alternate paths is only guaranteed for low traffic load in nonhierarchical networks as shown in [26] and [27]. For high traffic load, the blocking performance is much more stable without the use of alternate paths. Therefore fixed routing is preferable to use when the traffic load is high.

Let Aij be the offered traffic to link (i, j) with capacity c_{ij}. Based on the as-

sumption of the Poisson call arrival process, the link blocking probability is given by the Erlang B formula as

$$B_{ij} = E(A_{ij}, c_{ij}) = \frac{A_{ij}^{c_{ij}}}{c_{ij}!} \Big/ \sum_{l=0}^{c_{ij}} \frac{A_{ij}^{l}}{l!} \tag{4.1}$$

The overall network blocking probability, Z can then be calculated:

$$Z = \frac{\sum_i \sum_j A_{ij} B_{ij}}{\sum_i \sum_j A_{ij}} \tag{4.2}$$

4.3.2 - Random Routing (RR)

The random routing scheme operates as follows: a call arriving at the network for the source-destination pair (i, j) is routed along the direct route (i, j) if there is at least one free circuit on the link; otherwise a tandem node k $(k \neq i, j)$ is chosen at random with each of the possible $N - 2$ tandem nodes having equal probabilities. The call is routed along the two-link path (i, k, j) if at least one free circuit exists on both links. If the call fails to be routed along the two-link path, it is lost.

The reason for trying alternate paths is to reduce the blocking rate. The mechanism is similar to the idea of resource sharing. For example, if a direct path is busy, the call may be connected via any of the alternate paths and hence the blocking rate can be reduced. On the other hand, the direct path can also be an alternate path or part of an alternate path for other calls to help reducing their blocking. Therefore, calls with different source destination pairs share a larger bandwidth of channel ca-

pacity for connection. In the fully connected and nonhierarchical network model, this is only the case for low traffic load. For overload, the blocking rate increases rapidly with the traffic rate. This is in fact, the result of network instabilities which will be discussed in detail in a later chapter.

In what follows we explain mathematical models to calculate the overall blocking probabilities for random routing both without and with trunk reservation. First consider the case when no trunk reservation is employed. For the fully connected network model described in section 4.2, let:

A_{ij} be the external offered load to link (i,j)

ν_{ij} be the total offered load to link (i,j)

c_{ij} be the link capacity

$B(i,j)$ be the blocking probability on link (i,j)

$Q(i,j) = 1 - B(i,j)$ be the probability that link (i,j) is available

$P_k(i,j)$ be the proportion of overflow calls between nodes i and j attempting tandem node k.

The total load ν_{ij} offered to a link consists of the external offered load plus alternately routed calls overflowing from other direct paths.

$$\nu_{ij} = A_{ij} + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{ik}B(i,k)P_j(i,k)Q(j,k) + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{jk}B(j,k)P_i(j,k)Q(i,k) \qquad (4.3)$$

For random routing $P_k(i,j) = \frac{1}{N-2}$.

We assume that the external offered and overflow traffics are independent Poisson

streams. Therefore given offered load ν to a link and the number of channels c of the link, the blocking probability of the link is given by Erlang B formula:

$$B = E(\nu, c) = \frac{\nu^c}{c!} \Big/ \sum_{l=0}^{c} \frac{\nu^l}{l!} \tag{4.4}$$

The carried load per link C_{ij} is given as:

$$Cij = A_{ij}(1 - B(i,j)) + \sum_{\substack{k=1 \\ k \neq i \\ k \neq j}}^{N} A_{ij} B(i,j)(1 - L_k(i,j))P_k(i,j) \tag{4.5}$$

Where $L_k(i,j)$ $(k \neq i,j)$ is the blocking on the two-link alternative route via k and is given, using the independent blocking assumption, by:

$$L_k(i,j) = 1 - (1 - B(i,k))(1 - B(k,j)) \tag{4.6}$$

and the overall network blocking probability:

$$Z = \frac{\text{total offered load - total carried load}}{\text{total offered load}}$$

or

$$Z = \frac{\sum_i \sum_j A_{ij} - \sum_i \sum_j C_{ij}}{\sum_i \sum_j A_{ij}} \tag{4.7}$$

Given A_{ij}, c_{ij} and N we can implement the repeated substitution method to solve the above fixed-point equations. Assume initial values for B_{ij}, find ν_{ij}. Given ν_{ij} and c_{ij} implement the Erlang B formula to find new value for B_{ij}. Repeat the process until convergence.

Trunk reservations

When trunk reservation [26] is employed, then first-routed traffic is allowed to access all c trunks of a link trunk group. Overflow traffic is not allowed to access more than $m < c$ trunks of the trunk group. If all m channels are busy, alternately routed calls are blocked from this link. Therefore $r = c - m$ channels are reserved for first-routed traffic only. r is called Trunk Reservation Parameter (TRP).

Let $B_1(i,j)$ be the blocking probability to fresh calls and $B_2(i,j)$ be the blocking probability to overflow calls using link (i,j). The functions B_1 and B_2 differ because trunk reservation is applied to overflow calls. Let $Q_2(i,j) = 1 - B_2(i,j)$ be the link availability for alternately routed calls. When trunk reservation is used then the link blocking probability is no longer given by Erlang B formula [19]. Consider link (i,j), let l be the number of lines in use on the link. For $l \leq m$ the total traffic ν offered to the link consists of both fresh and overflow traffic. The probability of the link being in state l can then be written as:

$$p_l = \frac{\nu^l}{l!} p_0 \qquad 0 \leq l \leq m \tag{4.8}$$

For $l > m$ calls, the trunk reservation mechanism stops alternate routing. Therefore only first routed traffic is present. The offered load of this traffic is A and the probability of a link being in state l is:

$$p_l = \frac{A^{l-m}\nu^m}{l!} p_0 \qquad m < l \leq c \tag{4-9}$$

The link blocking probability is given by

$$B_1 = p_c = \frac{A^{c-m}\nu^m}{c!}p_0 \tag{4.10}$$

The link availability for the alternately routed traffic Q_2 is the probability that the system is in any of the states 0 to $m-1$. This is the probability that no more than $m-1$ trunks are busy.

$$Q_2 = \sum_{l=0}^{m-1} \frac{\nu^l}{l!}p_0 \tag{4.11}$$

The zero state probability is found by summing p_l over all possible states:

$$p_0 = [\sum_{l=0}^{m} \frac{\nu^l}{l!} + \sum_{l=m+1}^{c} \frac{A^{l-m}\nu^m}{l!}]^{-1} \tag{4.12}$$

The equations for B_1 and Q_2 are simplified in Appendix 1 as:

$$B_1 = \frac{\prod_{k=1}^{c-m} \frac{A}{(m+k)}}{\frac{1}{E(\nu,m)} + \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}} \tag{4.13}$$

$$Q_2 = \frac{1 - E(\nu,m)}{1 + E(\nu,m) \times \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}} \tag{4.14}$$

Where:

$$E(\nu,m) = \frac{\frac{\nu^m}{m!}}{\sum_{l=0}^{m} \frac{\nu^l}{l!}}$$

For the N-node fully connected network model, the offered load ν to a link (i,j) is:

$$\nu_{ij} = A_{ij} + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{ik}B_1(i,k)P_j(i,k)Q_2(j,k) + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{jk}B_1(j,k)P_i(j,k)Q_2(i,k) \tag{4.15}$$

And for random routing $P_k(i,j) = \frac{1}{N-2}$.

The total carried load per link, C_{ij} can be calculated :

$$C_{ij} = A_{ij}(1 - B_1(i,j)) + \sum_{\substack{k=1 \\ k \neq i \\ k \neq j}}^{N} A_{ij} B_1(i,j)(1 - L_k(i,j)) P_k(i,j) \tag{4.16}$$

Where

$$L_k(i,j) = 1 - Q_2(i,k)Q_2(k,j) \tag{4.17}$$

The overall network blocking probability can be calculated using Eq(4.7).

$$Z = \frac{\sum_i \sum_j A_{ij} - \sum_i \sum_j C_{ij}}{\sum_i \sum_j A_{ij}} \tag{4.7}$$

The iterative loop is similar to the case without trunk reservation. Assume initial values for $B_1(i,j)$ and $Q_2(i,j)$, find ν_{ij}. Given ν_{ij} and A_{ij}, implement equations (4.13) and (4.14) to find new values for $B_1(i,j)$ and $Q_2(i,j)$, repeat until convergence.

4.3.3 - Linear Reward Penalty (L_{R-P})

The L_{R-P} scheme operates as follows: consider a node pair (i,j), fresh offered traffic between nodes i and j is first offered to the direct link and is always routed along that link if there is a free circuit. Otherwise the call attempts a two-link alternative route via tandem node k_{ij} with trunk reservation applied to both links (i,k) and (k,j). If the call fails to be routed via k, it is considered lost.

In this scheme every alternative route is considered as an action α. At every stage the probability of choosing the kth action α_k is $P_k(n)$ and $\sum_k P_k(n) = 1$. When

a particular action α_k is chosen and results in call completion the probabilities are updated as follows:

$$P_k(n+1) = P_k(n) + a(1 - P_k(n)) \qquad 0.0 < a < 1.0$$

$$P_t(n+1) = (1 - a)P_t(n)$$

When α_k is selected and the call is blocked:

$$P_k(n+1) = (1 - b)P_k(n) \qquad 0.0 < b < 1.0$$

$$P_t(n+1) = P_t(n) + \frac{bP_k(n)}{r-1} \tag{4.18}$$

Where r=number of actions

a = Learning parameter

b = Penalty parameter

t = actions other than k

The next action is chosen using the modified probability distribution at stage (n+1).

In an N node fully connected network there are $N - 2$ alternatives or actions for each node pair (i, j). $P_k(i, j)$ is the probability of selecting the kth action for the node pair (i, j). L_{R-P} starts with assigning equal probabilities to each action. Therefore

initially:

$$P_k(i,j) = \frac{1}{N-2}$$

The theoretical overall blocking probability in this case is calculated using similar technique described in sec 4.3.2 for the random routing scheme except that the proportion of overflow calls $P_k(i,j)$ has to be calculated separately for the L_{R-P} scheme. These proportions are found using similar technique to the one stated in [28]. For the mathematical model B considered in section 2.5.2 it was shown that L_{R-P} equalizes blocking rates. Therefore for every node pair (i,j):

$$P_1(i,j)L_1(i,j) = P_2(i,j)L_2(i,j) = \ldots = P_r(i,j)L_r(i,j) \tag{4.19}$$

Where

$$L_k(i,j) = 1 - Q_2(i,k)Q_2(k,j) \tag{4.17}$$

Equation (4.19) shows that L_{R-P} equalizes the number of blocked calls on each alternative. For example from equation (4.19) we see that if $L_2(i,j)$ decreases, the two-link route via 2 is offered a greater proportion of the alternately routed traffic.

Given $Q_2(i,j)$ the proportions $P_k(i,j)$ can be calculated using Eq (4.17), (4.19) and the relation:

$$\sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} P_k(i,j) = 1$$

The repeated substitution method can be implemented to calculate the overall blocking probability. Assume initial values for $B_1(i,j)$ and $Q_2(i,j)$. The proportions $P_k(i,j)$ are derived from the link availabilities $Q_2(i,j)$, they may then be used to

calculate new estimates for the offered traffic ν_{ij} using Eq (4.15). Given ν_{ij} and A_{ij} equations (4.13) and (4.14) can be implemented to find new values for B_1 and Q_2. The iterative procedure is given in Fig 4.3.

4.3.4 - Dynamic Alternative Routing (DAR)

DAR operates as follows: a call arriving at the network for the source destination pair (i,j) attempts the direct path (i,j) first, if the path does not have any free circuit then a random tandem node k $(k \neq i,j)$ is selected and the call is offered to the two-link path (i,k,j) with trunk reservation applied to both links (i,k) and (k,j). If the call fails to be routed along the two link path, next time when a call arrives for the node pair (i,j) and the direct path is busy, a random tandem node is selected. If the call is successfully routed along the path (i,k,j), next time that a call arrives for the node pair (i,j) and the direct path is busy the same tandem node k is selected deterministically. This scheme was developed by Cambridge University in association with British Telecom [11] [21]. It is shown in [11] that DAR equalizes blocking rates. Therefore for every node pair (i,j):

$$P_1(i,j)L_1(i,j) = P_2(i,j)L_2(i,j) = \ldots = P_r(i,j)L_r(i,j) \tag{4.19}$$

The theoretical blocking probability for DAR strategy is calculated using the same method explained for L_{R-P} in section 4.3.3.

4.4 - Examples

Examples of the application of the routing strategies in this chapter are given in the following experiments where analytical results are verified using simulations. Also, some experiments are performed to study the behaviour of the different routing algorithms under failure conditions. First we explain the test network. Consider the five node fully connected network of Fig 4.1 with traffics $\underline{\lambda} = (\lambda_{ij}, i < j)$ and capacities $\underline{c} = (c_{ij}, i < j)$ given by:

$$\underline{\lambda} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 1734 & 1314 & 600 & 984 \\ - & - & 536 & 309 & 4011 \\ - & - & - & 6279 & 278 \\ - & - & - & - & 195 \\ - & - & - & - & - \end{pmatrix} \end{array}$$

$$\underline{c} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} - & 1710 & 1410 & 630 & 1050 \\ - & - & 570 & 300 & 4260 \\ - & - & - & 6360 & 270 \\ - & - & - & - & 180 \\ - & - & - & - & - \end{pmatrix} \end{array}$$

The call holding time is one time unit. One time unit is the time that the network receives an average of $\underline{\lambda}$ calls. The traffics $\underline{\lambda}$ were chosen to represent large routes in the planned British Telecom digital network. The link capacities were derived by

including:

- fixed route dimensioning using Erlang's formula
- the effect of modularity to groups of 30 circuits
- random effects due to forcasting and the upgrading of circuits.

Thus, an attempt was made to model the sort of mis-match between traffics and capacities likely to be present in real communication networks.

4.4.1 - Experiment 1

In this experiment we use our simulation model to verify the load equalization property of DAR and L_{R-P} for the test network. Consider link $(1,2)$ in Fig 4.1 with $c_{12} = 1710$ and $\lambda_{12} = 1734$. The following table for TRP=0 shows the average blocking rates for three possible alternatives $(1,3,2)$, $(1,4,2)$ and $(1,5,2)$.

	1-3-2	1-4-2	1-5-2
L_{R-P}	.0666	.0646	.0772
DAR	.0643	.0686	.0689

As can be seen as predicted by the theory the two algorithms allowing for the simulation variance, equalize blocking rates with DAR giving a better equalization of blocking rates in this case. Similar results have been obtained for both L_{R-P} and DAR with varying TRP with as expected an increase in the blocking rates as TRP is increased.

In this and the following experiments, the simulator was run for a very long time

(200 time units for 0% and 100 time units for 10% and 20% overloads) and blocking probabilities were recorded every one unit of time during the run. The network reached the steady-state in a few time units. This was confirmed by calculating blocking probabilities over equal intervals of the simulation results and observing that the time fluctuations of the blocking probabilities were small. Moreover, calculations showed that the effect of initial bias on the results was negligible.

4.4.2 - Experiment 2

The second experiment was performed to calculate the theoretical blocking probabilities for FR, RR, L_{R-P} and DAR and compare them with simulation results. The blocking probabilities were calculated for offered traffics (1+OVL)λ for values of an overload factor OVL using equations 4.2 and 4.7. The table below shows blocking probabilities for FR at 0%, 10% and 20% overload values.

OVL	0%	10%	20%
theory	.868	6.95	14.73
simulation	.85	7.02	14.43

As can be seen there is a good agreement between theoretical and simulation results.

Figs 4.4, 4.5 and 4.6 show theoretical and simulation results for RR, L_{R-P} and DAR respectively. The curves show the variation of blocking probability versus TRP for 0%, 10% and 20% overload values. The bars show 90% confidence intervals. For example consider Fig 4.5 for L_{R-P}. It is shown that as the trunk reservation parameter drops below 5 and especially when it drops from 1 to 0 the blockings increase

dramatically since the additional alternative routing only damages the performance under these circumstances. The results for 0% overload show that blocking probabilities are minimum around TRP=7, while 10% and 20% overload results show a continuous drop in blockings as TRP increases and above TRP=10 the results seem insensitive to the changes in TRP.

The results in Figures 4.4, 4.5 and 4.6 show an excellent agreement between theoretical and simulation results above TRP=5. It can be seen that simulation and analytical models do not have a good agreement for small TRP's. Therefore the accuracy of the model degrades for small TRP's and this is particularly noticable for the case where TRP=0 and there is a general overload of 0%. With higher overloads of 10% and 20% this effect is also present but to a much lesser degree. The table below compares numerical values of simulation and analysis for L_{R_P} with TRP=0 and three overload factors. Δ represents the difference between simulation and model results.

TRP=0

OVL	Simulation	model	Δ
0%	1.22	3.43	2.21
10%	12.96	14.06	1.10
20%	20.97	21.60	.63

It can be seen that the relative error is smaller as the overload factor is increased. Simulation variance is proportional to the stochastic behaviour of the network and this in turn depends on the degree of flexibility for routing. As OVL increases (i.e. less fluctuating) then simulation variance decreases and hence simulation results approach

theory.

Figure 4.7 compares theoretical blocking probabilities between L_{R-P} and FR for different overload values. It is shown that when no TRP is used, FR outperforms L_{R-P}. When TRP=7, L_{R-P} is slightly better than FR for overload values less than 3% and poorer for overloads greater than 3%. Similar results are given in Fig 4.8 for RR.

Simulation results given in the following tables for TRP=0 and TRP=10 compare the average blocking probabilities for different routing algorithms. Also included in comparison tables are results for the Least Busy Alternative (LBA) [29] routing strategy where for each call between nodes i and j that fails to be routed on the direct link the tandem node k_{ij} is selected by computing the two-link alternative route which currently has the most free circuits to carry calls between nodes i and j. This tandem node is then used subject to trunk reservation.

TRP=0

OVL	FR	RR	L_{R-P}	DAR	LBA
0%	.85	1.23	1.22	1.44	3.92
10%	7.02	12.31	12.96	12.90	16.90
20%	14.43	20.34	20.97	20.75	24.61

TRP=10

OVL	FR	RR	L_{R-P}	DAR	LBA
0%	.85	.57	.52	.52	.48
10%	7.02	7.12	7.25	7.31	7.28
20%	14.43	14.32	14.59	14.51	14.23

The above results indicate that for TRP=0 all routing techniques are inferior to FR and for TRP=10 they all perform as well as FR. From the above tables it can be seen that RR, L_{R-P} and DAR perform very close to each other. In the next experiment these three algorithms and FR are compared under failure conditions.

4.4.3 - Experiment 3

The test network was simulated for the case of 10% overload and the routing policies: FR, RR, DAR and L_{R-P} (with three sets of reward-punish parameters $(a = b = 0.1)$, $(a = 0.1, b = 0.01)$, $(a = 0.1, b = 0.001)$. While the network was operating in the steady state, one link (e.g. 1-2) was disconnected to study the effect on overall blocking probabilities. The tables below contain numerical results for the above mentioned routing policies and TRP=0 and 10.

TRP=0

	FR	RR	DAR	$L_{R-P}^{(.1,.1)}$	$L_{R-P}^{(.1,.01)}$	$L_{R-P}^{(.1,.001)}$
Z%	16.57	22.84	23.27	23.54	24.44	24.55
NAC	0	640	653	660	660	630

TRP=10

	FR	RR	DAR	$L_{R-P}^{(.1,.1)}$	$L_{R-P}^{(.1,.01)}$	$L_{R-P}^{(.1,.001)}$
Z%	16.57	16.92	16.80	16.97	16.81	16.84
NAC	0	74.80	78.07	75.80	80.57	78.01

Where Z is the average network blocking probability

NAC is the average number of calls accepted per link (1,2)

The tables indicate that (except for the case of FR), a) blocking probabilities are similar for different routing schemes, and b)blocking probabilities are higher for TRP=0 than for TRP=10. The network's internal operation in terms of accepted calls for the failed link is shown in the same tables.

In the previous experiment FR was superior to dynamic routing schemes under steady state conditions. This experiment shows that under failure conditions FR still results in a lower network blocking probability than dynamic routing schemes. The disadvantage of FR is that under link failure there will be no communications between origin and destination of the failed link. With dynamic routing schemes and TRP=0 when link (1,2) fails a considerable amount of calls are routed using alternatives (1,3,2), (1,4,2) and (1,5,2). This is shown in terms of the number of accepted calls, NAC per link (1,2) in the above tables. For TRP=10 less dynamic routing is allowed in the network and the number of accepted calls per link (1,2) is significantly decreased. Also it can be seen from the tables that in this particular example the change in learning parameters does not have any significant influence on the network blocking probability.

4.4.4 - Experiment 4

Consider the origin destination pair (1,3), when a call arrives, if no direct link is available one of the three alternatives (1,2,3), (1,4,3) or (1,5,3) is examined. When link (1,2) fails the arc (1,2,3) is a failed alternative. With the RR and DAR schemes

the probability of selecting (1,2,3) is always high because the two algorithms cannot detect any failure in the network. In the case of the L_{R-P} scheme the probability of selecting the failed arc is expected to converge to zero because the algorithm can detect the failure by adjusting the probabilities. The table below shows the numerical values of selecting probabilities, $P_2(1,3)$ for failed arc (1,2,3). These values are given for three sets of L_{R-P} parameters and two TRP's, 0 and 10.

TRP=0

	$L_{R-P}^{(.1,.1)}$	$L_{R-P}^{(.1,.01)}$	$L_{R-P}^{(.1,.001)}$	DAR
$P_2(1,3)$	.21	.060	.0077	.26

TRP=10

	$L_{R-P}^{(.1,.1)}$	$L_{R-P}^{(.1,.01)}$	$L_{R-P}^{(.1,.001)}$	DAR
$P_2(1,3)$	.30	.20	.045	.31

The table for TRP=10 does not show steady state results because the total number of times that alternatives are tried is small and consequently much more runs are needed to observe steady state probabilities. The TRP=0 results show that when $a = b = 0.1$ is tried the probabilities are high but if b is decreased (i.e.reduced penalty probability) the probabilities converge to smaller numbers which means that there is a very low probability of selecting the failed arc.

4.4.5 - Experiment 5

There are two questions which need to be discussed as a consequence of the previous experiments.

(a) - Why is the network so sensitive to small TRP's (e.g. TRP=5) compared to large link capacities (e.g. c_{15}=1050)? (b) - How does TRP affect the spread of traffic over the entire network. To answer the first question, in a typical simulation program [L_{R-P}, 10% overload and TRP=0] the number of free circuits per links were recorded. A small sample of the results is shown below. The results show the number of free circuits, FC at an instant of time as a call arrives.

call arrives at node 3 with destination node 5

FC(1,2)=0

FC(1,3)=3

FC(1,4)=1

FC(1,5)=1

FC(2,3)=1

FC(2,4)=1

FC(2,5)=7

FC(3,4)=7

FC(3,5)=0

FC(4,5)=0

call routed via node 1

call arrives at node 3 with destination node 4

FC(1,2)=1

FC(1,3)=2

FC(1,4)=1

FC(1,5)=0

FC(2,3)=1

FC(2,4)=1

FC(2,5)=7

FC(3,4)=6

FC(3,5)=0

FC(4,5)=0

call routed directly

In the first set of results for a call arriving at node 3 with destination 5 there are no free circuits on the direct link and the call is routed via node 1 along the routes (1,3) and (1,5). In the second case the call is routed directly. The above results show that the number of free links are very small and comparable to small TRP's. This is why the network is sensitive to small TRP's in spite of large link capacities.

To answer the second question, the network's traffic loading distribution with LBA and 20% overload are drawn for two cases of TRP=0 and 10, Figs 4.9 and 4.10. Consider Fig 4.9, the horizontal axis represents 40 paths including 10 direct and 30 two-link alternatives. Direct paths are indicated by upward arrows. The vertical axis shows the number of calls in progress on each path. For example path 1 is the direct link between nodes 1 and 2; the vertical axis shows the number of calls in progress on that path. Path 2 is the alternative (1,3,2) with the vertical axis showing the number of calls in progress on the path. Comparison of Figs 4.9 and 4.10 shows that there is a better spread of traffic for TRP=0 than for TRP=10. The TRP=10 figure indicates

that traffic load is mainly on direct routes and is very small or zero on alternatives. In fact the network's behaviour is very much the same as FR when TRP=10 or more is selected.

4.5 - Summary

Simulation studies in experiment one verified theoretical results explained in the previous chapter in terms of the convergence characteristics of the L_{R-P} scheme. In the second experiment the theoretical blocking probabilities for FR, RR, L_{R-P} and DAR were compared to simulation results for different values of overload and TRP. An excellent agreement between analytical and simulation results was found for $TRP > 5$. For all routing strategies the results illustrate the sensitivity of the performance to the parameter TRP. In each case provided TRP is greater than 10 the results are invariant for further increase in TRP. Comparison tables given in experiment two indicate that with TRP=0, all dynamic routing schemes (RR, LBA, L_{R-P} and DAR) are inferior to FR. The test network is designed for fixed routing and as a result any attempt to introduce alternate routing only deteriorates the performance in these circumstances. Comparison tables with TRP=10 indicate that all routing policies perform as well as FR. This is hardly surprising since as shown in experiment five under these conditions a small amount of alternative routing of traffic takes place.

Experiment three shows results for link failure at 10% overload conditions. It

may be seen that with TRP=0 all routing strategies are inferior to FR but with TRP=10 the results for the various strategies are very similar. However in terms of the number of accepted calls for the failed link (1,2), TRP=10 results show a significant deterioration compared to TRP=0. For dynamic routing to be effective for a traffic source directly affected by the failed link, a zero TRP is needed. Clearly a compromise is involved between the effective capacity on alternative links for a traffic source λ_{12} and the overall blocking probability for the network. Effectively priority can be given to λ_{12} by reducing the TRP values for the alternative links while the rest of the links in the network retain TRP=10.

The test network considered in this chapter is designed for FR and as was demonstrated in different experiments, FR resulted in a lower network blocking probability under both steady state and failure conditions. In the next chapter we study the behaviour of different routing algorithms on networks which are designed for dynamic routing strategies. Specifically we will compare learning automata to the DAR scheme.

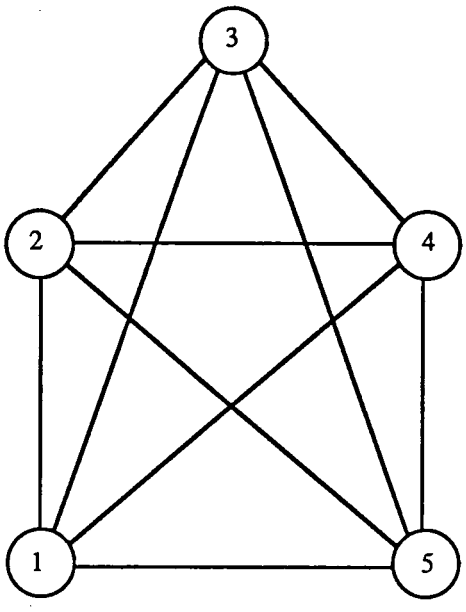


Fig 4.1 A 5-node fully connected network.

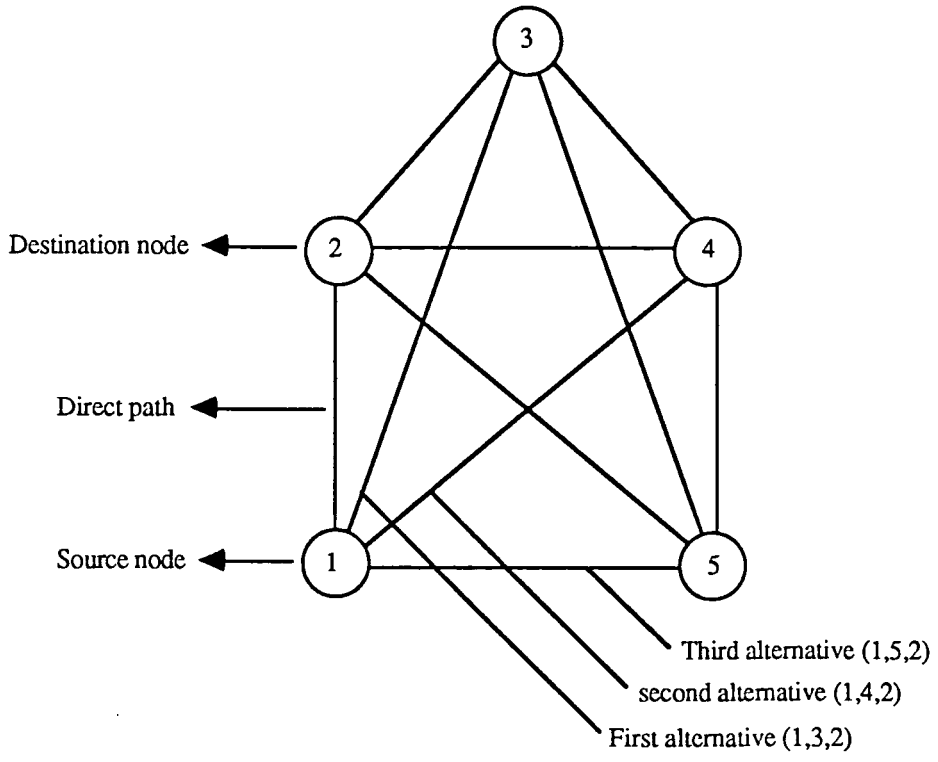


Fig 4.2 An example of direct and alternative paths for a 5-node network.

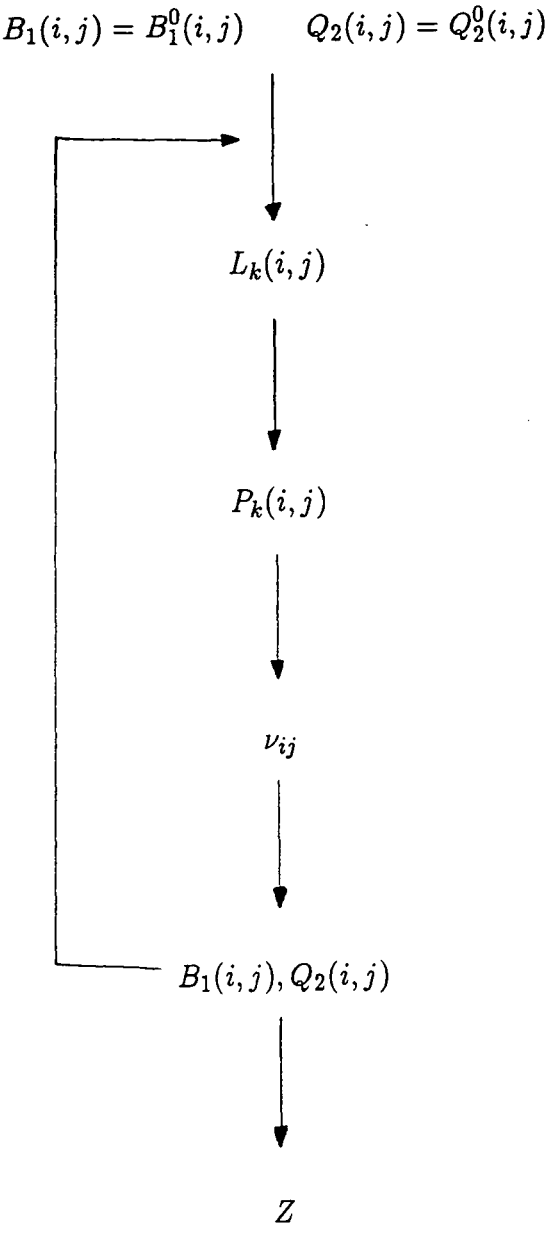


Fig 4.3 Iterative loop to calculate the theoretical blocking probability for L_{R-P}.

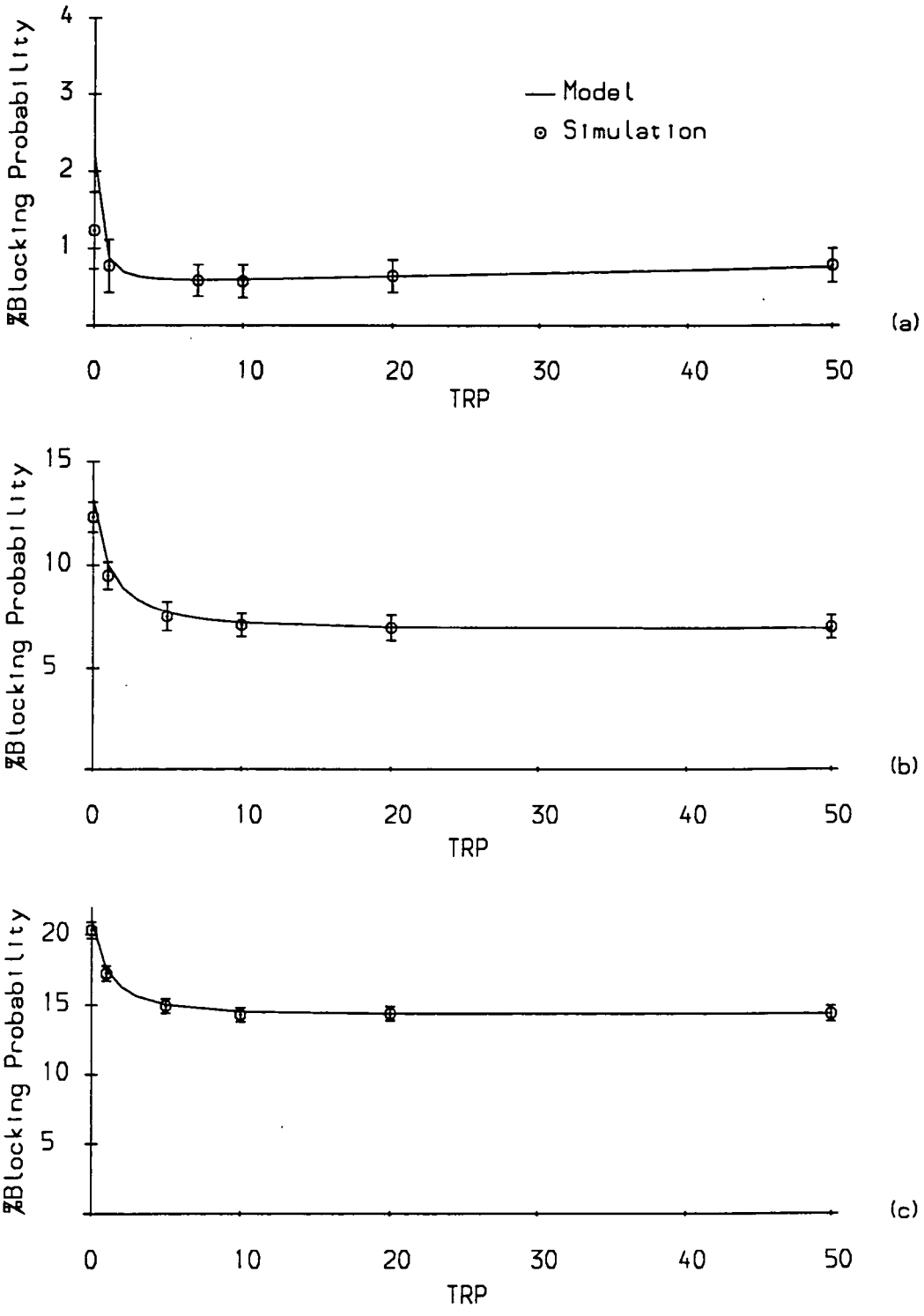


Fig 4.4 Theoretical and simulation results for the five node test network with RR scheme, a) 0%, b) 10%, c) 20% overload.

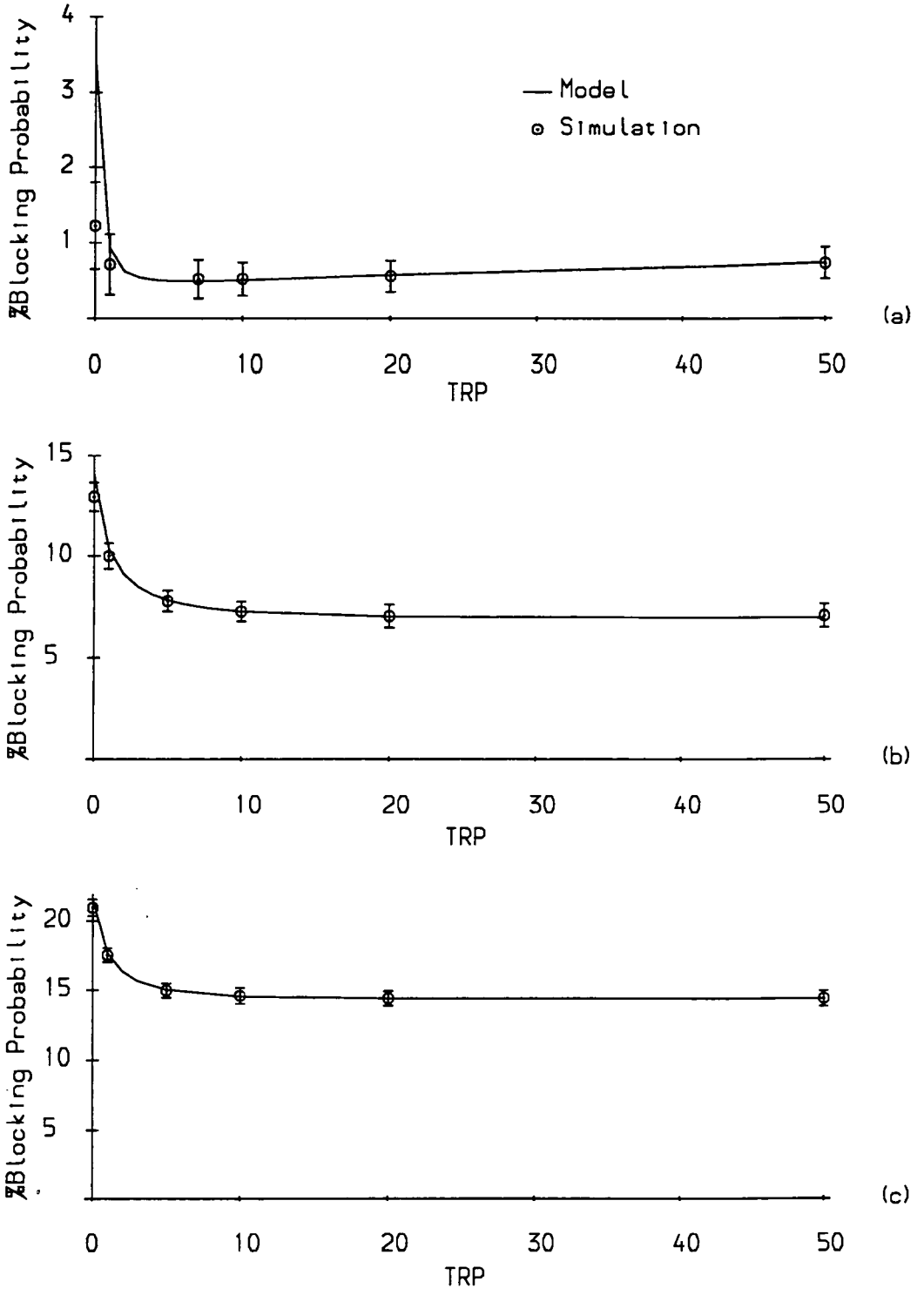


Fig 4.5 Theoretical and simulation results for the five node test network with LRP scheme, a) 0%, b) 10%, c) 20% overload.

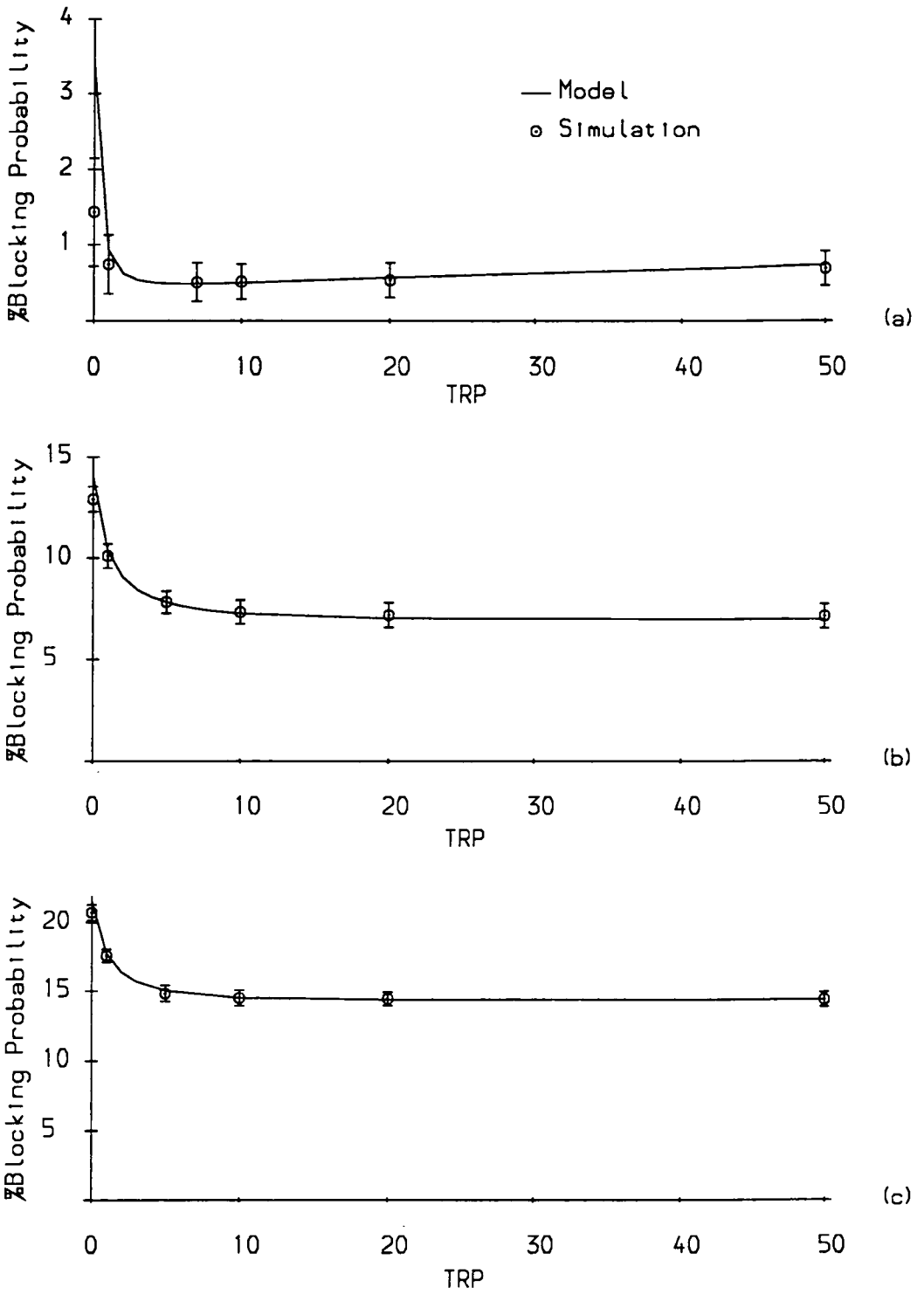


Fig 4.6 Theoretical and simulation results for the five node test network with DAR scheme, a) 0%, b) 10%, c) 20% overload.

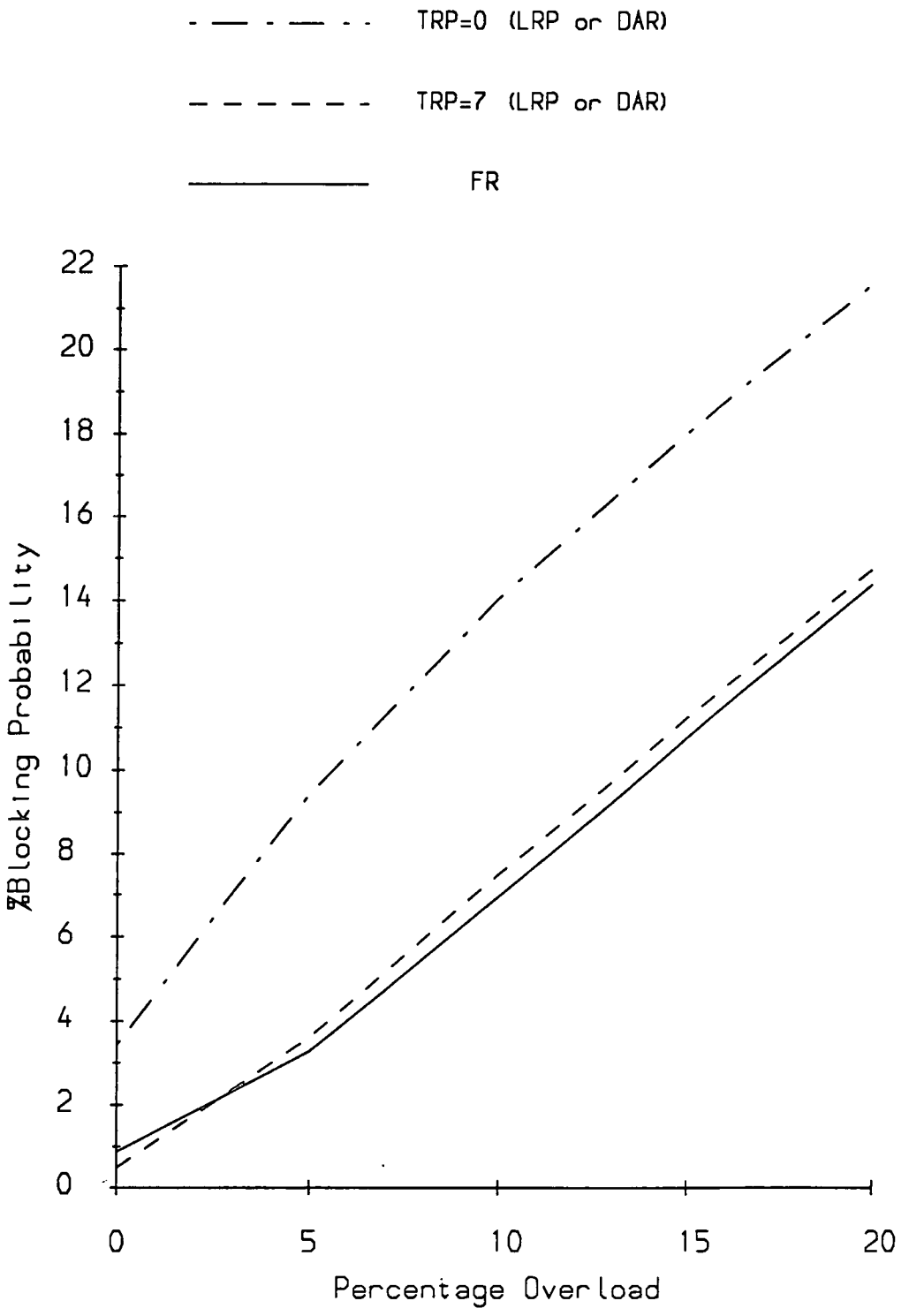


Fig 4.7 Theoretical Results for LRP, DAR and FR.

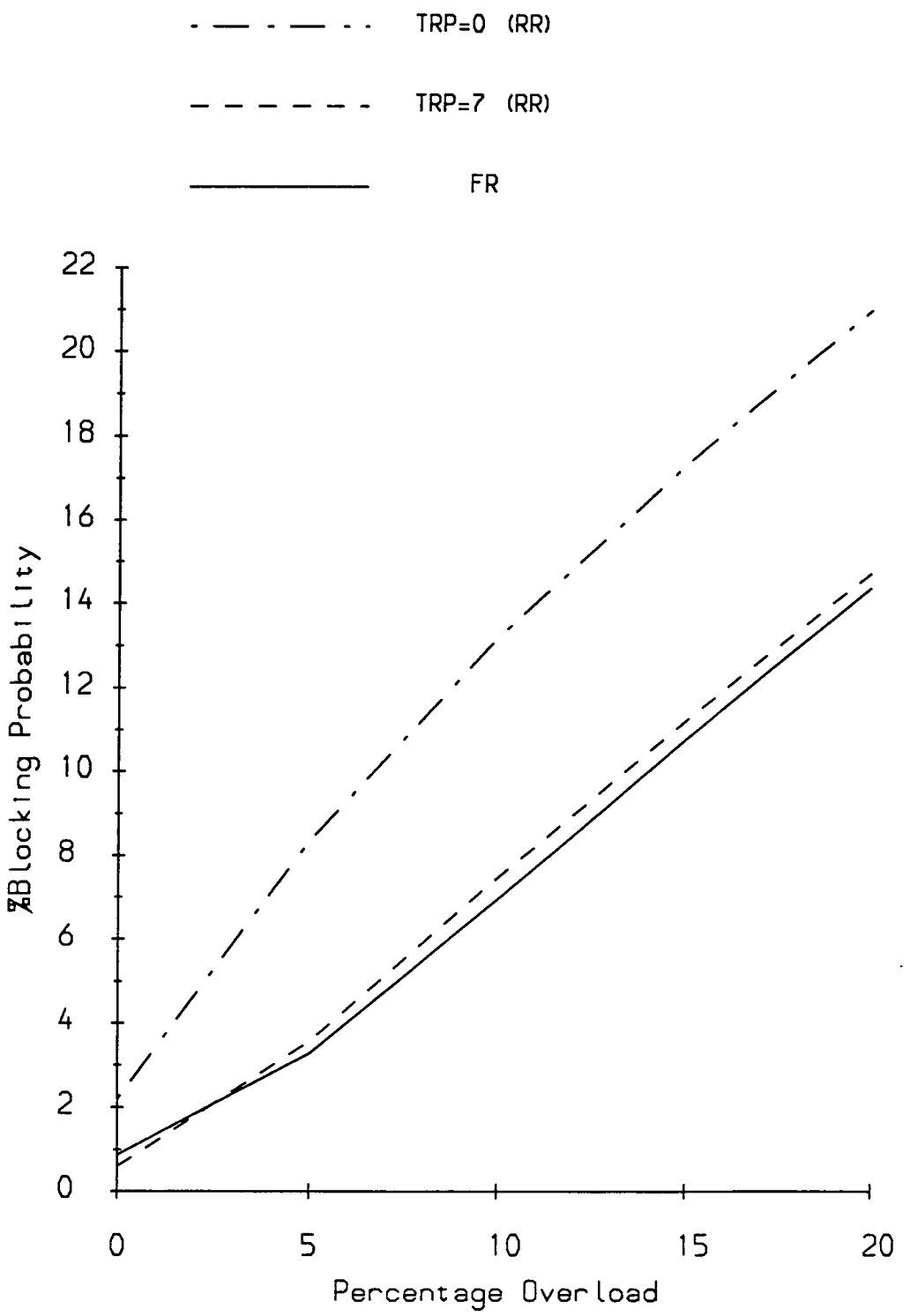


Fig 4.8 Theoretical Results for RR and FR.

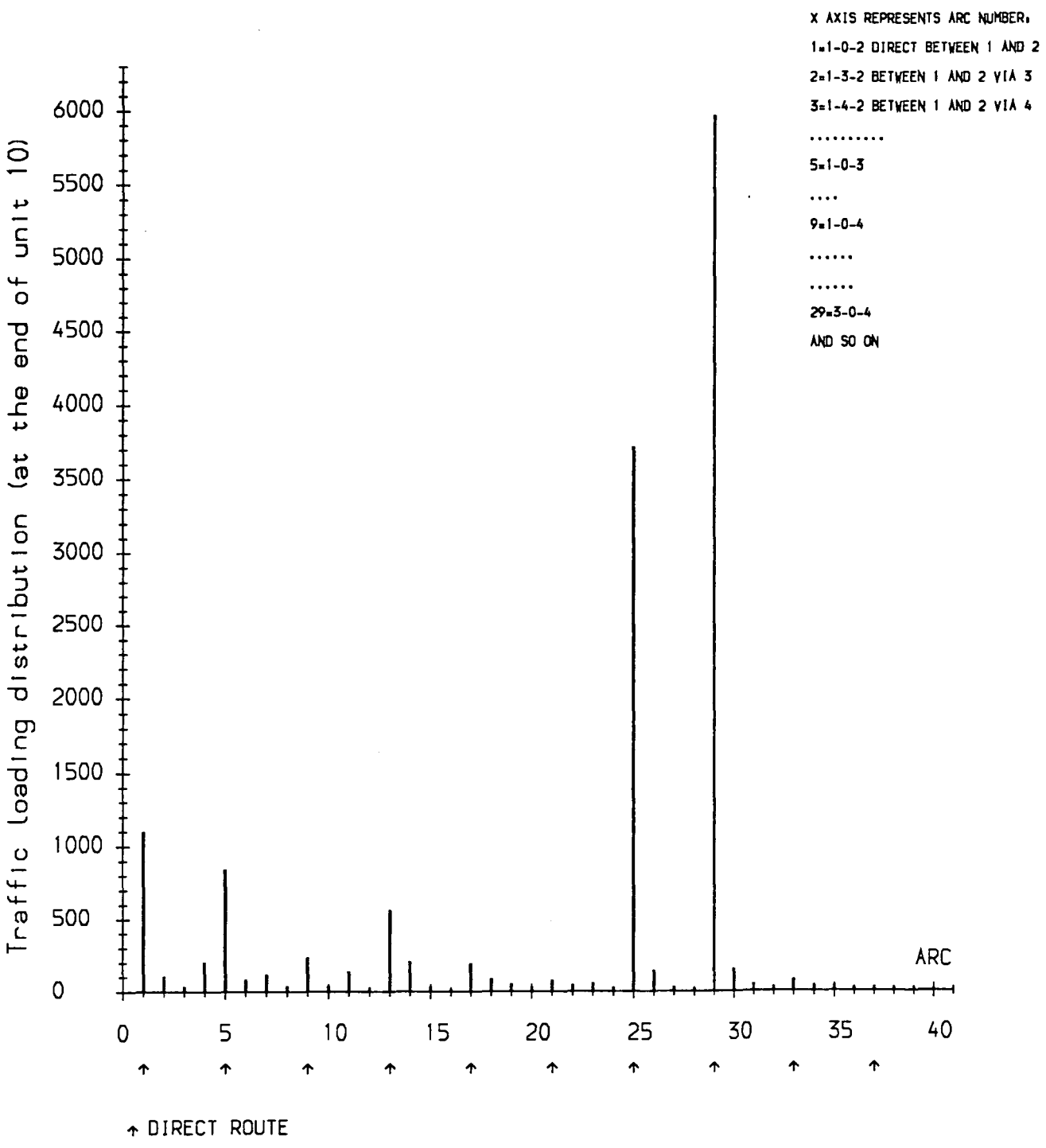


Fig 4.9 Traffic loading distribution, LBA, TRP=0, 20% overload.

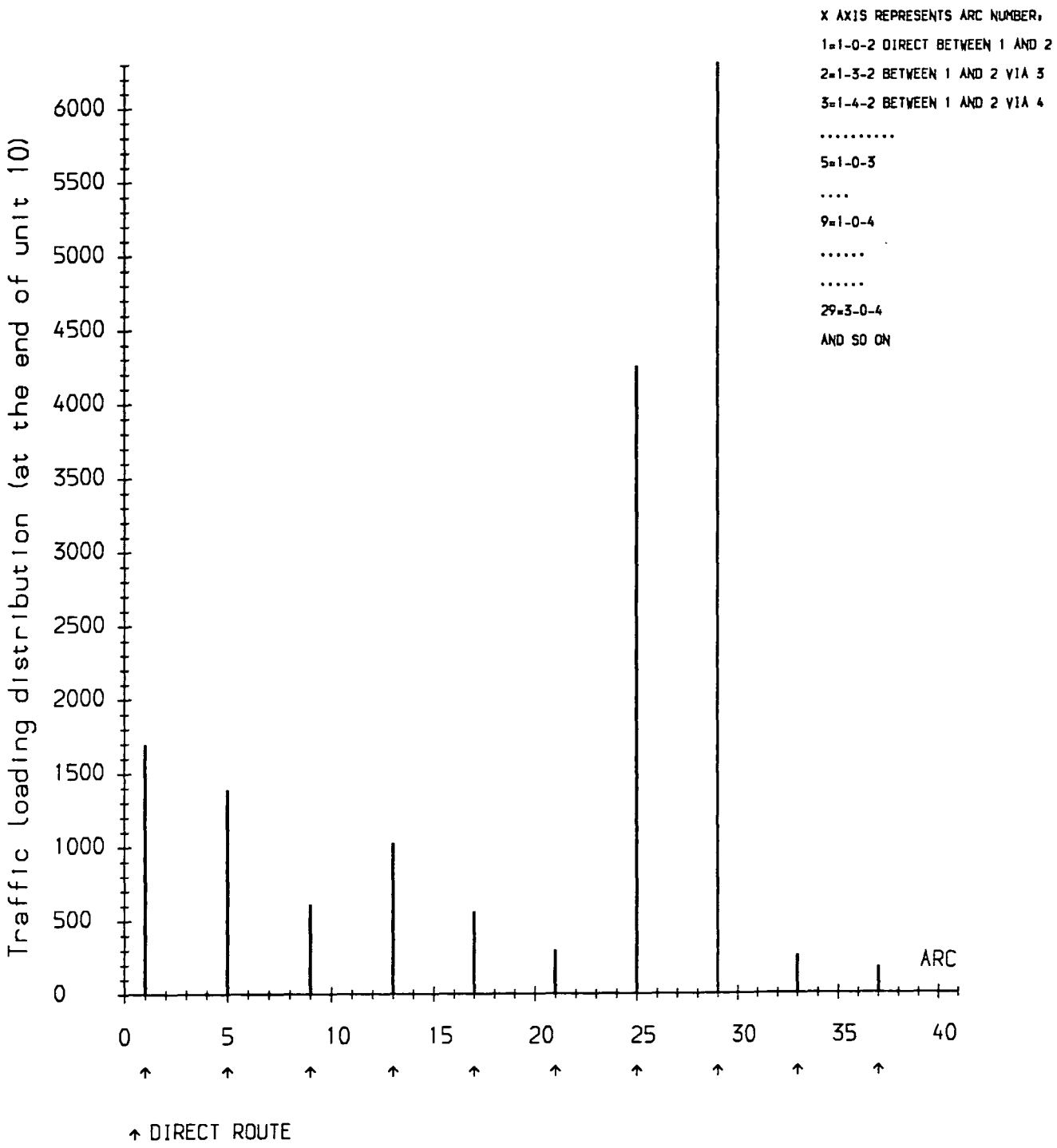


Fig 4.10 Traffic loading distribution, LBA, TRP=10, 20% overload.

Chapter Five

Learning Automata & Dynamic Alternative Routing

5.1 - Introduction

In chapter four the performance of different routing algorithms were compared for a network which was designed for FR and as a result FR gave the best performance. In this chapter the performance of Learning Automata (LA) is compared with DAR on small networks with link capacities and traffic patterns chosen to force dynamic routing. From the routing strategies studied we have selected DAR for further comparison with LA because DAR has recently been implemented by BT and therefore provides a good reference for comparison with LA.

The test networks used in this chapter are shown in figures 5.1, 5.2 and 5.3. Network 1 is a four node network, network 2 is a five node fully connected one used elsewhere [7] and network 3 is a five node with links (1,3) and (3,5) assumed to have failed.

5.2 - Tuning dynamic algorithms

Some dynamic algorithms have parameters that can influence their performance significantly. For example the time between updates for delta routing [7]. The performance of the L_{R-I} scheme can be improved by properly tuning the learning parameter. Consider the network of Fig 5.1 where calls arrive from node 1 to be routed to node 3. The automaton at node 1 selects action 1 (path (1,2,3)) with a probability P_1 and action 2 (path (1,4,3)) with a probability P_2. Let B_1 and B_2 be the blocking probabilities of the two alternate paths, (1,2,3) and (1,4,3) and Z be the overall blocking probability.

Fig 5.4 shows the influence of the learning parameter on the performance of the algorithm. The test network and its traffic input are shown in the same figure. From Fig 5.4 it can be seen that the blocking probability is very high for $a > .01$ and drops significantly as a decreases with the minimum Z at $a = .001$. For $a < .001$ the blocking probability increases until $a = 0.0$ which is actually random routing. The reason for the poor performance with $a > .01$ is that the algorithm converges to the selection of the path with a higher link capacity, path (1-2-3), (Fig 5.4) and as a result the available capacity of the other path, (1-4-3) will not be utilized. In other words when blocking probabilities are coarsely adjusted the algorithm locks on one path. Simulation results with $.01 < a < .02$ showed the sensitivity of the network to initial seed values for the simulator. For example for $a = .015$, we obtained $Z = 4.65\%$ with one seed and $Z = 20.97\%$ with another seed value. This indicates that for this

range of the learning parameter, there is always a high probability that the algorithm converges to the selection of one path.

The convergence properties of the L_{R-I} scheme is discussed in [10]. For a two action L_{R-I} automaton it has been shown that $\lim_{n\to\infty} \Delta P_1(n) = 0$ and this can happen in three ways:

a) $P_1(n) \to 0$

b) $P_1(n) \to 1$

c) $E[B_2(n) - B_1(n)|\hat{P}_1(n)] \to 0$

The first two conditions correspond to the absorbing states of the L_{R-I} scheme. The third case represents the equality of the expected values of blocking probabilities B1 and B2 of the two alternate paths (1,2,3) and (1,4,3). For the curve in Fig 5.4, when $a > .01$ then $P_1 = 1$ and when $a = .001$, the average blocking probabilities for the two alternate paths (1-2-3) and (1-4-3) are 1.57% and 1.21% which are very close to each other.

This experiment has been repeated for another two sets of link capacities shown in Figs 5.5 and 5.6. In Fig 5.5 where the capacities of the two alternate paths are equal, the minimum blocking probability is at a=0.0 (random routing) as expected. Fig 5.6 shows that the minimum Z is at a=.001, therefore changing the capacities of alternate paths does not have a significant effect on the optimum value of the learning parameter.

The simulation results presented in this chapter were obtained for the optimized algorithm, i.e. where the value of the learning parameter was selected to give the best

performance for the problem studied.

5.3 - Simulation I : Four node network

5.3.1 - EXP 1:

For network 1:

$c_{12} = 50$ $c_{14} = 30$

$c_{23} = 50$ $c_{34} = 30$

The traffic from node 1 to 3, A_{13} changes over the range from 30 to 110 Erlangs. The network was simulated for L_{R-I}, DAR and L_{R-P}. The steady state blocking probabilities are tabulated below:

	Z%			
A_{13}	L_{R-I}	DAR	L_{R-P}	$DAR - L_{R-I}$
30	.0	.014	.0	.014
40	.03	.90	.045	.87
50	.28	2.36	.173	2.12
60	1.54	4.26	1.21	2.8
70	4.91	6.86	4.79	2.0
80	11.04	11.58	11.21	.6
90	18.27	18.49	18.00	.2
100	24.80	25.06	24.96	.3
110	30.58	30.70	30.31	.1

The parameters used are $a = .001$, $b = .001$ for L_{R-P} and $a = .001$ for L_{R-I}. The values for reward and penalty parameters for L_{R-P} have been chosen experimentally to give the best performance in this example. Fig 5.7 shows plots of network blocking

probability against traffic arrivals for L_{R-I} and DAR. The above table shows that L_{R-I} and L_{R-P} perform very close to each other and they both are superior to DAR. The last column shows the difference in blocking probabilities for L_{R-I} and DAR, which is most significant over a traffic range between about 40 to 80 Erlangs. When A_{13} is smaller than 40 Erlangs, the blocking probability is small for both schemes because there is sufficient capacity available. When A_{13} is higher than 80 Erlang a large proportion of calls will be lost and the two schemes utilize the available capacity.

For A_{13} between 40 and 80 Erlangs, the routing schemes play a more important role because the traffic should correctly be divided between the two paths. The superiority of L_{R-I} to DAR suggests that L_{R-I} should have a better spread of traffic as will be shown later. For the rest of this chapter, L_{R-I} will be compared to DAR. However in practice, $L_{R-\epsilon P}$ is preferable to L_{R-I} as a small amount of penalty prevents the algorithms convergence to the selection of one action.

5.3.2 - EXP 2:

For network 1:

$c_{12} = 60$ $c_{14} = 20$

$c_{23} = 60$ $c_{34} = 20$

A_{13} changes over a range from 30 to 100 Erlangs.

This experiment investigates the difference in performance of L_{R-I} and DAR for a four node network similar to the previous experiment but with different link

capacities. We are interested to see how the algorithms perform when the ratio of the effective link capacities are higher. In the previous example the ratio was 5/3 but here it is 6/2 with the total useful capacity constant.

First, two sets of simulations have been performed for L_{R-I} (a=.01) and L_{R-I} (a=.001) with $A_{13} = 60$ Erlangs. The steady state results are:

	$L_{R-I}^{(.001)}$	$L_{R-I}^{(.01)}$
Z%	1.43	8.08

The reason for the high blocking probability with $L_{R-I}^{(.01)}$ is that it locks on the upper path (more capacity) as was explained before in section 5.2.

To compare L_{R-I} with DAR, simulations have been performed for each algorithm and for the above specified traffic range.The learning parameter is $a = .001$. The steady state results are listed below:

	Z%		
A_{13}	L_{R-I}	DAR	$DAR - L_{R-I}$
30	.57	.0	-.57
40	.06	.02	-.04
50	.1	1.07	.97
60	1.43	2.62	1.19
70	5.02	6.24	1.22
80	10.93	11.41	.48
90	18.59	18.87	.28
100	25.61	25.09	-.52

Fig 5.8 shows plots of network blocking probability against traffic arrivals for the two algorithms. From figures 5.7 and 5.8 it can be seen that the two algorithms perform

closer when the link capacity ratio is higher. This reflects the fact that DAR performs better the higher the ratio of the link capacities of the two alternate paths. Under these conditions DAR requires less switching between alternate paths. Each time the DAR algorithm switches, a call is lost. When less switching is required then DAR loses less calls.

5.3.3 - EXP 3:

For network 1:

$c_{12} = 40$ $c_{14} = 40$

$c_{23} = 40$ $c_{34} = 40$

λ_{13} changes from 30 to 110.

This experiment is similar to the two previous ones but with a 1/1 link capacity ratio. The steady state blocking probabilities are tabulated below. The reward parameter is $a = .001$.

	Z%		
λ_{ij}	L_{R-I}	DAR	$DAR - L_{R-I}$
30	.0	.26	.26
40	.02	1.51	1.49
50	.1	2.78	2.68
60	1.42	4.55	3.13
70	5.55	7.52	1.97
80	11.17	12.78	1.61
90	16.74	18.19	1.45
100	23.77	24.01	.24
110	30.59	30.77	.18

Fig 5.9 shows plots of the above results. The results indicate that the difference between the two algorithms is higher than the two previous experiments. Comparison between this experiment with the two previous ones shows that as the ratio of the effective capacity of the upper to lower path is increased, the two algorithms perform closer to each other. As was stated previously, each time the DAR switches, a call is lost. As the ratio of the capacity of the two alternative paths approaches unity, more switching will be required and as a result DAR loses more calls.

5.3.4 - EXP 4:

For network 1:

$c_{12} = 50$ $c_{14} = 30$

$c_{23} = 50$ $c_{34} = 30$

A_{13} = (30,120) Erlangs

In this experiment we calculate the optimum probabilistic split for traffic A_{13}. We then compare the optimum network performance to DAR and L_{R-I} schemes. The routing strategy considered is Proportional Routing (PRO) with A_{13} offered independently to each of the paths (1,2,3) and (1,4,3) with probabilities P_1 and P_2. The aim is to find the optimum probabilistic split of the traffic A_{13}. Assume Z to be the overall blocking probability and B_1 and B_2 be the blocking probabilities along paths (1,2,3) and (1,4,3). Then:

$$Z = P_1 B_1 + P_2 B_2$$

The blockings B_1 and B_2 of the two alternate paths are calculated using Erlang's formula:

$$B_1 = E(c_{123}, P_1 A_{13})$$

$$B_2 = E(c_{143}, P_2 A_{13})$$

Where c_{123} and c_{143} are effective capacities along paths (1,2,3) and (1,4,3).

Fig 5.10 shows a plot of B_1, B_2 and Z versus P_1 for network 1 with $c_{12} = c_{23} = 50$, $c_{14} = c_{43} = 30$ and $A_{13} = 60$. As can be seen the overall blocking probability Z is minimum at the intercept of the B_1 and B_2 curves corresponding to $P_1 = .65$ which is the optimum value for P_1. Fig 5.11 shows a plot of Z calculated for the optimum probabilistic split for a range of traffic A_{13}. Simulation results for both L_{R-I} and DAR schemes are given in the same figure. Also shown in Fig 5.11 is another curve corresponding to the minimum blocking probability for the same network. To calculate this curve consider a strategy that first offered the call to path 1-2-3 and

then if no circuit is available chose path 1-4-3. Therefore a call is lost only if there are no free circuits on either paths. The probability that a call is lost is calculated using Erlang's formula:

$$Z_{min} = E(A_{13}, c_{123} + c_{143})$$

Z_{min} is also a lower bound on the performance of the network.

The results in Fig 5.11 indicate that for $A_{13} < 80$, L_{R-I} gives virtually optimum performance. For higher values of traffic, both L_{R-I} and DAR perform very close to the optimum. Similar results were obtained for 60/20 and 40/40 ratio of the effective link capacities of the two alternate paths.

5.3.5 - EXP 5:

For network 1:

$c_{12} = 50$ $\quad$ $c_{14} = 30$

$c_{23} = 50$ $\quad$ $c_{34} = 30$

$A_{13} = 60$ Erlangs

In this experiment we consider L_{R-P} and L_{R-I} and study their convergence behavior. Simulation results in Fig 5.12 for L_{R-P} with $(a = b = .1)$ illustrate the variation of P(1,2,3) versus call arrivals. Also shown is a straight line corresponding to the theoretical prediction for P(1,2,3). As can be seen, P(1,2,3) shows a significant variance associated with the large stepsizes chosen to update the probabilities. In another set of simulations, we reduced the values of stepsizes by reducing the reward

and penalty parameters to ($a = b = .01$). The results given in Fig 5.13, show that after about 400 calls, P(1,2,3) converges to the theoretical level and fluctuates around that point with considerably less variance compared to the curve in Fig 5.12 for ($a = b = .1$).

Similar simulations were performed for L_{R-I}. Fig 5.14 for ($a = .1$) illustrates that after 320 calls, P(1,2,3) converges to 1. As was discussed in section 5.2, for ($a = .1$), L_{R-I} converges to the selection of the path with a higher link capacity, path (1,2,3). Fig 5.15 shows results for L_{R-I} with ($a = .01$). It can be seen that after about 560 calls, P(1,2,3) converges to the theoretical value and exhibits a relatively small variance around that point.

5.3.6 - EXP 6:

For network 1:

$c_{12} = 4$ $c_{14} = 4$

$c_{23} = 4$ $c_{34} = 4$

$\lambda_{13}=6$

This experiment compares the performance of L_{R-I} and DAR on a call by call basis. The network has been simulated with the above specified data and the steady state blocking probabilities were recorded as 17.1% and 22.5% for L_{R-I} and DAR respectively. In the simulation program, number of free circuits on paths 1-2-3 and 1-4-3 were recorded during an interval of time. Two data samples were recorded for

the two algorithms. First we consider the DAR sample:

DAR			
no	FC(1,2,3)	FC(1,4,3)	Routed
1	1	4	V2
2	0	4	R
3	0	4	R
4	2	4	V4
5	4	3	V4
6	4	2	V4
7	4	1	V4
8	4	0	R
9	4	1	V4
10	4	0	R
11	4	0	R
12	4	1	V4
13	4	1	V4
14	4	0	R
15	4	0	V2

Where:

FC = free circuit

V2 = call routed via node 2

V4 = call routed via node 4

R = call rejected

In the above table for DAR, line one shows one free circuit on path 1-2-3 and four on path 1-4-3; a call arrives and is routed via node 2. At line two when a call arrives, DAR attempts path 1-2-3 (because last time this path was successful); there is no free circuit and the call is rejected. At this point DAR selects a random path

for the routing of the next call. Line three shows that DAR selects 1-2-3 randomly and the call is rejected again. Line four shows that two circuits have been released on 1-2-3 and four free circuits exists on 1-4-3. This time DAR selects 1-4-3 randomly and the call is successful. Lines 5 to 7 show that DAR keeps selecting 1-4-3 until line 8 when the call is rejected. Line 9 shows that DAR selects 1-4-3 randomly; a circuit has already been freed on this path and the call is successful. Lines 10 to 14 show that DAR loses three more calls on 1-4-3 while free circuits exists on 1-2-3. Finally at line 15 DAR switches to 1-2-3. The second table shows results for L_{R-I}.

L_{RI}			
no	FC(1,2,3)	FC(1,4,3)	Routed
1	1	4	V4
2	2	3	V4
3	2	2	V2
4	1	2	V2
5	1	2	V2
6	0	3	R
7	1	3	V4
8	1	2	V2
9	0	2	R
10	1	2	V2
11	4	4	V2
12	3	4	V4
13	4	3	V2
14	3	3	V4
15	3	4	V2

From the L_{R-I} sample it can be seen that there is a better distribution of traffic at each instant compared to DAR. L_{R-I} makes less mistakes and therefore gives a

better result due to this smoother spread of traffic.

5.4 - Simulation II : Five node network

5.4.1 - Exp 7:

Simulations were carried out for network 2, shown with its link capacities in Fig 5.2. The values of the base traffic matrices are given in tables 5.1 and 5.2. For each routing scheme the blocking probabilities were plotted against the scale factor OVL. The graphs are shown in figures 5.16-17 and the numerical results are listed below. The learning parameter is $a = .01$ for L_{R-I}.

Traffic matrix 1

OVL	L_{RI}	DAR
10%	0.80	1.09
20%	2.29	2.62
30%	4.70	5.29

Traffic matrix 2

OVL	L_{RI}	DAR
10%	1.97	2.98
20%	4.04	4.75
30%	6.50	7.35

The results indicate the superiority of L_{R-I} over DAR for the two traffic matrices and different overload factors. Further simulations have been performed to compare fixed routing with dynamic routing and also to study the effect of trunk reservation on the performance of the network. The table below shows the steady state blocking

probabilities for FR and dynamic routing for 10% overload and traffic matrices 1-2.

Traffic matrix 1

L_{R-I}	DAR	FR
.80	1.09	6.49

Traffic matrix 2

L_{R-I}	DAR	FR
1.97	2.98	26.62

Clearly different traffic patterns can amplify the differential between FR and dynamic routing. The effect of TRP on blocking probabilities can be seen in table below for both DAR and L_{R-I} for 10% overload and traffic matrix 1.

TRP	0	1	5	7	10	∞
DAR	1.09	1.23	2.96	4.45	5.83	6.49
L_{R-I}	.80	1.36	3.23	4.23	5.71	6.49

For the particular network considered any attempt to introduce TRP adversely affects the network performance. Clearly by increasing TRP more traffic will be routed directly and the differential between FR and dynamic routing will be reduced. However any attempt to introduce TRP for this class of networks increases blocking probability.

5.4.2 - EXP 8:

This experiment compares L_{R-I} and DAR under failure conditions. Consider the network of Fig 5.3 where links 1-3 and 3-5 have been failed. A simple traffic pattern

was considered:

A_{13}=10 Erlangs , A_{35}=10 Erlangs

The network has been simulated for the two routing schemes and different overload values. The steady state blocking probabilities are listed below and the curves are shown in Fig 5.18.

OVL	%0	%10	%20	%30
L_{RI}	1.9	3.0	4.1	5.5
DAR	9.3	11.0	12.6	14.0

The above results show a significant superiority of L_{R-I} over DAR. L_{R-I} performs better for two reasons, first it reduces the probability of selecting the failed links to zero i.e it learns not to attempt them. DAR does not detect the failure and at the time of switching from one path to another it attempts the failed link and loses some calls. Second the traffic must be split equally between two possible paths which exists for each traffic source, e.g paths 1-2-3 and 1-4-3 for the node pair 1-3. Under these conditions L_{R-I} loses less calls than DAR when switching from one path to another, as was explained in simulation I.

5.5 - Summary

The effect of the learning parameter on the performance of a simple four node network has been investigated. As the curves in Figs 5.4-5.6 indicate the algorithm can be tuned for an optimal performance. Simulation results for network 1 show

that as the ratio of the effective capacity of the upper to lower path increases, the advantages of the L_{R-I} scheme are reduced. In the case of DAR a high ratio of link capacities on the two alternative paths is a favorable condition. Under these circumstances less switching will be required between the alternative paths for the DAR algorithm. It should be recalled that each time the DAR algorithm switches, a call is lost. Clearly as the ratio of the capacity of the two alternative paths approaches unity, then increasing switching will be required and therefore more calls will be lost.

Results on five node networks deliberately designed to force dynamic routing illustrated an improved performance with L_{R-I}. The failure condition experiment showed a significant superiority of L_{R-I} over DAR for the case studied. Finally it was shown that for a traffic matrix not designed for FR, FR resulted in higher blocking probabilities than dynamic routing and any attempt to introduce TRP adversely affected the network performance.

In the next chapter we study the behaviour of different dynamic routing algorithms on more realistic larger networks. First the important problem of instability is considered. Then dynamic routing strategies are compared on two ten node networks from BT and Bell Labs.

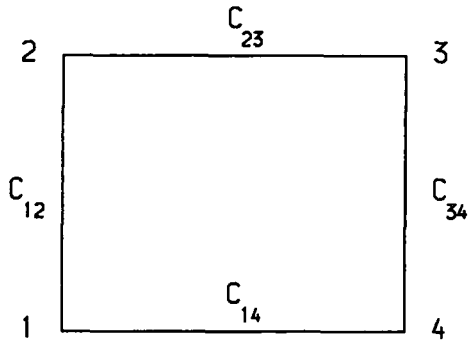


Fig 5.1 Network 1: Four node

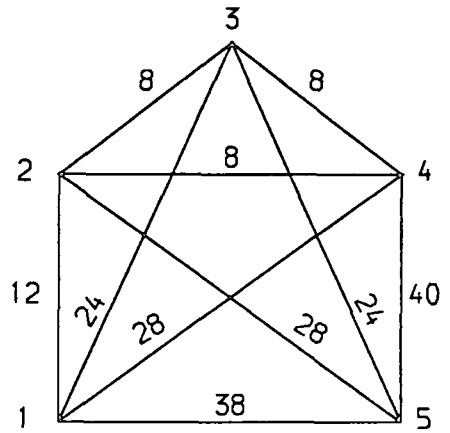


Fig 5.2 Network 2: Five node

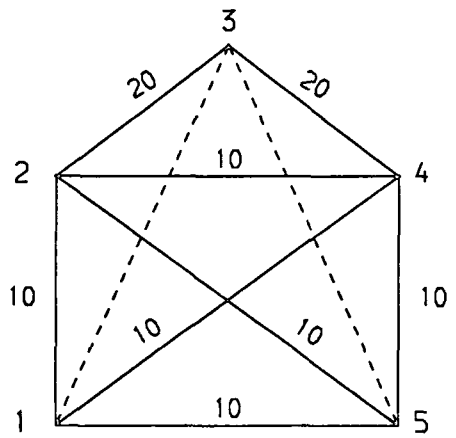


Fig 5.3 Network 3: Five node

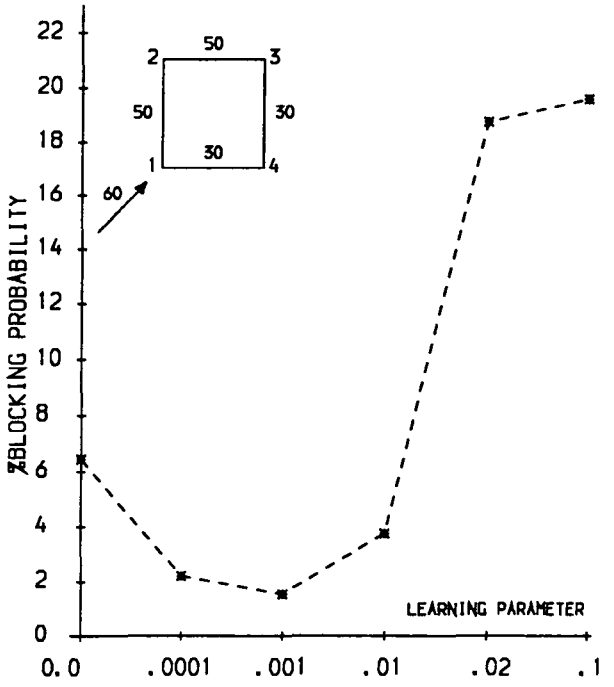


Fig 5.4 Effect of the Learning parameter on the performance of the LRI scheme.

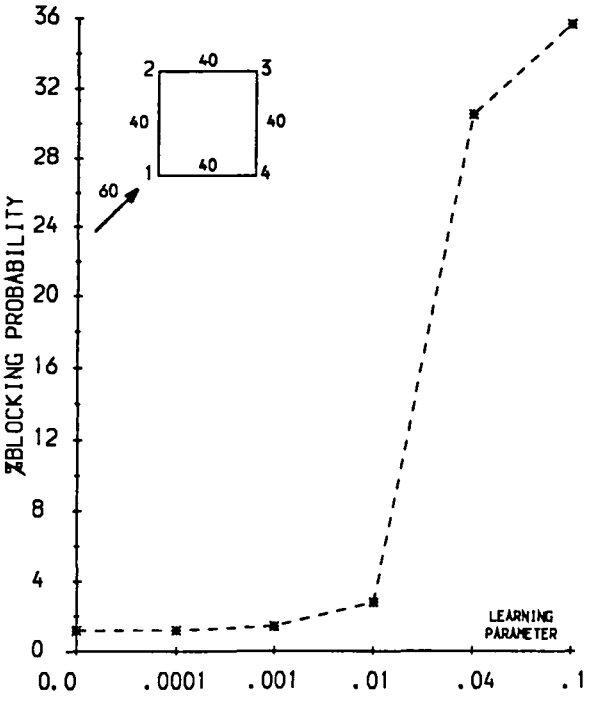


Fig 5.5 Effect of the Learning parameter on the performance of the LRI scheme.

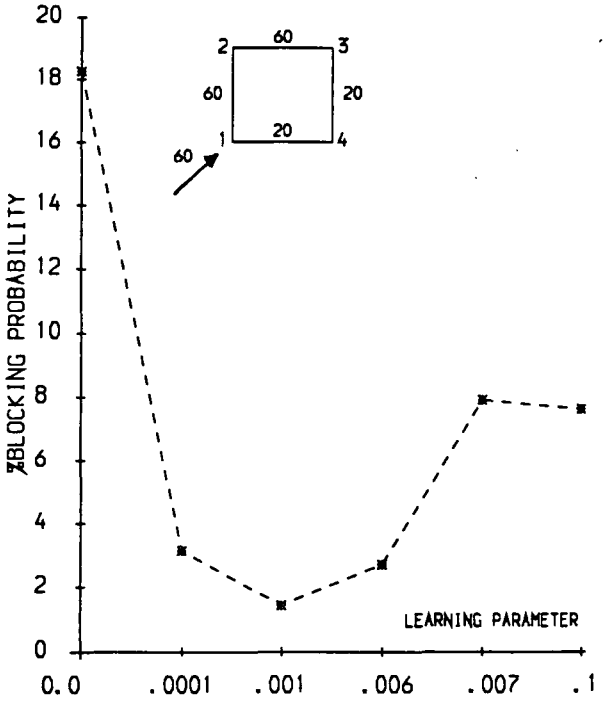


Fig 5.6 Effect of the Learning parameter on the performance of the LRI scheme.

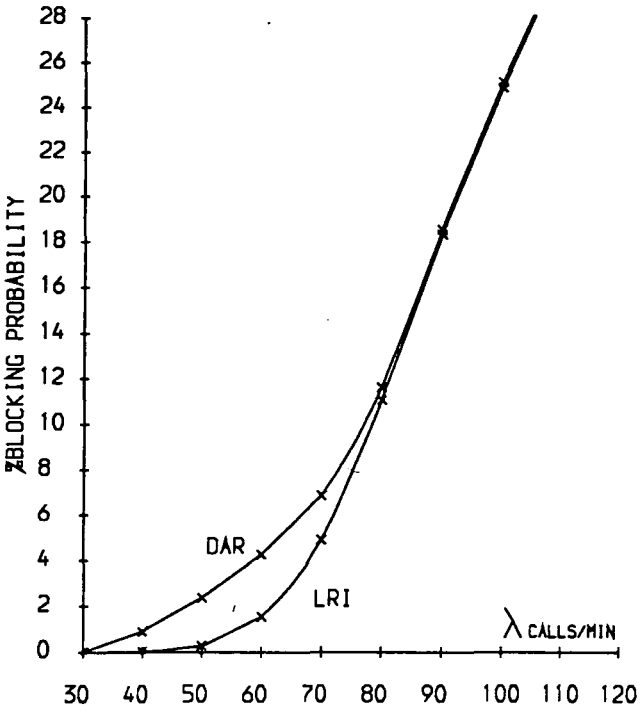


FIG 5.7 Comparison between LRI and DAR.
Network 1, link capacity ratio 50/30.

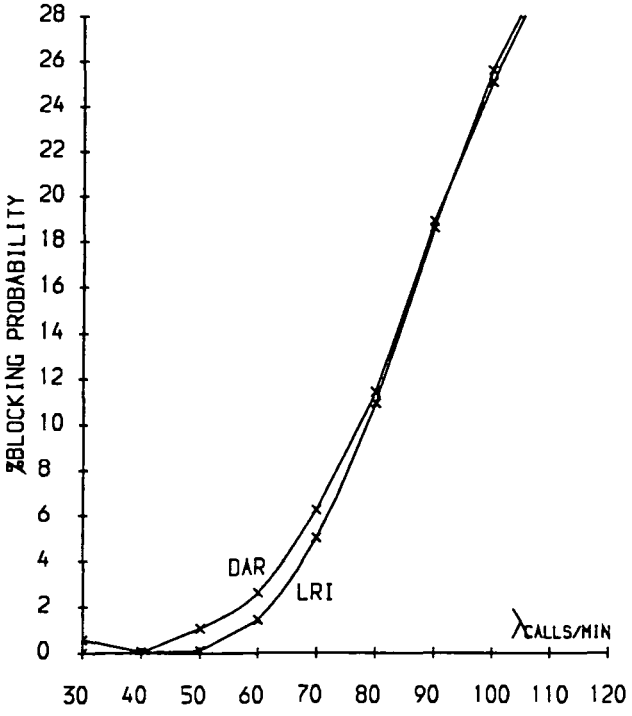


FIG 5.8 Comparison between LRI and DAR.
Network 1, link capacity ratio 60/20.

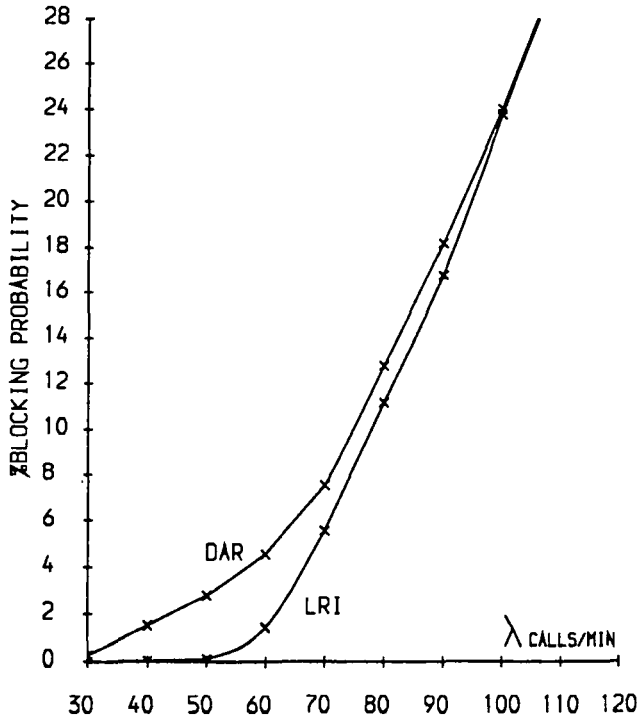


FIG 5.9 Comparison between LRI and DAR.
Network 1, link capacity ratio=40/40.

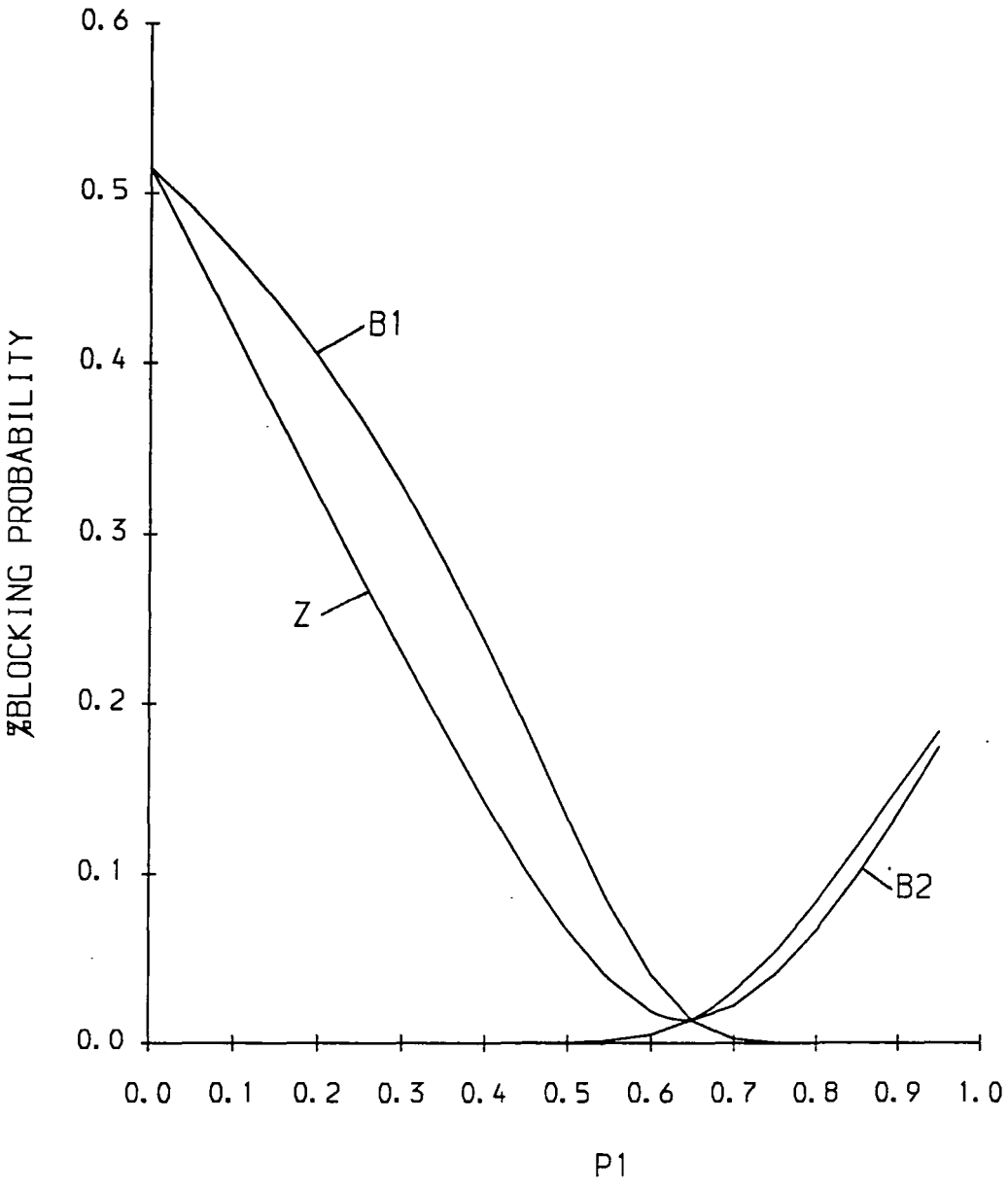


Fig 5.10 Calculation of the optimum probabilistic split.

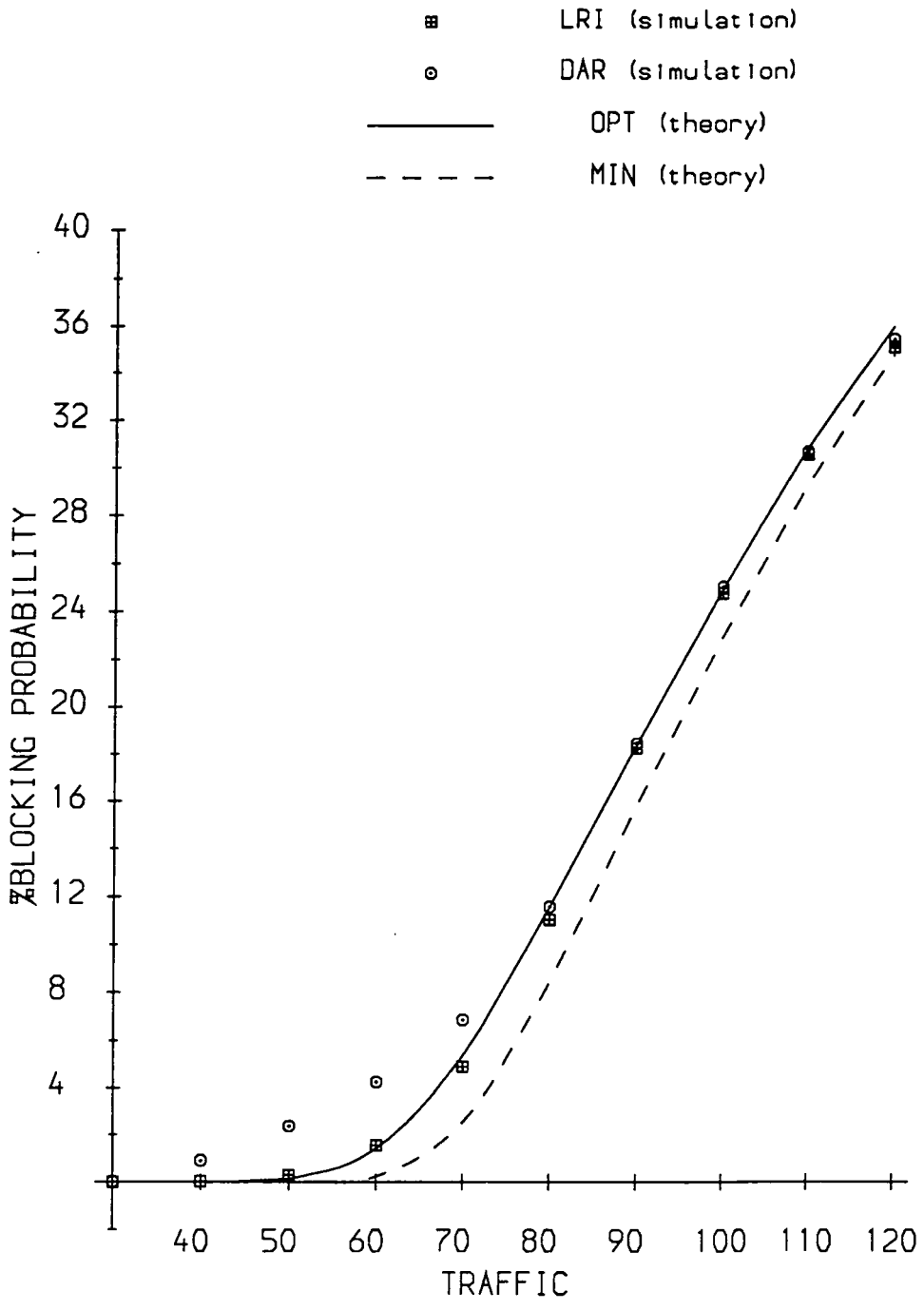


FIG 5.11 Comparison of LRI and DAR to the optimum and minimum performance, 50/30 ratio.

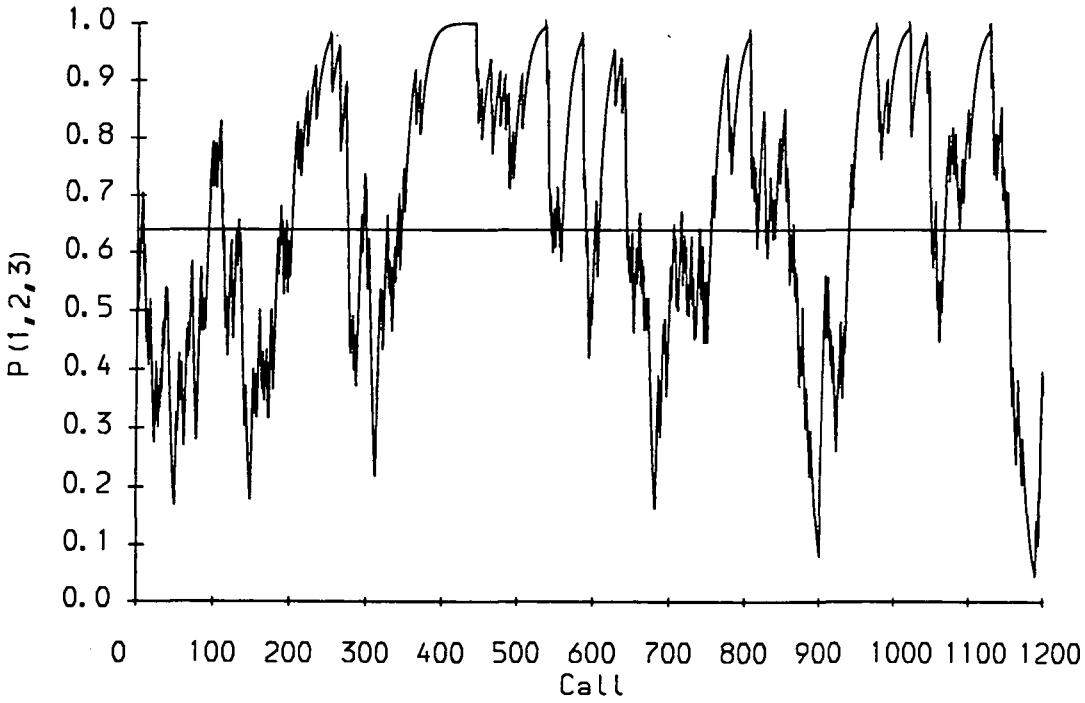


Fig 5.12 Evolution of P(1,2,3). LRP with a=b=.1.

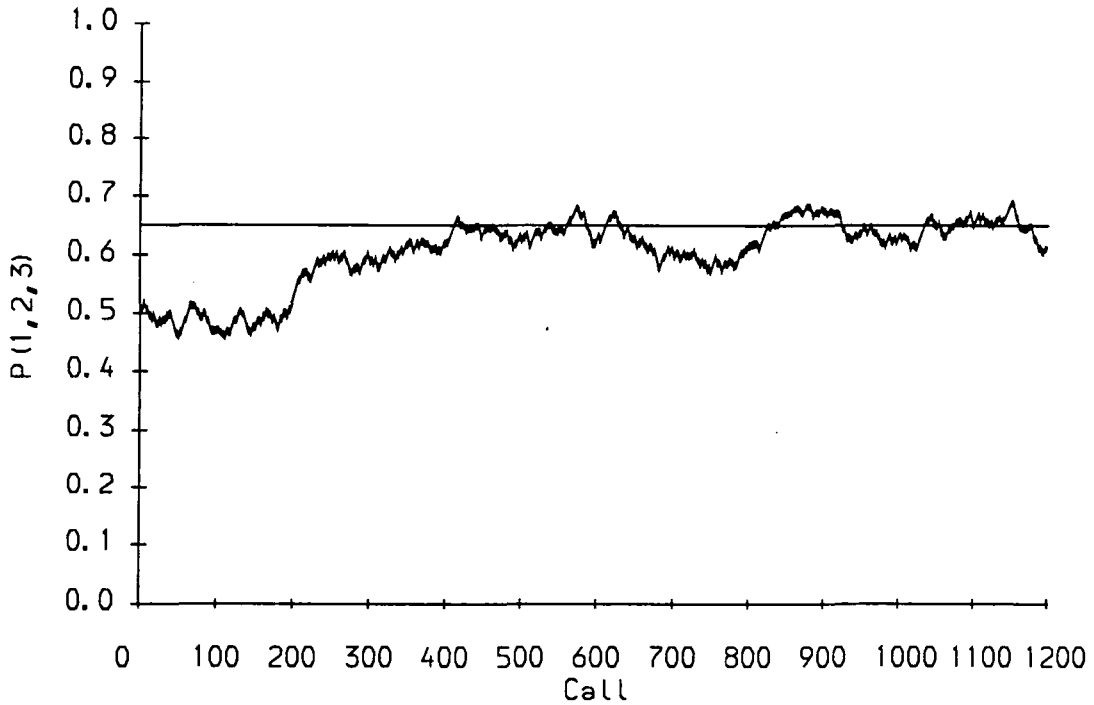


Fig 5.13 Evolution of P(1,2,3). LRP with a=b=.01.

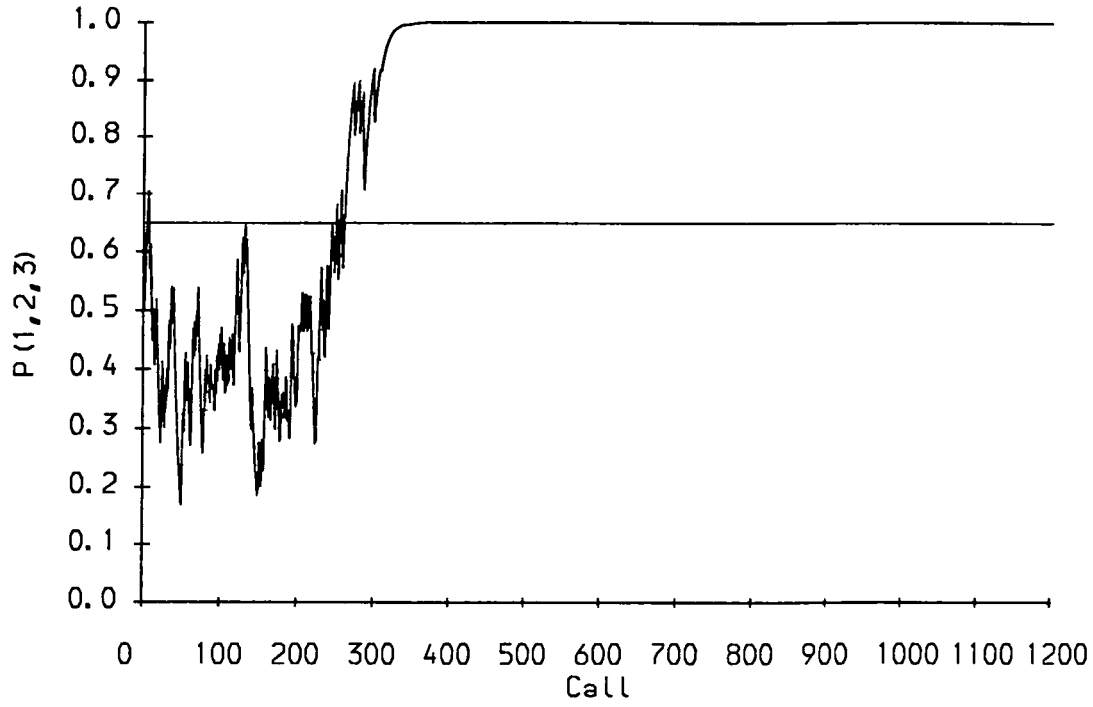


Fig 5.14 Evolution of P(1,2,3). LRI with a=.1.

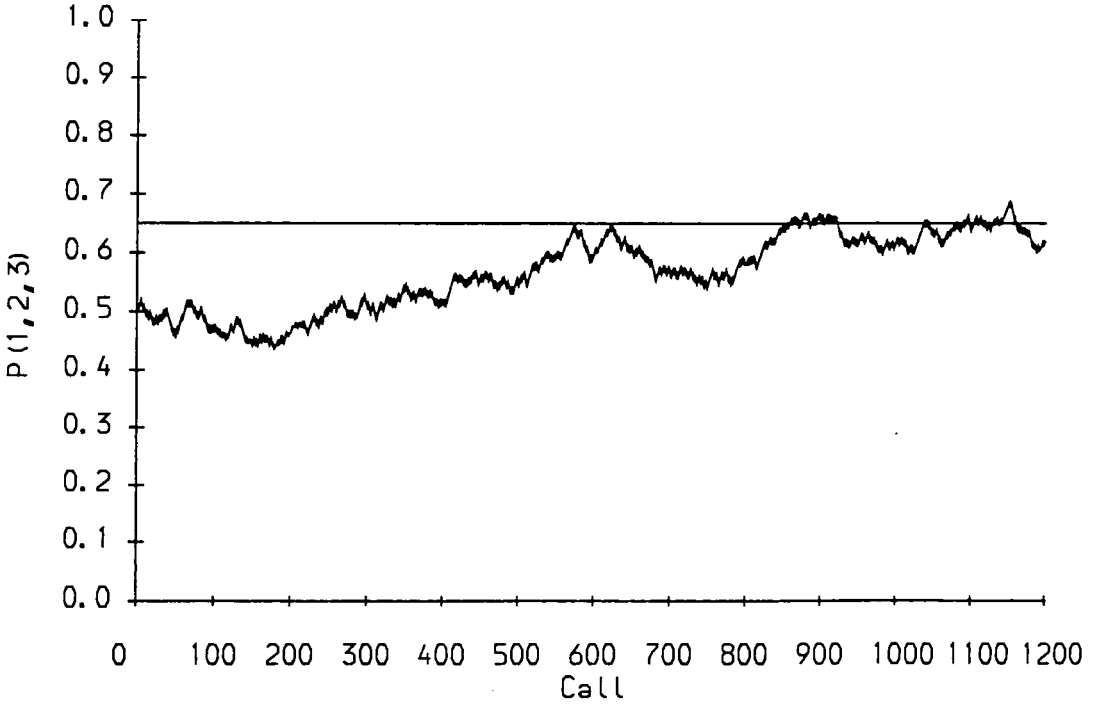


Fig 5.15 Evolution of P(1,2,3). LRI with a=.01.

$$\lambda_{ij} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0.0 & 15.0 & 15.0 & 15.0 & 30.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 15.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 15.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 15.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \end{array}$$

Table 5.1. Traffic matrix 1.

$$\lambda_{ij} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0.0 & 15.7 & 7.7 & 12.5 & 2.5 \\ 0.0 & 0.0 & 18.0 & 7.3 & 1.9 \\ 0.0 & 0.0 & 0.0 & 10.7 & .9 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.2 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \end{array}$$

Table 5.2. Traffic matrix 2.

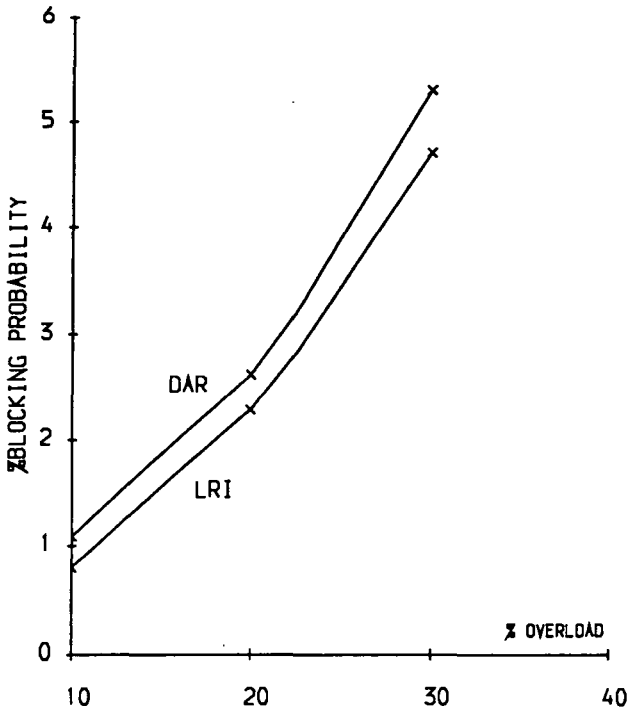


Fig 5.16 Comparison between LRI and DAR. Network2, traffic 1.

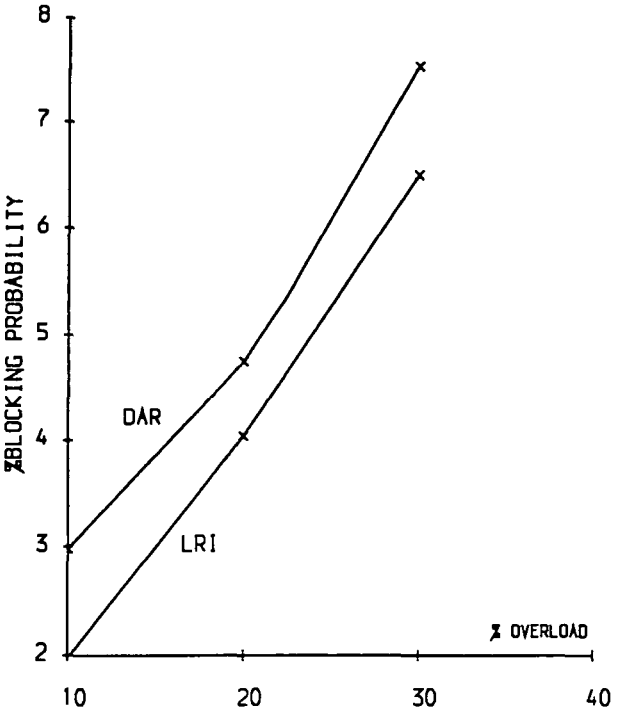


Fig 5.17 Comparison between LRI and DAR. Network 2, traffic 2

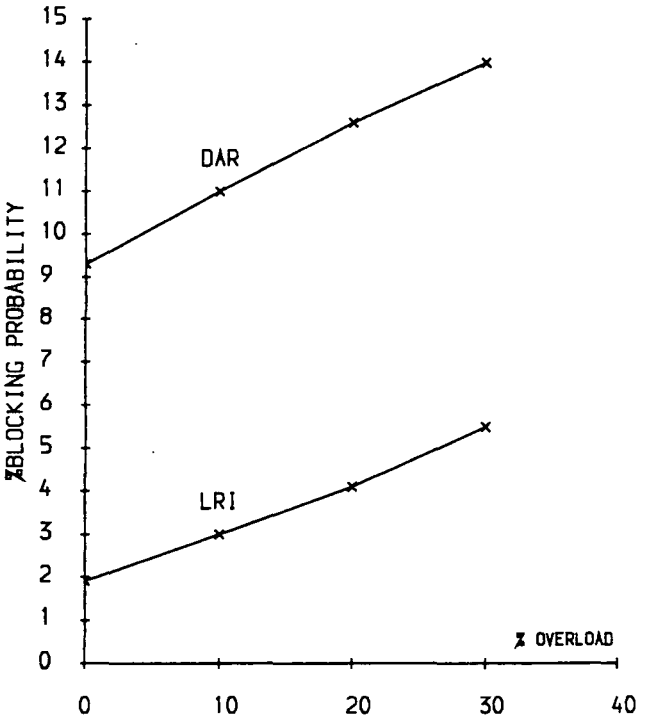


Fig 5.18 Comparison of LRI and DAR under failure conditions.

Chapter Six

Instability and larger networks

6.1 - Introduction

This chapter consists of two parts. In the first part we study the problem of instability when Automatic Alternative Routing (AAR) is used in a symmetric network using both analytical and simulation models. We then use our models to consider whether L_{R-P}, L_{R-I} and DAR routing strategies exhibit such unstable behaviour. No instabilities are found when these routing schemes are used. However the network shows a consistent drop in carried load at overloads. It will also be shown that using trunk reservation for first-routed traffic prevents instability and provides a high level of network carried load during overloads.

In the second part of this chapter we perform a series of experiments on two 10 node networks with general topologies and traffics from BT and Bell labs. The aim is to study the performance of the LA and compare it to different routing algorithms on more realistic larger networks.

6.2 - Instability

The problem of instability in non-hierarchical networks was first pointed out by Nakagome and Mori [30]. They considered small symmetrical network models and found hysteresis behaviour of networks through quantitative analysis. That is, once congestion occurs, it does not disappear immediately even if the load is decreased subsequently. Later Krupp carried out work on small symmetrical networks using both analytical and simulation methods. His work demonstrated the multiple equilibrium states and nonoptimal traffic handling under overload conditions [26]. Krupp introduced a simple control mechanism to prevent instability in nonhierarchical networks. This control strategy uses trunk reservation for direct or first-routed calls. In 1984, Akinpulu extended the mathematical model developed by Krupp to networks of more general type and has found both through analysis and through simulation models that the instabilities due to alternate routing do persist [27]. Studies of network models representing subsets of a fully engineered network do not show the instabilities. They do indicate a drop in carried load under overload conditions. In a recent paper Ackerley reports on hysteresis-type behaviour where the system spontaneously flips between high and low congestion levels even though the mean offered traffic is constant [31].

6.2.1 - Network model and statistical assumptions

The network model is assumed to be fully connected, symmetric and of nonhier-

archical type. All the links have the same number of channels and are capable of transmission in both directions. Assume that there are N nodes in the network. The direct path from source to destination is just one link long. Alternative paths are two links long and there are always $M = N - 2$ such alternate paths available. The test network shown in Fig 6.1, is a 10 node fully interconnected network with 45 full-duplex links each having a capacity of 50 circuits. Each of the 45 external traffic streams has a direct route plus eight two link overflow paths (the maximum possible). This network has been considered in [31].

Call arrival process entering the network is assumed to be Poisson with rate λ and the call holding time is exponentially distributed with mean $1/\mu$, for an Erlang rate $A = \lambda/\mu$. The call connection and disconnection times are assumed to be very short compared to the call holding time and hence are assumed to be negligible. Each link consists of c channels. If all c channels are carrying calls, the link is said to be busy. If a path consists of two links and at least one of them is busy, then the path is busy.

6.2.2 - Automatic Alternative Routing (AAR)

Each call is allowed to attempt the direct path first. If the path is busy, the call will attempt up to M more alternate paths in a prescribed order. If all M alternate paths are busy then the call is blocked and is considered as lost. Automatic re-routing, which is often refered to as 'crankback', is employed. This means that if the

second link of a two link overflow path is blocked, a call may crankback to the origin node and try the next overflow path. For symmetrical routing [32] the tandem nodes must be selected in a specific order to give a uniform pattern of overflow traffic. An example of symmetrical routing in a five node network is shown in Fig 6.2. From this figure it can be seen that each link appears:

- Once as a link in a direct route
- Once as each link in a first, second and third choice overflow path.

Such routing tables are not always possible to construct. It depends on the number of nodes in the network [32]. For example symmetrical routing is not possible for a 10 node network, so for the test network overflow paths are chosen cyclically and the routing from node i to node j is the same as from node j to node i (Fig 6.3).

6.2.3 - A mathematical model for AAR

For the network model explained in section 6.2.1 let:

A be the external offered load to a link

ν be the total offered load to a link

c be the link capacity

When dynamic routing is used the total load offered to a link consists of the external offered load plus alternate-routed calls overflowing from other direct paths. We assume that the external offered and overflow traffics are independent Poisson streams. Therefore given offered load ν to a link and the number of channels c of the

link, the blocking probability of the link is given by the Erlang B formula:

$$B = E(\nu, c) = \frac{\nu^c}{c!} \Big/ \sum_{i=0}^{c} \frac{\nu^i}{i!} \qquad (6-1)$$

We assume that link blocking probabilities are independent and each equal to B. The blocking probability of a two link path is given by:

$$L = 1 - (1 - B)(1 - B) \qquad (6-2)$$

The call blocking probability Z is the probability that a call entering the network finds the first choice direct path and all allowed alternate paths busy. For our fully connected symmetric network with N nodes, there are M=$N - 2$ two link alternate paths available. Therefore:

$$Z = B[1 - (1 - B)^2]^M = BL^M \qquad (6-3)$$

Given Z the carried portion of the externally offered load can be calculated:

$$C = A(1 - Z) \qquad (6-4)$$

The total load K carried on a link is the nonblocked portion of the total load ν offered to a link:

$$K = \nu(1 - B) \qquad (6-5)$$

We will now show that K can also be written as:

$$K = A(1 - B) + 2AB(1 - L^M) \qquad (6-6)$$

The first term of the above equation is the nonblocked portion of the external offered load A and the second term is the overflow traffic carried on each link. Consider a

link (a,b), the external offered load A to this link, overflows with a probability of B. The probability that this traffic will be blocked on a two link alternative path is L and on M two link paths is L^M. The probability that it will not be blocked and will be accepted as a call is $1 - L^M$. Therefore $AB(1 - L^M)/M$ is carried per each of the M alternate paths (a,k,b), due to overflow calls from link (a,b). Consider a link (i,j), the carried load K_{ij} on the link can be written as:

$$K_{ij} = A_{ij}(1 - B_{ij}) + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} \frac{A_{ik}B_{ik}(1 - L^M)}{M} + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} \frac{A_{jk}B_{jk}(1 - L^M)}{M}$$

The above equation can be simplified for our symmetric network model:

$$K = A(1 - B) + 2AB(1 - L^M) \qquad (6-6)$$

Equating the two expressions for K we find a relationship between the total offered load ν and the external offered load A:

$$\nu = \frac{A(1 + B - 2Z)}{(1 - B)} \qquad (6-7)$$

Given A, c and N we can implement the repeated substitution method to solve the above equations. Assume an initial value $B0$ for B, find Z and ν. Given ν and c implement the Erlang B formula to find the new value for B. Repeat the process until B converges. The carried proportion of the externally offered load can then be calculated. The iterative loop is given below:

$$B = B0$$

Iterative loop:

$$Z = B[1 - (1 - B)^2]^M \qquad (6-3)$$

$$\nu = \frac{A(1 + B - 2Z)}{(1 - B)} \qquad (6-7)$$

$$B = \frac{\nu^c}{c!} \Big/ \sum_{i=0}^{c} \frac{\nu^i}{i!} \qquad (6-1)$$

repeat until convergence.

The carried load C, per each link can then be calculated:

$$C = A(1 - Z) \qquad (6-4)$$

6.2.4 - Experiment 1

This experiment discusses the theoretical and simulation results obtained for AAR. A set of analytic curves were calculated for the test network. Figures 6.4 and 6.5 show plots of carried load for various numbers of alternate routes M. The instability is evident in the multiple values of carried load for $M = 8$ alternate routes. (Solutions corresponding to low network carried load were obtained by starting with high trunk-group blocking estimates, while solutions corresponding to high network carried load were obtained by starting with low trunk-group blocking estimates.) The case of one alternate route in Fig 6.5 shows a drop in carried load as the offered load is increased beyond 42 Erlangs/trunk. The $M = 0$ curve, that for direct routing only, does not exhibit instability and provides a constant increase in carried load when offered load is increased. It does provide poorer performance at lower loads (Fig 6.5).

Fig 6.6 plots the proportion of directly routed calls, PDC per point to point offered load for the case of $M = 8$. This is calculated as:

$$PDC = \frac{A(1-B)}{A(1-Z)} = \frac{(1-B)}{(1-Z)} \qquad (6-8)$$

The multiple values of the proportion of directly routed calls for the range $35 < A < 40$ demonstrate instability in the network. Fig 6.7 shows the same plot but for $M = 1$ alternate path allowed.

Simulation results given in Fig 6.8 for $A = 38$ show the proportion of directly routed calls in progress at intervals of one holding time throughout an interval of a simulation. The network appears to flip between two quasi-stable states, one a low congestion state in which almost all calls use their shorter first choice path and the blocking probability is low and the other a high congestion state in which a large proportion of calls use longer alternate paths and the blocking probability is high. When the network is in either of these states it tends to stay there and when it is between these states it rapidly moves to one or other of the states. The existence of two recognizable states in the simulation results for $A = 38$ in Fig 6.8 is analogous to the double-valued results found using the analytic curve in Fig 6.6.

The mechanisms that cause the system to flip between low and high congestion states can be explained as follows: Assume that the system is in the low congestion state. The number of call arrivals and departures during a time interval varies stochastically even though the call arrival rate is constant. A large number of call arrivals and small number of call departures during the response time of the network

will cause the link congestion to increase. This in turn will cause more calls to use overflow paths which consist of two links, and thus link congestion increases further. In this way the network flips to to the high congestion state. On the other hand assume that the network is in the high congestion state. A large number of call departures (especially two link calls) and a small number of call arrivals will cause the link congestion to decrease. This in turn causes fewer calls to use overflow paths and so the link congestion decreases further. In this way the network flips to a low congestion state.

Simulation results in Fig 6.9 provide the proportion of directly routed calls for $A = 42$. It can be seen that after a few holding times the network goes into the high congestion state and remains there throughout the rest of the simulation. For this value of A, a large proportion of calls use alternate paths and carried load drops dramatically as was predicted by analytical results given in Fig 6.4.

6.2.5 - Experiment 2

Two examples are given in this experiment. Fig 6.10 shows the result of a simulation of the network with an initial load of 38 Erlangs. The load was increased to 39 Erlangs after 80 holding times. As Fig 6.10 illustrates a dramatic change in proportion of directly routed calls occurs after the offered load is increased. With the initial load the blocking probability is low, and few calls are alternately-routed. After the load changes, the blocking probability goes up and number of alternately-routed

calls increases.

Another example is given in Fig 6.11, which shows simulation results for the same network with an initial point to point load of 42 Erlangs. For this load the network is highly congested. After 50 holding times, the load is dropped to 38 erlangs. It can be seen that when the load is dropped congestion persists for another 10 holding times.

6.2.6 - A mathematical model for AAR with trunk reservations

Let B_1 and B_2 be the blocking probabilities to fresh and overflow traffic on a link respectively. Let $Q_2 = 1 - B_2$ be the link availability on a link. B_1 and Q_2 were calculated in chapter 4:

$$B_1 = p_c = \frac{A^{c-m}\nu^m}{c!}p_0 \tag{4.10}$$

$$Q_2 = \sum_{l=0}^{m-1} \frac{\nu^l}{l!}p_0 \tag{4.11}$$

Where $c - m$ is the trunk reservation parameter.

The end to end blocking probability can then be calculated as:

$$Z = B_1(1 - Q_2^2)^M \tag{6-9}$$

To find the offered load ν to a link we first calculate the carried load K on a link. As was done in Eq.(6-6), K can be written as the sum of first-routed carried load and alternately routed carried load.

$$K = A(1 - B_1) + 2AB_1[1 - (1 - Q_2^2)^M] \qquad (6-10)$$

The carried load on a link is the nonblocked portion of the total load offered to a link.

$$K = A(1 - B_1) + (\nu - A)Q_2 \qquad (6-11)$$

Equating 6-10 and 6-11:

$$K = A(1 - B_1) + 2A(B_1 - Z) = A(1 - B_1) + (\nu - A)Q_2$$

The offered load can be found from the above equation.

$$\nu = A + \frac{2A}{Q_2}(B_1 - Z) \qquad (6-12)$$

The carried load per link is calculated as before:

$$C = A(1 - Z) \qquad (6-4)$$

Given m, A, c and N we can implement the repeated substitution method to solve the above equations for B_1 and Q_2. The expressions for B_1 and Q_2 are simplified in Appendix 1. The iterative loop is given below:

Assume initial values for B_1 and Q_2:

$$B_1 = B_10 \qquad Q_2 = Q_20$$

The iterative loop:

$$Z = B_1(1 - Q_2^2)^M \qquad (6-9)$$

$$\nu = A + \frac{2A}{Q_2}(B_1 - Z) \qquad (6-12)$$

$$B_1 = p_c = \frac{A^{c-m}\nu^m}{c!}p_0 \qquad (4-10)$$

$$Q_2 = \sum_{j=0}^{m-1} \frac{\nu^j}{j!}p_0 \qquad (4-11)$$

repeat until convergence

$$C = A(1 - Z) \qquad (6-4)$$

6.2.7 - Experiment 3

This experiment discusses theoretical and simulation results for AAR with trunk reservation. Analytic curves in figures 6.12 and 6.13 for $M = 8$ and $M = 1$ alternate paths, compare carried load for various values of TRP. It is evident that reserving a small number of trunks produces a significant improvement in carried load. Figures 6.14 and 6.15 show the effect of trunk reservation on the proportion of directly routed calls for the cases of $M = 8$ and $M = 1$ alternate paths.

Simulation results in Fig 6.16 show the proportion of directly routed calls for $M = 8$ alternate paths and $A = 38$ Erlangs, similar to Fig 6.8 but with $TRP = 1$. As can be seen the quasi-stable behaviour disappears. Simulation results for $A = 42$

given in Fig 6.17, show a significant improvement in proportion of directly routed calls with TRP=1, compared to the case with no trunk reservation (Fig 6.9).

6.2.8 - Dynamic routing algorithms

In this subsection we show that for our symmetric network model, the mathematical model for different dynamic routing algorithms, L_{R-P}, L_{R-I}, DAR and RR is the same as the one calculated for AAR with $M = 1$. A discussion of theoretical and simulation results obtained for those algorithms will be given next.

When no trunk reservation is employed, for L_{R-P}, L_{R-I}, DAR and RR, the offered load to a link (i, j) is:

$$\nu_{ij} = A_{ij} + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{ik}B_{ik}P_j(i,k)Qjk + \sum_{\substack{k=1 \\ k\neq i \\ k\neq j}}^{N} A_{jk}B_{jk}P_i(j,k)Q_{ik} \qquad (4-3)$$

For our network model: $\nu_{ij} = \nu$, $A_{ij} = A$, $B_{ij} = B$, $Q_{ij} = Q$, $P_k(i,j) = \frac{1}{N-2}$.

Therefore:

$$\nu = A + (N-2) \times AB \times \frac{1}{N-2} \times Q + (N-2) \times AB \times \frac{1}{N-2} \times Q$$

$$\nu = A + 2ABQ \qquad (6-13)$$

The call blocking probability Z is the probability that a call entering the network finds the first choice direct path and a two link alternate path busy.

$$Z = B(1 - Q^2) \qquad (6-14)$$

The two above equations for ν and Z are the same as equations (6-7) and (6-3) obtained for AAR with M replaced by 1.

When trunk reservation is employed the offered load to a link can be obtained by simplifying equation (4.15) for the symmetric network model:

$$\nu = A + 2AB_1Q_2 \qquad (6-15)$$

The call blocking probability Z, is the probability that a call is blocked on the first choice direct path and on a two link alternate path:

$$Z = B_1(1 - Q_2^2) \qquad (6-16)$$

If we replace $M = 1$ in equations (6-9) and (6-12) for AAR then equations (6-16) and (6-15) follow. Therefore the theoretical calculations given in figures 6.5, 6.7, 6.13 and 6.15 for AAR with $M = 1$, apply to different dynamic routing schemes, L_{R-P}, L_{R-I}, DAR and RR.

6.2.9 - Experiment 4

As was explained in the previous subsection, the curve given in Fig 6.5 for AAR with M=1, applies to L_{R-P}, L_{R-I} and DAR. From this curve, it can be seen that these routing algorithms do not exhibit instability. Fig 6.5 compares dynamic routing schemes with fixed routing ($M = 0$). It can be seen that fixed routing provides poorer performance at lower loads and better performance at higher loads. Fig 6.5 also shows a drop in carried load as the offered load increases beyond a certain point. The reason that the carried load decreases at higher offered loads is that an alternately routed call uses two trunks (one on each link in the alternate path), rather than one trunk

required in the direct path. For small carried load this improves the performance compared to the case with no alternate routing (Fig 6.5). For higher loads alternately routed calls use double the resources, blocking directly routed calls in both links used and causing them to overflow more often, which in turn blocks direct calls even more and results in a drop in carried load. Fig 6.13 shows the theoretical results for carried load when trunk reservation is employed. It can be seen that as TRP increases, the carried load at higher offered loads is increased.

A series of simulations have been performed for L_{R-P}, DAR and L_{R-I} for two values of offered load $A = 38$ and $A = 42$ and $TRP = 0$. The plots are given in Figs 6.18-6.23. They show no instability as was predicted from the theoretical results.

Tables 6.1 ,6.2 and 6.3 list simulation and theoretical results for different routing schemes and various values of A and TRP. Table 6.1 lists results for $A = 38$ and $TRP = 0$. As can be seen the blocking probabilities are lower for L_{R-P}, L_{R-I} and DAR. The theoretical results for AAR are double valued as was explained in section 6.2.4. The theoretical and simulation results do not agree for AAR, because simulation results are averaged over high and low congestion states. The analytic and simulation results show a close agreement for L_{R-P}, L_{R-I} and DAR. Table 6.2 for $A = 42$ and $TRP = 0$ shows a significant superiority for L_{R-P}, L_{R-I}, DAR over AAR. Table 6.3 illustrates the effect of TRP on the network. It can be seen that by reserving one trunk for first-routed traffic, the blocking probability for AAR drops from 21.43 to 4.14 percent. In all the three tables the theoretical results for AAR are slightly different from the simulation results because in theory we assumed symmetric

routing while in simulations symmetric routing is not possible for a 10 node network as was explained in section 6.2.2.

6.3 - Large networks

In chapter four we compared different dynamic routing strategies on a five node network which was designed for FR. It was shown that FR was superior to other routing schemes and different dynamic routing methods were as good as FR when a TRP=10 was employed. In chapter five the performance of Learning Automata was compared to DAR on four and five node networks which were designed to force dynamic routing. It was shown that Automata always outperformed DAR. In this section we perform a series of experiments on two ten node networks from BT and Bell Labs. The aim is to compare different routing schemes on more realistic larger networks.

6.3.1 - The BT network

The network is a 10 node fully inter-connected subset of the main BT network (Fig 6.1). Two traffic matrices are considered, one for the morning and one for the evening. The capacity and traffic matrices are given in appendix 2. Figs 6.24 and 6.25 for the morning and the evening traffic matrices show blocking probabilities versus TRP for different routing schemes. The reward and penalty parameters are (a=.01, b=.01) for L_{R-P} and (a=.01) for L_{R-I}. As can be seen L_{R-P}, L_{R-I} and DAR perform very

close to each other and result in significantly lower blocking probabilities than FR and RR. As TRP is increased the blocking probabilities for L_{R-P}, L_{R-I}, DAR and RR increase and approach the one for FR. Unlike the five node network studied in chapter four, the performance of this network deteriorates once TRP is introduced. As was mentioned in chapter four, when TRP is employed alternate routing is reduced and the dynamic routing algorithm's behaviour approaches FR. For the five node network studied in chapter four, FR performed better than dynamic routing algorithms so by introducing TRP, the algorithms approached FR and their performance improved. For the 10 node network considered in this section, FR results in significantly higher blocking probabilities than dynamic routing algorithms and therefore by introducing TRP, the algorithms approach FR and their performance deteriorates.

The results given in Figs 6.24 and 6.25 are for no overflow of traffic. The same experiment was repeated for different values of overload. Fig 6.26 for L_{R-P} and DAR with the morning traffic matrix shows the theoretical blocking probabilities versus TRP for a range of overload values. As can be seen for 0% and 5% overload curves the minimum blocking probabilities are at TRP=0. The minimum blocking probability for 10% overload case is at TRP=1 and at overload values higher than 10%, the best TRP is equal to 5. Therefore the optimum TRP for the network varies with overload. For small values of overload, the use of two-link paths improves the performance of the network and no trunks are needed to be reserved. At high overload values more calls attempt two-link alternate paths, each blocking two calls which could otherwise be routed along two one-link direct paths. Under these conditions trunk

reservation is needed to protect direct paths from being used as part of alternative paths. As overload goes up, the network needs more protection and a higher TRP is needed. Similar results were obtained for the evening traffic matrix, Fig 6.27.

Fig 6.28 for the morning traffic matrix with 30% overload, shows theoretical and simulation results for different dynamic routing algorithms. From this figure it can be seen that all dynamic routing algorithms outperform FR and the minimum blocking probability is obtained for TRP=5 with both analytic and simulation models. Note that the minimum blocking probability for L_{R-I} is at TRP=1.

In another experiment the network was simulated under abnormal traffic conditions. For this purpose the morning traffic matrix was varied in two stages. First, all streams were overloaded by 10%. Then half the links, selected at random, had their traffic levels increased by X% and the rest reduced by X%. This gives a simplified model of forcasting error, unforseen demand, etc which is useful to test a routing scheme's ability to cope with traffic and capacity mismatch. The average blocking probabilities for X=(0,25) are shown in Fig 6.29. As can be seen L_{R-I} and L_{R-P} are superior to DAR with the best performance for the L_{R-I} scheme. Under normal traffic conditions L_{R-P}, L_{R-I} and DAR schemes perform close to each other (Figs 6.24 and 6.25), but under abnormal traffic conditions L_{R-P} and L_{R-I} outperform DAR. This shows that L_{R-P} and L_{R-I} have better adaptation to changes in traffic conditions than DAR.

6.3.2 - The Bell network

In previous work [25] a series of simulations have been performed on a 10 node network obtained from Bell laboratories. The network is shown in Fig 6.30, the capacity and three traffic matrices are given in appendix 3. The three traffic matrices are implemented as follows: after 4000 calls arriving to the network the simulator switches from traffic matrix 1 to traffic matrix 2 , after another 3000 calls it switches to traffic matrix 3 and after another 3000 calls the simulator stops. The routing algorithms considered in [25] are: Fixed Rule Alternate Path, conventional L_{R-I} and three modifications to the conventional L_{R-I} scheme (short rule learning automata, limited path feedback, initial bias scheme). A brief description of these routing algorithms is given below:

In **Fixed Rule Alternate Path**, a set of routing matrices are given. Each matrix provides the routing information for a particular node with each row dictating the sequence of order for attempting to route a call for a certain destination. For example consider the routing table at node 5:

	1	2	3	4	5	6	7	8	9	10
1	1	3	6	7	2	9	8	10	–	–
2	1	6	3	2	7	9	8	10	–	–
3	3	1	6	7	2	9	8	10	–	–
4	1	3	6	7	2	9	8	10	–	–
5	–	–	–	–	–	–	–	–	–	–
6	6	9	8	9	10	3	1	2	–	–
7	7	6	8	9	10	3	1	2	–	–
8	8	9	10	6	7	3	1	2	–	–
9	9	8	10	6	7	1	3	2	–	–
10	10	9	8	6	7	3	1	2	–	–

The network has been designed to obtain optimal link capacities and routing tables for the given topology and traffic patterns. Therefore this scheme offers an optimal

behaviour with which the Learning Automata schemes can be compared with.

Conventional L_{R-I} scheme, this uses the same routing matrices as the fixed rule alternate routing. The difference is that L_{R-I} assigns probabilities to each node in the routing table. For example consider calls at node 5 whose destination is node 8. In this case 8 is picked with probability P_1, 9 with probability P_2 and so on. If there is no free circuits between node 5 and the first node chosen, then the node is dropped from the list of possible next nodes, the action probabilities are renormalized and a further selection is made. This continues until a line is found or the call is lost.

In the **Short Rule Learning Automata scheme**, the choice of next node has been reduced. In the reported experimental work, the choice was restricted to the top three nodes of the original optimized fixed rule. For example consider node 5:

	P_1	P_2	P_3
1	1	3	6
2	1	6	3
3	3	1	6
4	1	3	6
5	–	–	–
6	6	9	8
7	7	6	8
8	8	9	10
9	9	8	10
10	10	9	8

This scheme performed better than the conventional LA.

The **Limited path feedback scheme**, rewards calls which are completed using two or less links, punishes calls which are lost or completed using more than two links.

Finally the **Initial Bias Scheme** has been implemented to improve the performance of the routing scheme during the early phases of the simulation. The initial

probabilities of the limited path feedback scheme are set as follows:

$$P_1 = .5, P_2 = .25, P_3 = .175, \ldots$$

where $\sum_i P_i = 1$.

The average blocking probabilities obtained in [25] are listed below:

scheme	traffic 1	traffic 2	traffic 3
Fixed Rule	5.3	2.1	2.7
Conventional L_{R-I}	12.1	8.9	10.1
Short Rule L_{R-I}	16.1	8.2	4.7
Limited Path L_{R-I}	10.7	7.4	5.8
Initial Bias L_{R-I}	13.0	11.0	5.4

As can be seen the Fixed Rule performs significantly better than the other routing schemes and the limited path feedback L_{R-I} where two link alternatives are encouraged, is the best of the automata schemes.

In the learning automata schemes implemented in this thesis, the direct link path is always attempted first and alternatives are only two links long. The same network was simulated for L_{R-P} and L_{R-I} and the blocking probabilities show a significant improvement over the above results. The table below shows blocking probabilities for L_{R-P}, L_{R-I}, DAR, RR and FR. The limited path feedback L_{R-I} is also included for comparison.

table 6.4

scheme	traffic 1	traffic 2	traffic 3
FR	12.55	14.20	20.49
RR	9.33	10.04	15.03
DAR	8.1	5.0	8.8
$L_{R-I}^{(.1)}$	5.8	5.2	7.8
$L_{R-P}^{(.1,.1)}$	6.2	4.0	6.2
$L_{R-I}^{(.01)}$	9.4	7.3	9.4
$L_{R-P}^{(.01,.01)}$	8.9	7.2	8.4
Limited Path	10.7	7.4	5.8

As can be seen, $L_{R-P}^{(.1,.1)}$ and $L_{R-I}^{(.1)}$ are superior to other schemes and the highest blocking probabilities are obtained for FR.

In another set of experiments the simulator was run for a long time with the traffic matrices 1, 2 and 3 switched after 51000, 45000 and 38000 calls respectively. The blocking probabilities are shown in the below table:

table 6.5

scheme	traffic 1	traffic 2	traffic 3
DAR	6.54	4.72	7.56
$L_{R-I}^{(.1)}$	5.88	6.05	8.04
$L_{R-P}^{(.1,.1)}$	5.52	3.70	5.69
$L_{R-I}^{(.01)}$	5.62	3.52	4.92
$L_{R-P}^{(.01,.01)}$	5.78	3.70	5.71

Comparing tables 6.4 and 6.5 for short and long simulation times, a number of conclusions may be made:

1 - $L_{R-P}^{(.1,.1)}$ and DAR give lower blocking probabilities when averages are taken

over a longer simulation time. This suggests that the previous simulation time was not sufficient for the two algorithms to converge to steady state results. For example in table 6.4, For the traffic matrix 2, the averages are taken over 3000 calls arriving to the network. There are 45 O-D pairs in the 10 node test network and on average each O-D pair receives 3000/45=67 calls. This must be compared to 45000/45=1000 calls per O-D pair for traffic matrix 2 in table 6.5.

2 - For $L_{R-I}^{(.1)}$, the blocking probabilities are higher over a longer simulation time. As was explained in chapter five, when L_{R-I} with $a = .1$ is implemented the algorithm converges to selecting one action for each O-D pair and as a result loses its flexibility and can not fully utilize the spare capacity in the network. This happens after running the simulator for a long time to allow the algorithm to converge to its steady state probabilities. The action probabilities for the 45 O-D pairs were recorded at the end of the simulations. At the end of the short simulation time none of the probabilities were equal to 1 (L_{R-I} did not select any particular action with a probability of one) while at the end of the long simulation time almost all O-D pairs converged to the selection of one out of 8 possible alternate paths.

3 - $L_{R-P}^{(.01,.01)}$ and $L_{R-I}^{(.01)}$ perform significantly better over a longer time. The reason is that when a is small, more time is needed for the algorithms to converge to steady state results. In table 6.4 the algorithms have not converged to steady state action probabilities and as a result the blocking probabilities are higher than the ones in table 6.5 where the simulation was run for a long time.

4 - Comparing tables 6.4 and 6.5, it can be seen that $L_{R-P}^{(.1,.1)}$ has a faster conver-

gence than $L_{R-P}^{(.01,.01)}$ and $L_{R-I}^{(.01)}$ as the $L_{R-P}^{(.1,.1)}$ results are not significantly different in the two tables. $L_{R-P}^{(.1,.1)}$ is the best routing algorithm in this case because it converges fast and also results in a low blocking probability compared to other routing schemes. $L_{R-I}^{(.01)}$ performs slightly better than $L_{R-P}^{(.1,.1)}$ in table 6.5 for long simulations but over the short simulation time it is significantly worse than $L_{R-P}^{(.1,.1)}$.

6.4 - Summary

In the first part of this chapter, a mathematical model and its application was explained for AAR in symmetric, uniformly loaded nonhierarchical networks. Experiments using the mathematical model, showed the existence of network instabilities for a range of overload. Simulation results showed that the network can experience quasi-stable behaviour, flipping between low and high congestion states which may each last for considerable periods of time. This behaviour is analogous to the double-valued results found using the mathematical model. It was shown through analysis that carried load drops dramatically at overload.

Trunk reservation for first-routed traffic was applied to the test network. This control strategy improves the performance of the network at overloads. By reducing the amount of alternate routing in the network under overloads, trunk reservation permits 1-link calls to use the trunks more efficiently. This reduces the quasi-stable behaviour and results in a continuous increase in carried load with increasing offered load over the entire range of overloads considered. However at light overloads, carried

load may drop slightly when trunk reservation is implemented.

Using a mathematical model together with a simulation model, the performance of different dynamic routing schemes, (L_{R-P}, L_{R-I}, DAR) was studied. It was shown that for the symmetric uniformly loaded network considered, the mathematical models for the routing algorithms are identical. This was verified by comparing simulation results for the above three routing algorithms. The mathematical model is the same as the one for AAR with only one alternate path allowed. Analytic and simulation results do not exhibit instability for L_{R-P}, L_{R-I} and DAR. They do show a drop in carried load as the offered load is increased beyond a certain point. Using the mathematical model it was demonstrated that trunk reservation increased the carried load at overload.

In the second part of this chapter, the performance of different dynamic routing algorithms was studied on two large networks with realistic capacity and traffic matrices. The first network was a 10 node network subset of the main BT network with two traffic matrices for the morning and the evening. The results for the case of no traffic overloads illustrate that L_{R-P}, L_{R-I} and DAR performed close to each other and they all significantly outperformed RR and FR. Also the performance of the network deteriorates when TRP is implemented under no overload of traffic. However theoretical results for L_{R-P} and DAR under different values of traffic overloads showed that the optimum TRP varies with overload. For example for the morning traffic matrix the minimum blocking probabilities are at TRP=0 for overload values less than 10% , at TRP=1 for overload=10% and at TRP=5 for overload values greater than 10%.

The simulation results for abnormal traffic conditions showed that L_{R-I} and L_{R-P} outperform DAR with the best performance for the L_{R-I} scheme.

The second network considered is a 10 node network from Bell labs used elsewhere [25]. Simulation results in [25] are obtained for three L_{R-I} schemes with routing tables. The LA schemes in this thesis are easier to implement because they need no routing tables. Comparison of the results in [25] with our results for FR, RR, DAR L_{R-P} and L_{R-I} showed that $L_{R-P}^{(.1,.1)}$ is the best routing algorithm for this network.

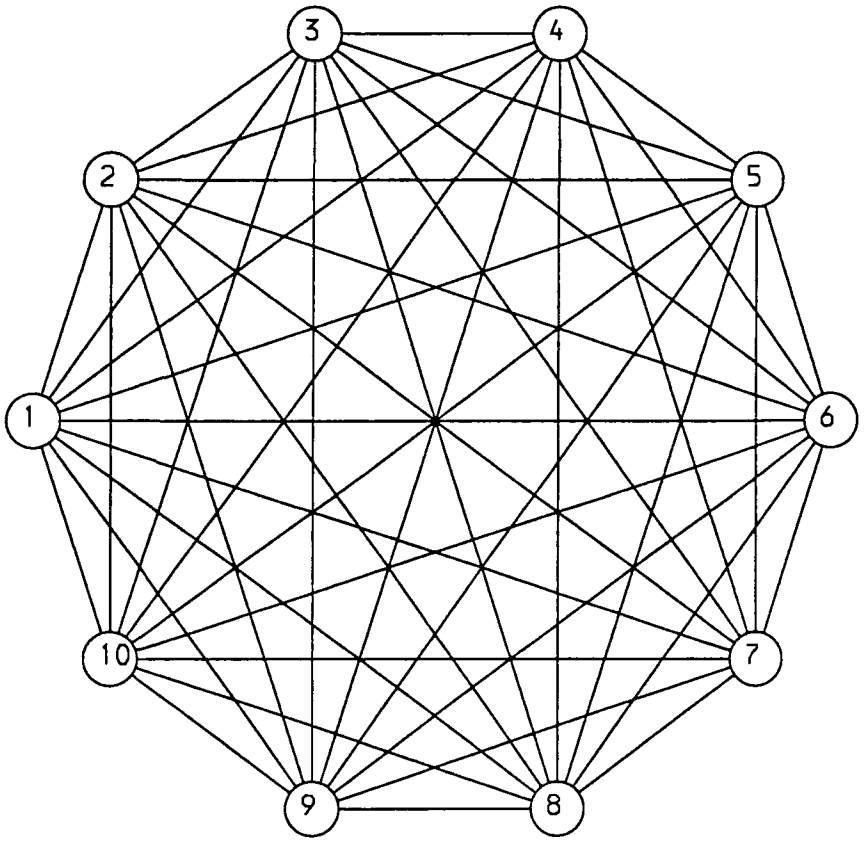
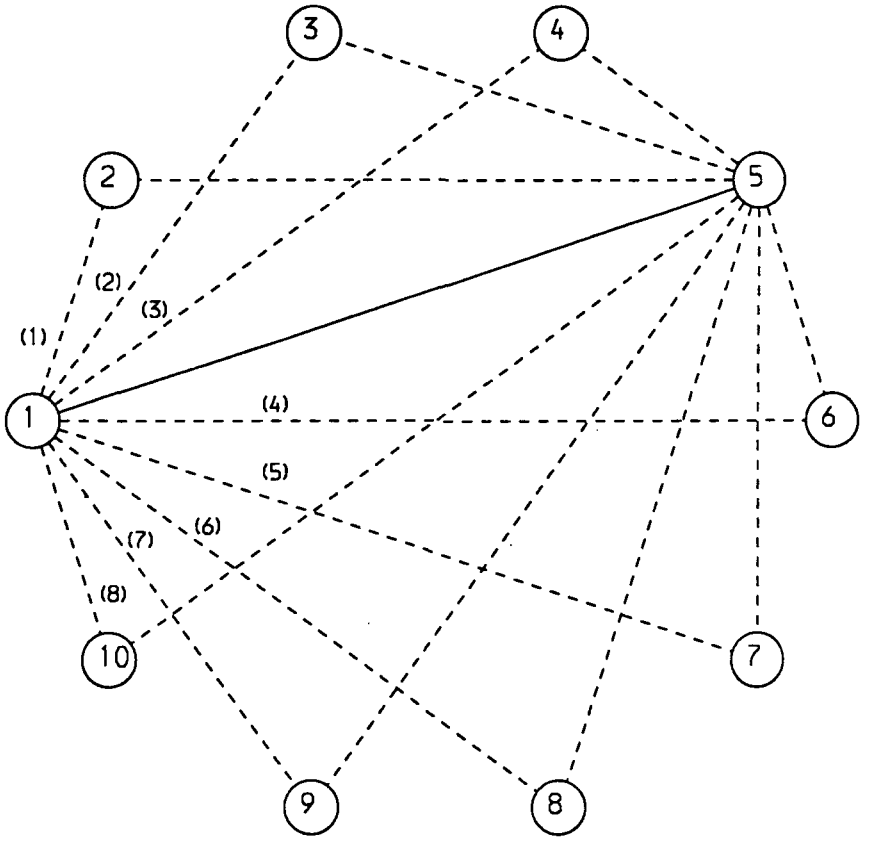


Fig 6.1 A 10 node fully connected network.

Origin Node	Destination Node	Direct Route	Overflow paths 1st	2nd	3rd
1	2	1-2	1-5-2	1-4-2	1-3-2
1	3	1-3	1-4-3	1-2-3	1-5-3
1	4	1-4	1-3-4	1-5-4	1-2-4
1	5	1-5	1-2-5	1-3-5	1-4-5
2	1	2-1	2-3-1	2-4-1	2-5-1
2	3	2-3	2-1-3	2-5-3	2-4-3
2	4	2-4	2-5-4	2-3-4	2-1-4
2	5	2-5	2-4-5	2-1-5	2-3-5
3	1	3-1	3-5-1	3-2-1	3-4-1
3	2	3-2	3-4-2	3-5-2	3-1-2
3	4	3-4	3-2-4	3-1-4	3-5-4
3	5	3-5	3-1-5	3-4-5	3-2-5
4	1	4-1	4-2-1	4-5-1	4-3-1
4	2	4-2	4-1-2	4-3-2	4-5-2
4	3	4-3	4-5-3	4-1-3	4-2-3
4	5	4-5	4-3-5	4-2-5	4-1-5
5	1	5-1	5-4-1	5-3-1	5-2-1
5	2	5-2	5-3-2	5-1-2	5-4-2
5	3	5-3	5-2-3	5-4-3	5-1-3
5	4	5-4	5-1-4	5-2-4	5-3-4

Fig 6.2. An example of symmetrical routing for a five node network.

routing from node 1 to node 5 (and from node 5 to node 1)



—— direct path

(n)
---- nth overflow path

Fig 6.3 Automatic Alternative Routing in a 10 node network with 8 overflow paths.

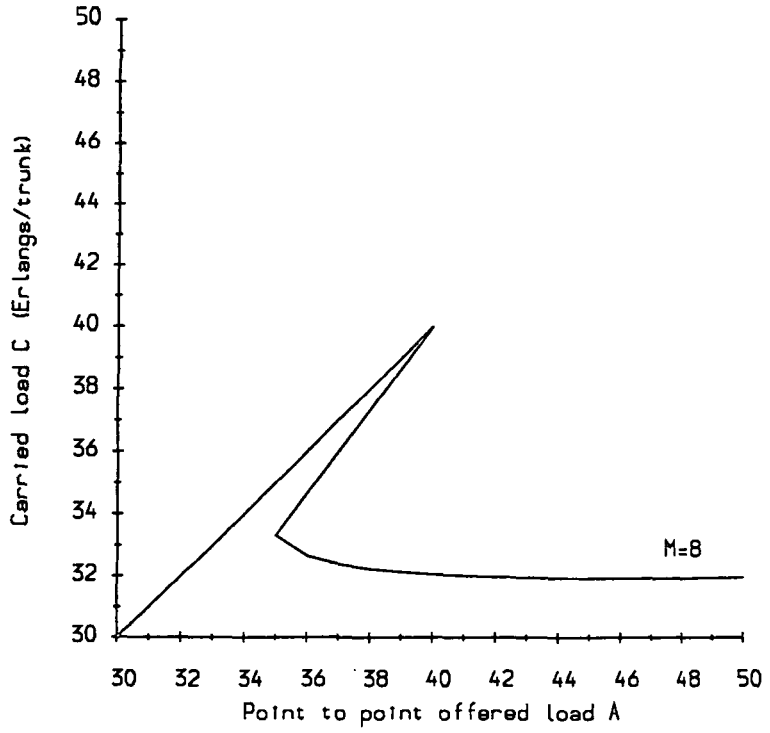


Fig 6.4 Carried load, AAR, M=8, TRP=0

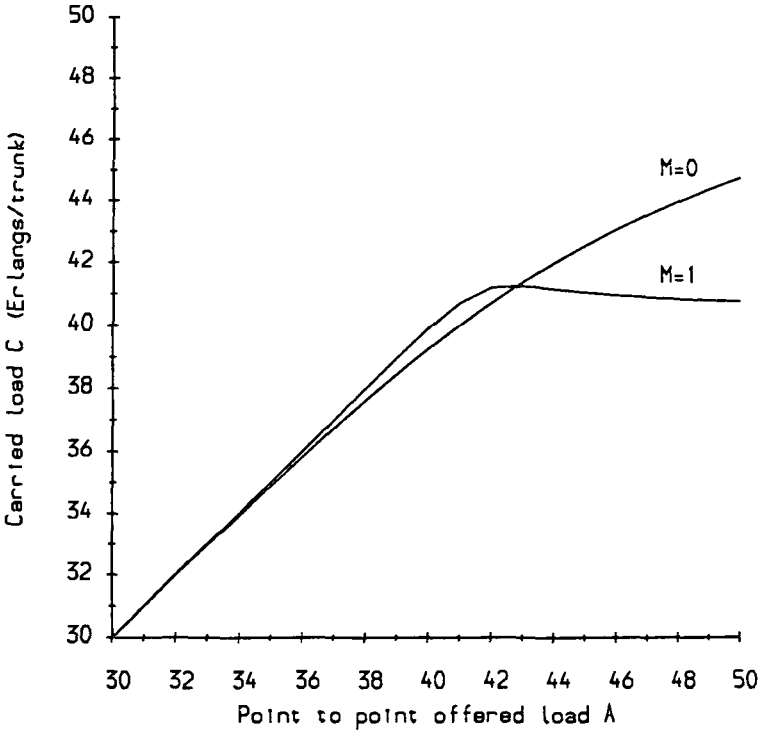


Fig 6.5 Carried load, AAR, M=0, M=1, TRP=0

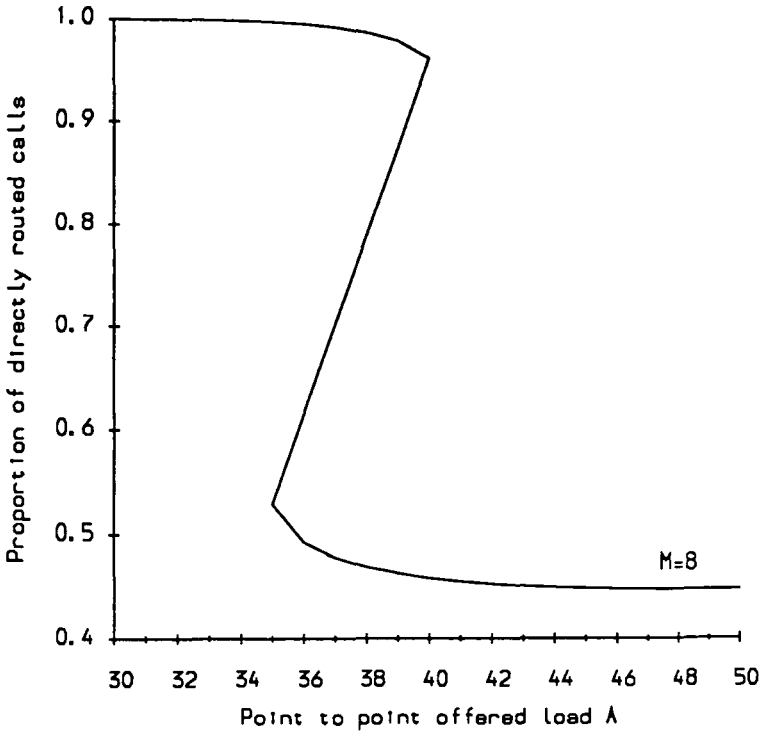


Fig 6.6 Proportion of directly routed calls, AAR, M=8, TRP=0

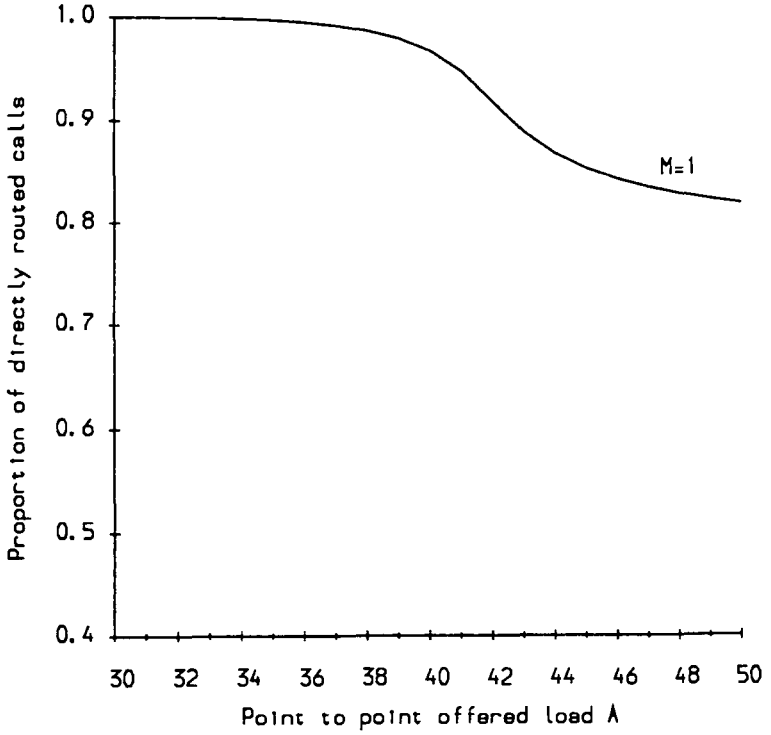


Fig 6.7 Proportion of directly routed calls, AAR, M=1, TRP=0

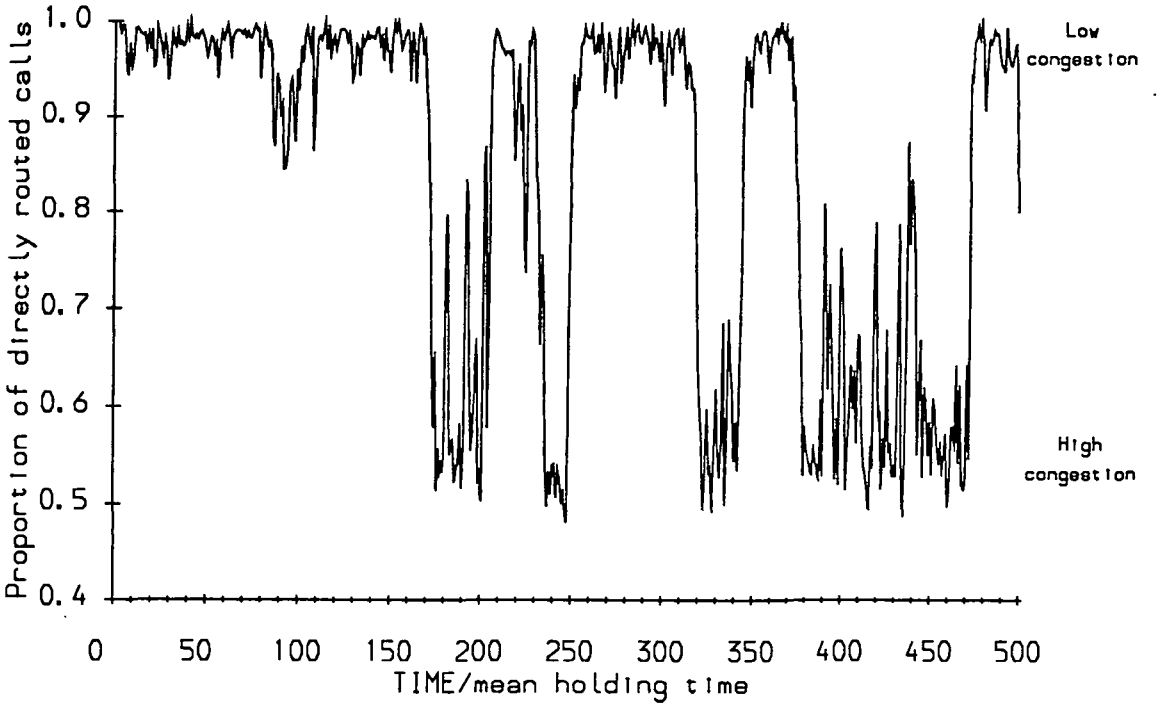


Fig 6.8 Simulation results, 10 node, c=50, A=38, TRP=0, AAR, M=8.

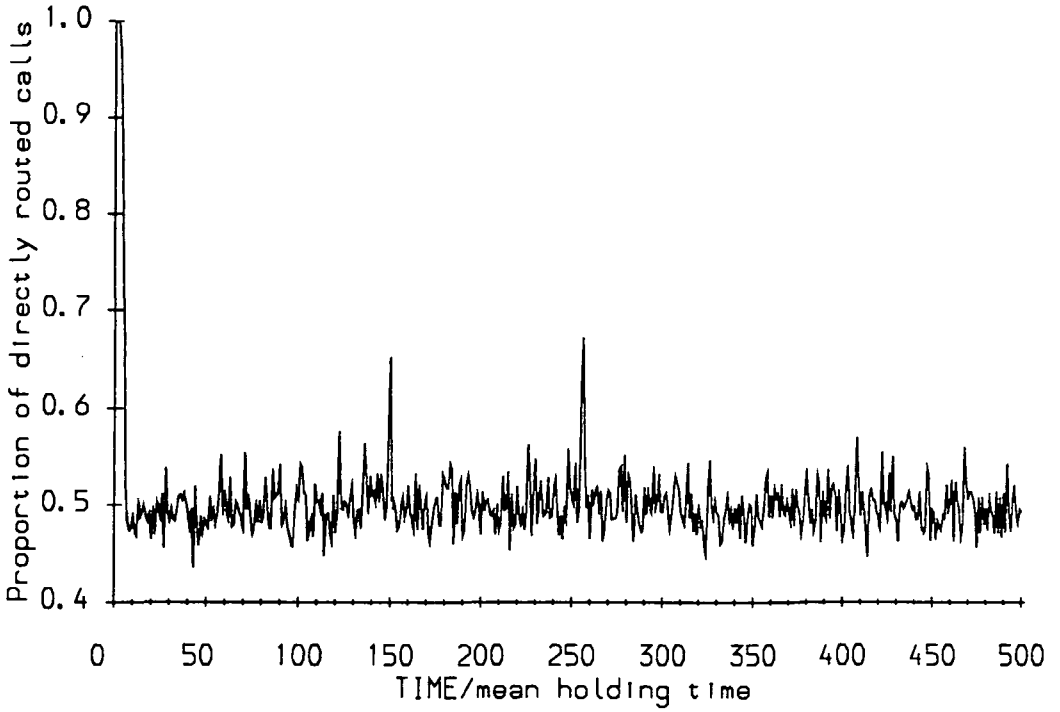


Fig 6.9 Simulation results, 10 node, c=50, A=42, TRP=0, AAR, M=8.

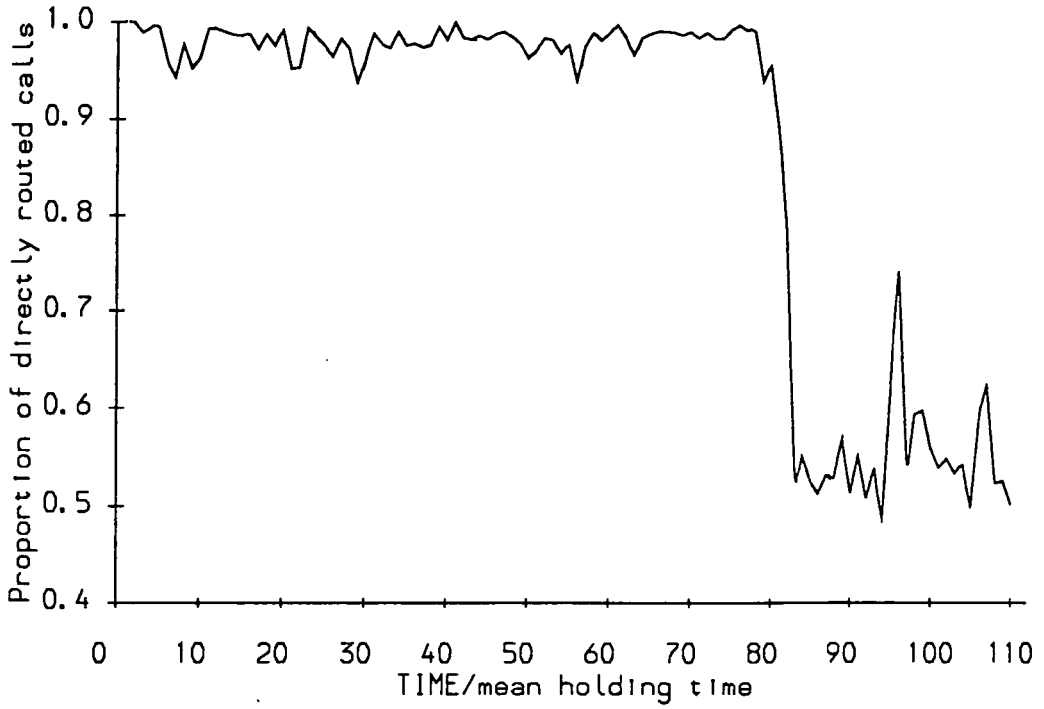


Fig 6.10 Simulation results, 10 node, c=50, A=38, A increased to 39 Erlangs at time=80.

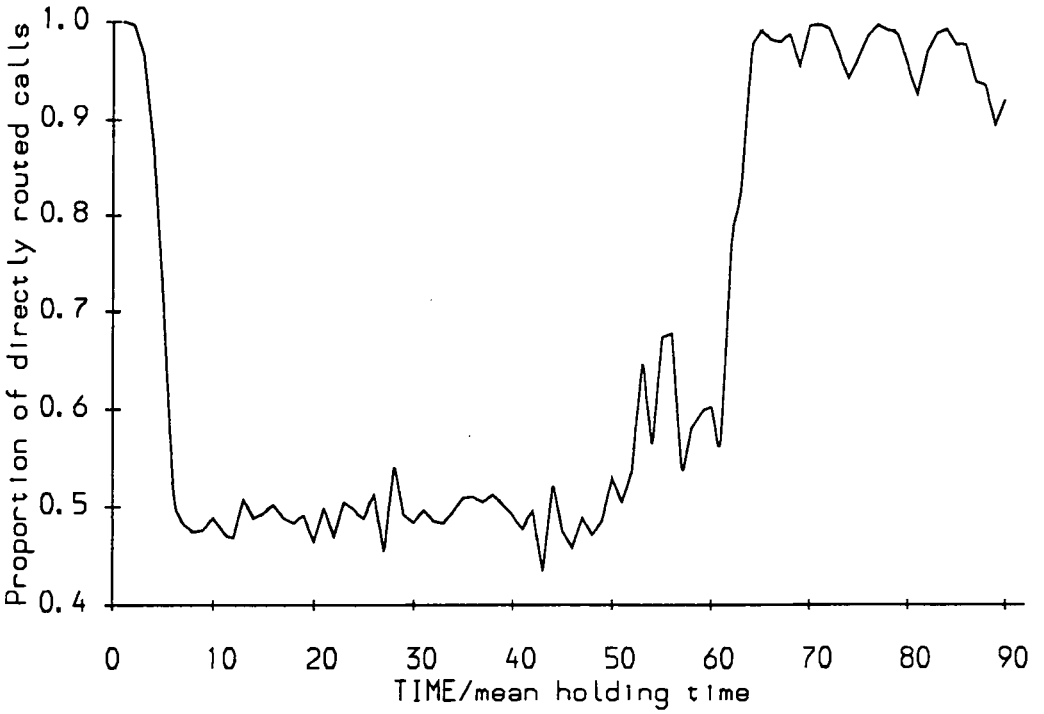


Fig 6.11 Simulation results, 10 node, c=50, A=42, A dropped to 38 Erlangs at time=50.

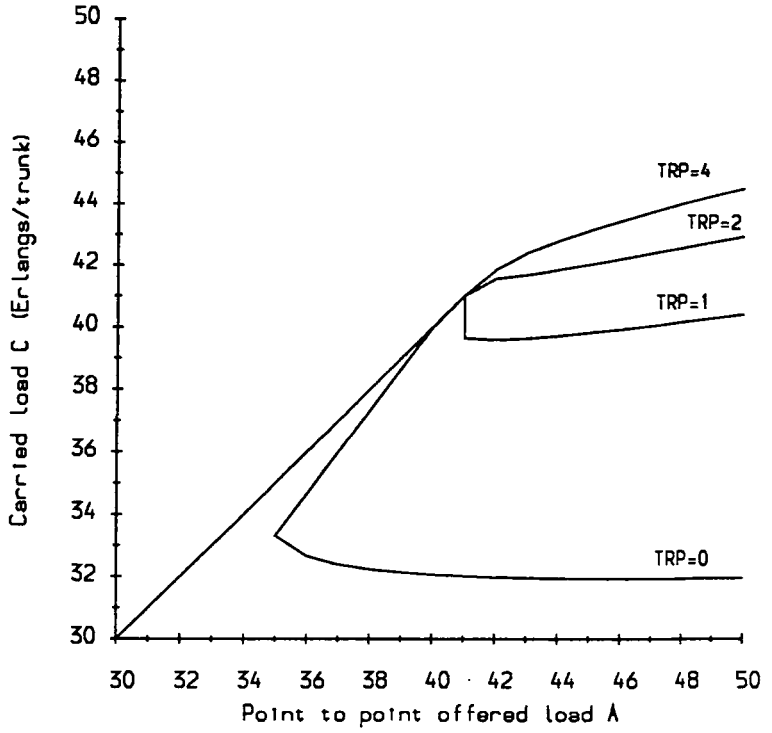


Fig 6.12 Carried load, AAR, M=8,
different TRPs.

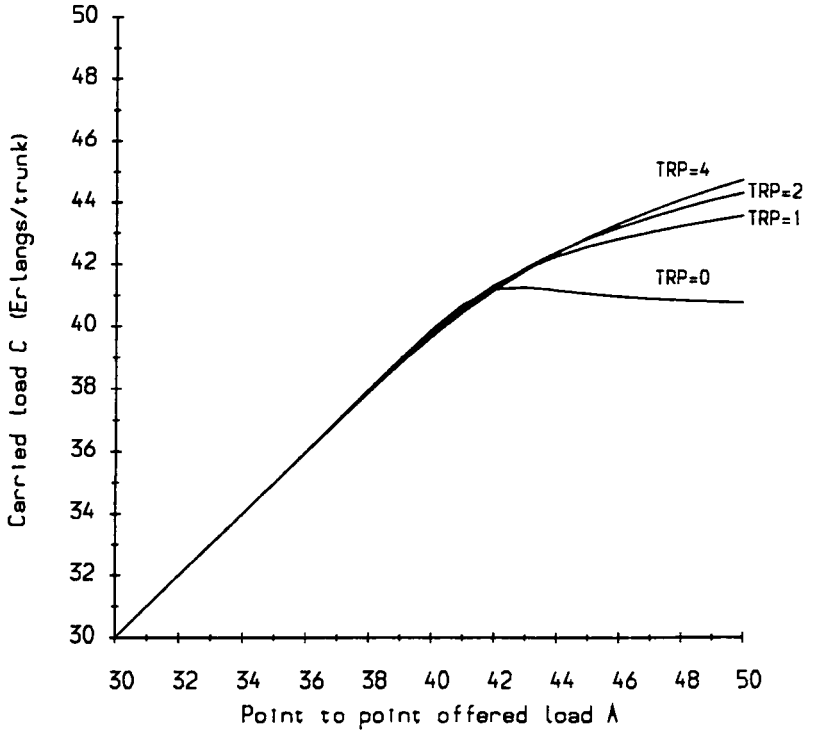


Fig 6.13 Carried load, AAR, M=1,
different TRPs.

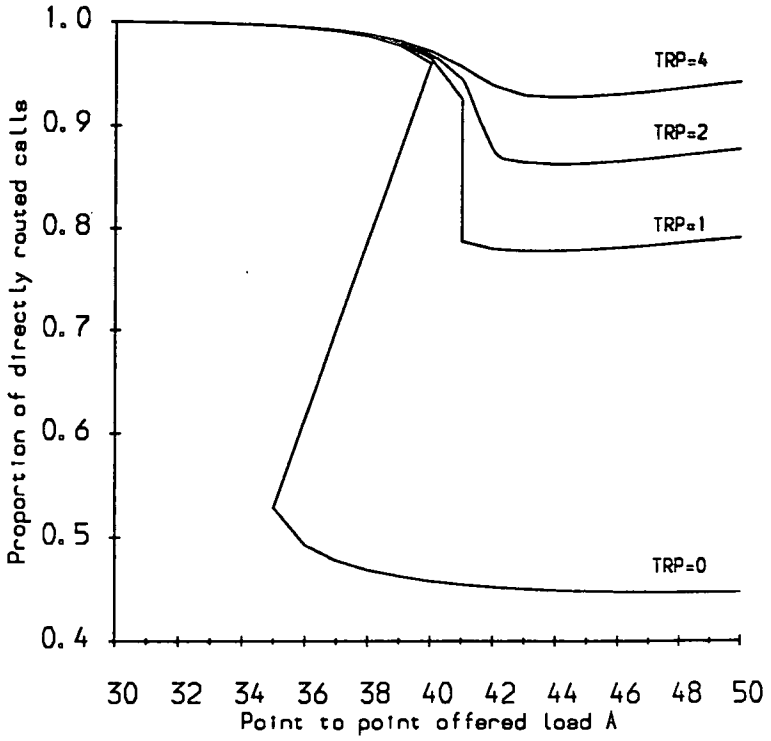


Fig 6.14 Proportion of directly routed calls, AAR, M=8, different TRPs.

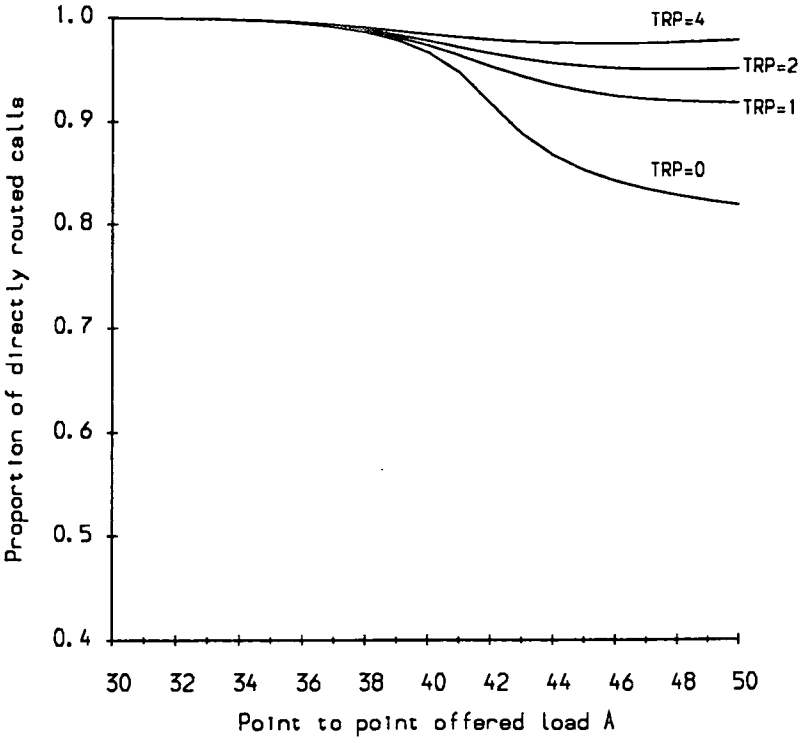


Fig 6.15 Proportion of directly routed calls, AAR, M=1, different TRPs.

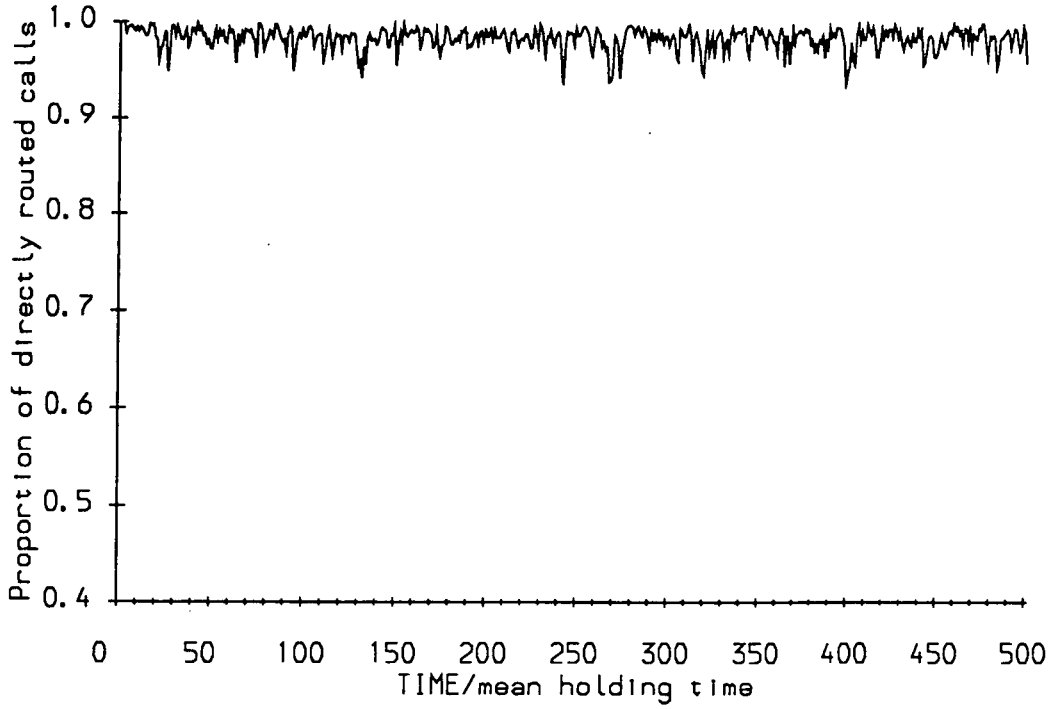


Fig 6.16 Simulation results, 10 node, c=50, A=38, TRP=1, AAR, M=8.

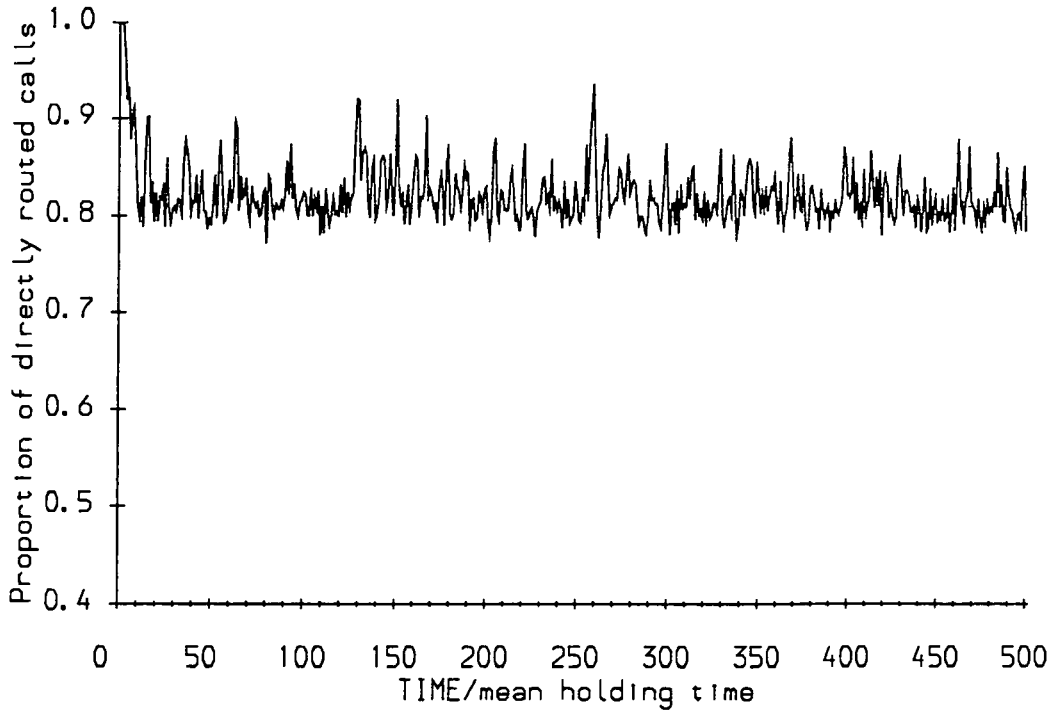


Fig 6.17 Simulation results, 10 node, c=50, A=42, TRP=1, AAR, M=8.

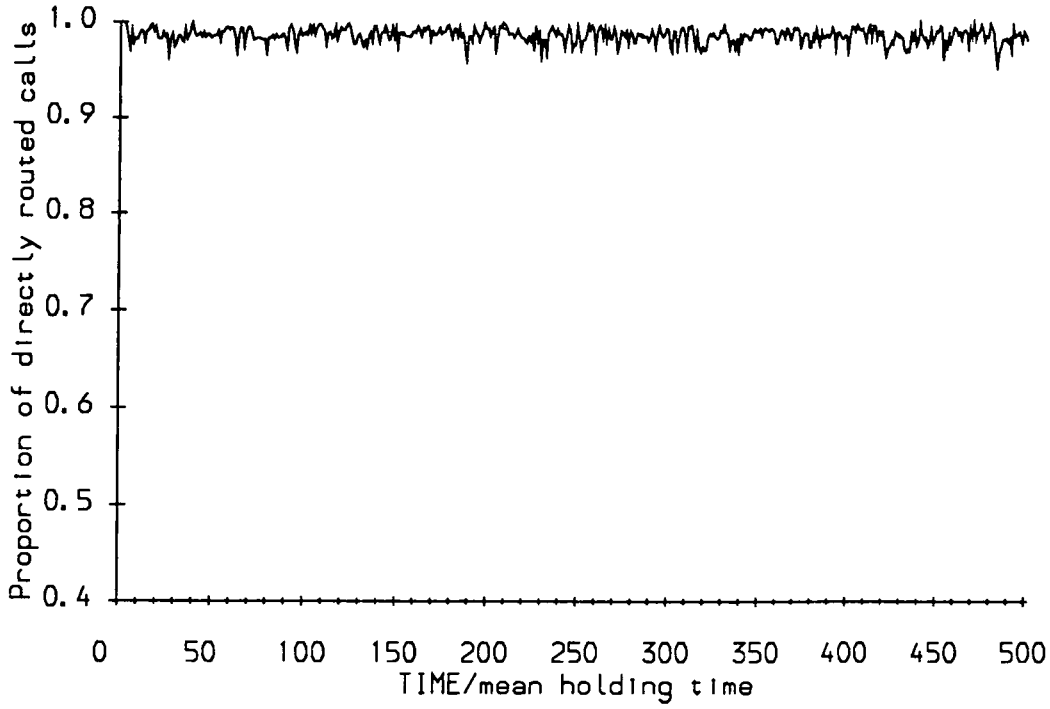


Fig 6.18 Simulation results, 10 node, c=50, A=38, TRP=0, LRP.

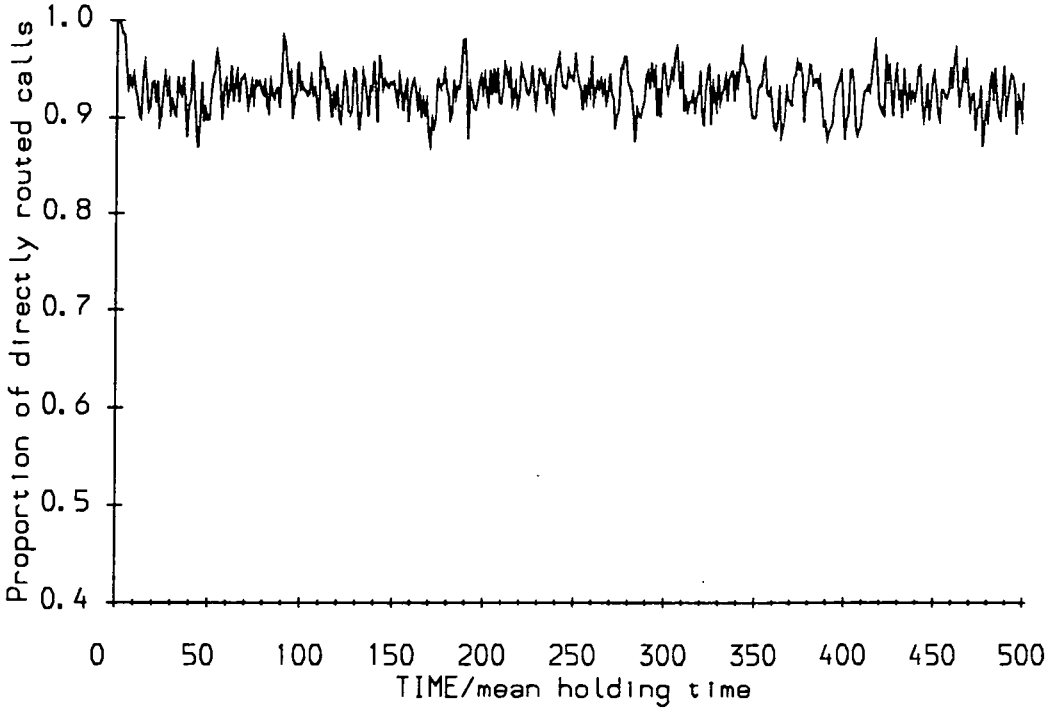


Fig 6.19 Simulation results, 10 node, c=50, A=42, TRP=0, LRP.

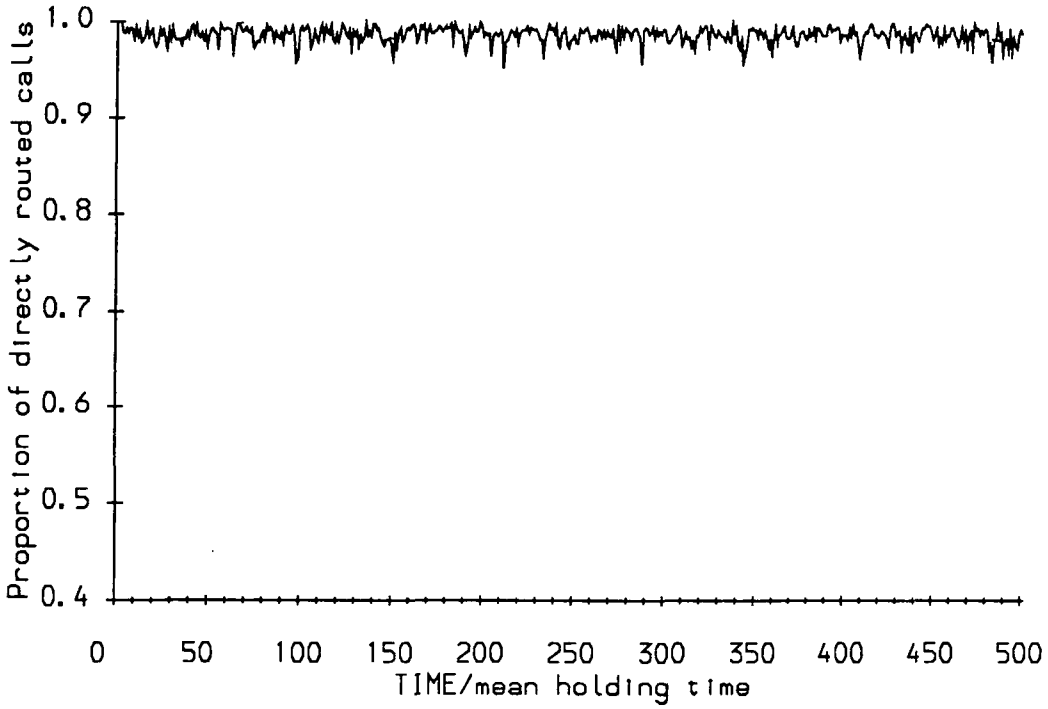


Fig 6.20 Simulation results, 10 node, c=50, A=38, TRP=0, DAR.

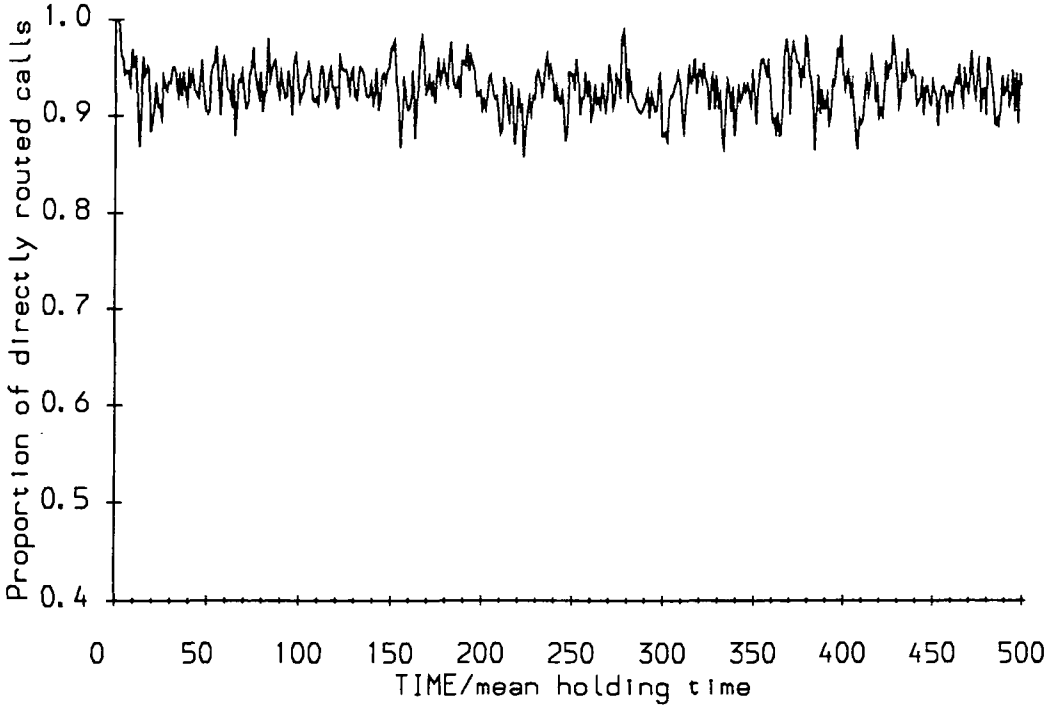


Fig 6.21 Simulation results, 10 node, c=50, A=42, TRP=0, DAR.

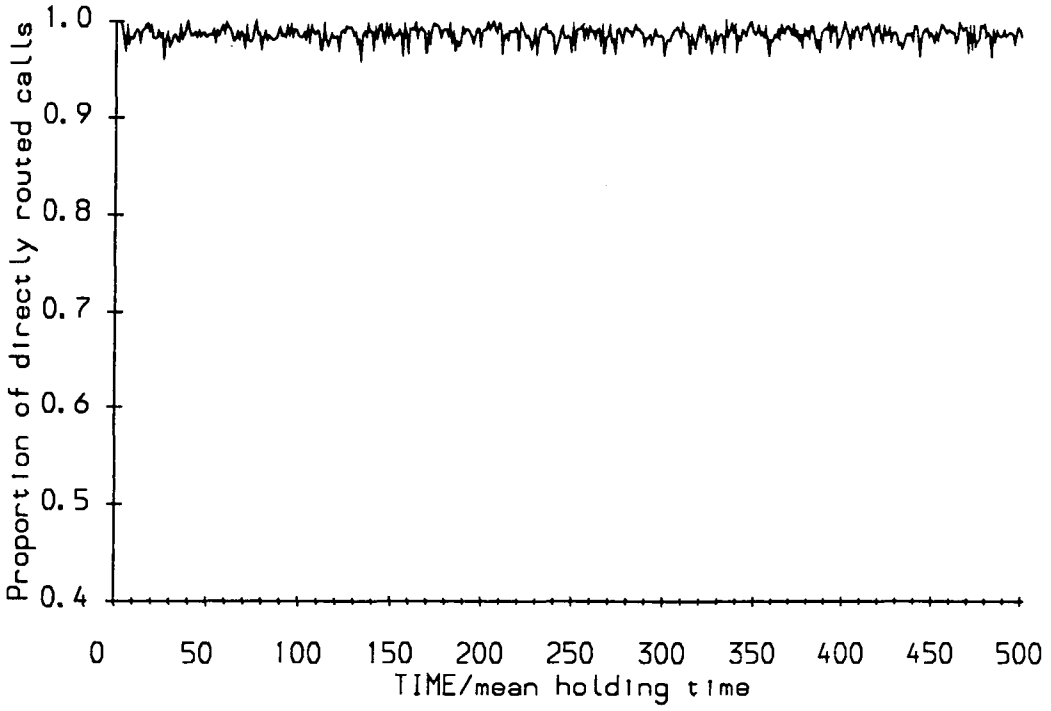


Fig 6.22 Simulation results, 10 node, c=50, A=38, TRP=0, LRI.

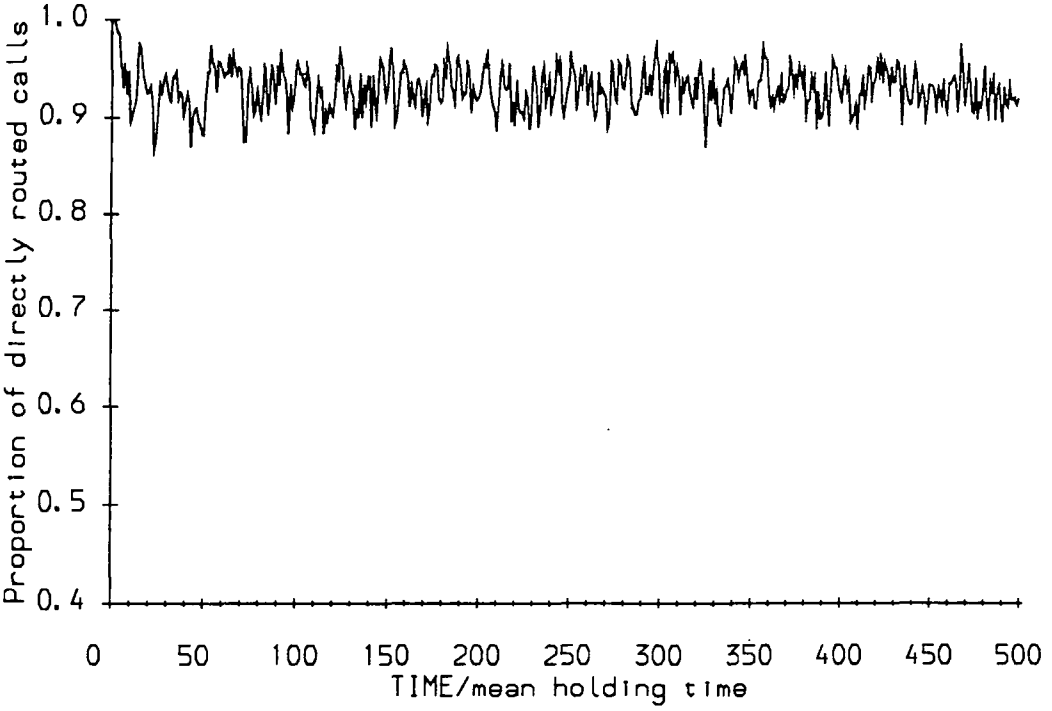


Fig 6.23 Simulation results, 10 node, c=50, A=42, TRP=0, LRI.

c=50 A=38 TRP=0

Scheme	C	PDC	Z
AAR (simulation)	36.77	0.8378	3.086
LRP (simulation)	37.93	0.9867	0.057
LRI (simulation)	37.93	0.9867	0.052
DAR (simulation)	37.92	0.9864	0.087
AAR (theory)	38.00 32.24	0.9858 0.4690	0.0 15.17
* (theory)	37.97	0.9871	0.087

Table 6.1 : Simulation (averages over 500 time units) and analytic results.

C Carried load per link

PDC Proportion of Directly Routed Calls

Z Percentage blocking probability

* LRP, LRI and DAR

c=50 A=42 TRP=0

Scheme	C	PDC	Z
AAR (simulation)	32.95	0.5029	21.43
LRP (simulation)	41.12	0.9283	2.00
LRI (simulation)	41.12	0.9300	1.99
DAR (simulation)	41.08	0.9294	2.083
AAR (theory)	31.98	0.4524	23.87
* (theory)	41.20	0.9471	1.90

Table 6.2 : Simulation (averages over 500 time units) and analytic results.

c=50 A=42 TRP=1

Scheme	C	PDC	Z
AAR (simulation)	40.21	0.8215	4.14
LRP (simulation)	41.29	0.9569	1.60
LRI (simulation)	41.27	0.9570	1.65
DAR (simulation)	41.28	0.9533	1.63
AAR (theory)	39.59	0.7788	5.73
* (theory)	41.34	0.9530	1.57

Table 6.3 : Simulation (averages over 500 time units) and analytic results.

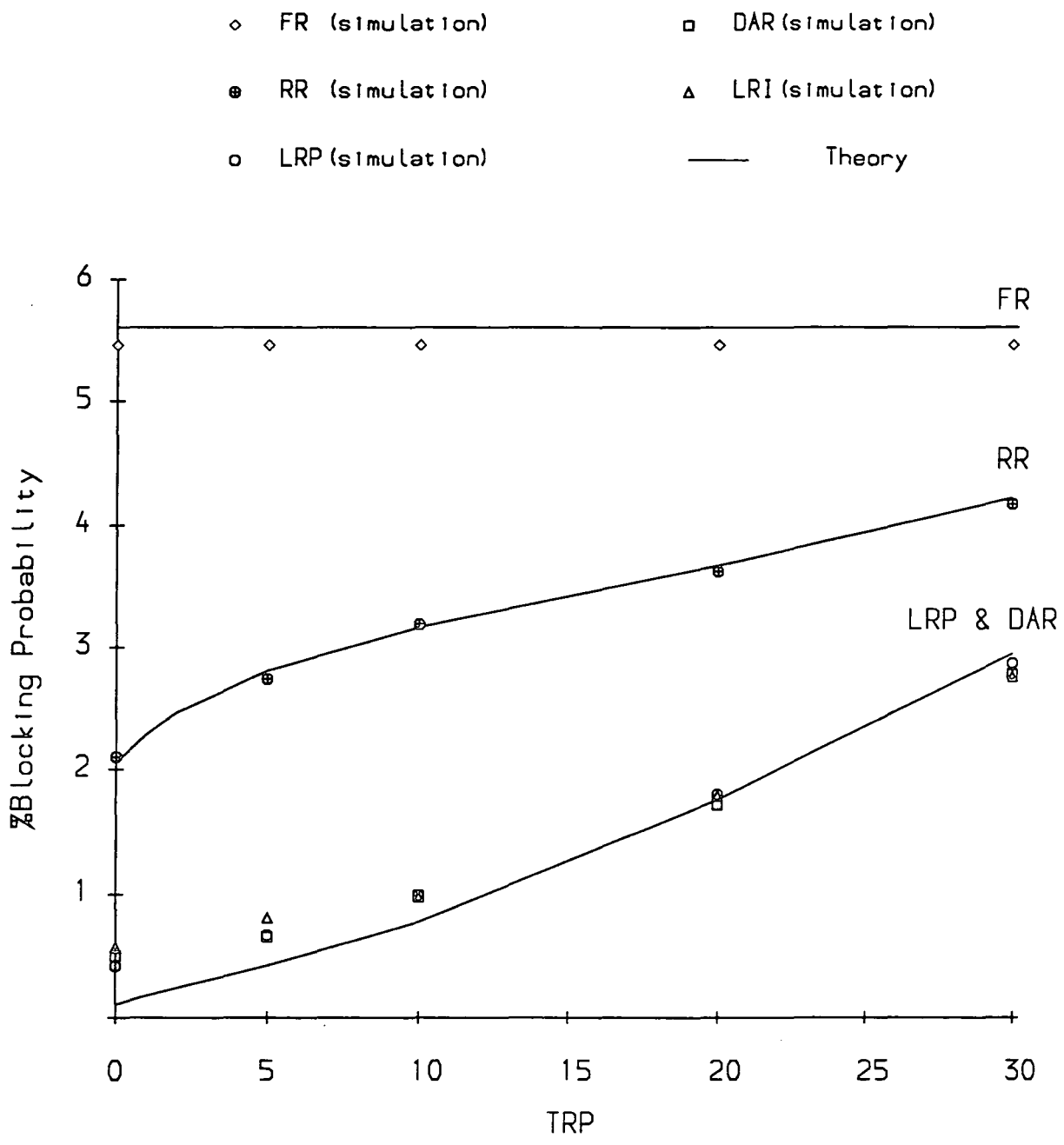


Fig 6.24 Theoretical and simulation results for the BT network with the morning traffic matrix, no overload.

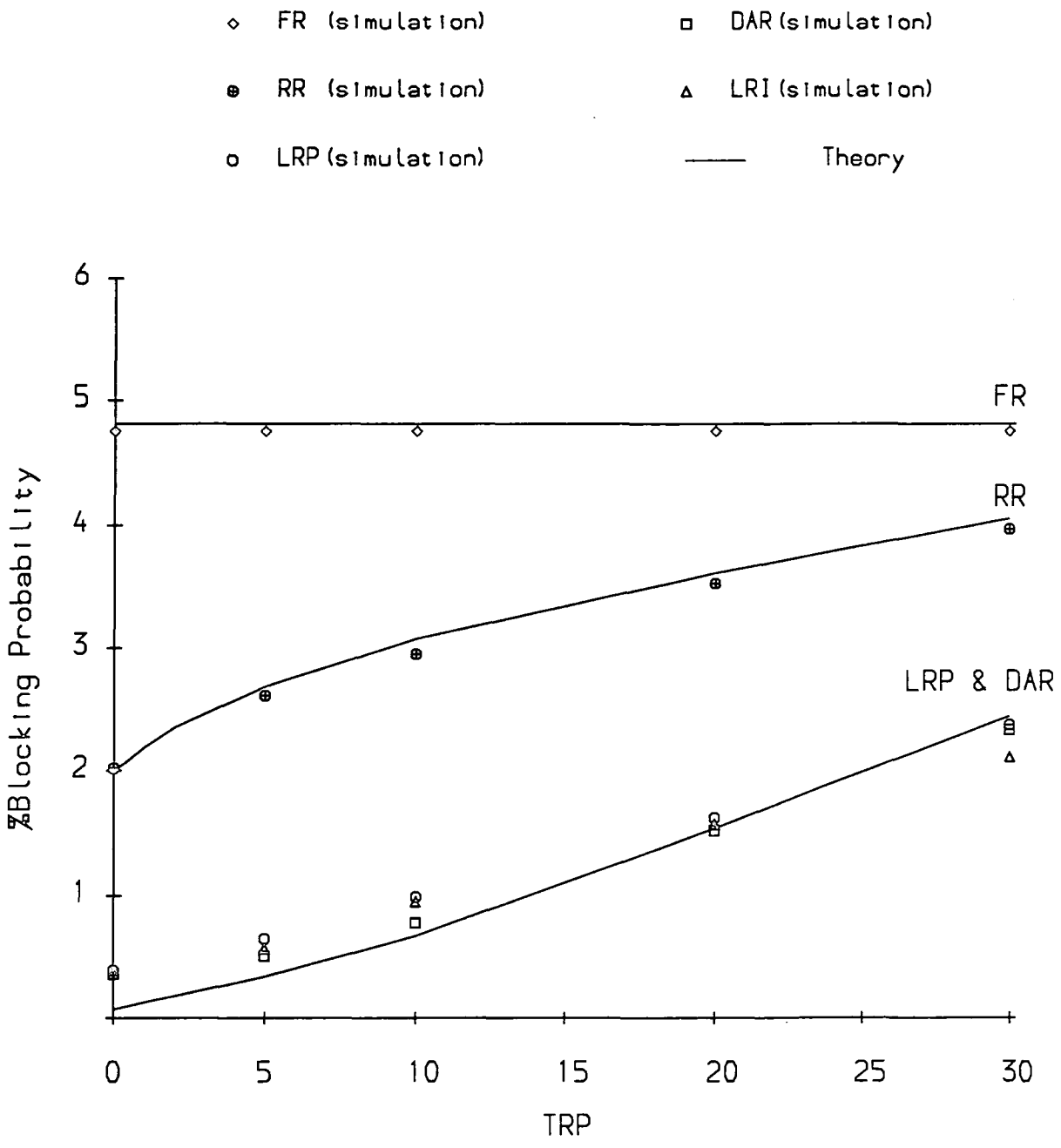


Fig 6.25 Theoretical and simulation results for the BT network with the evening traffic matrix, no overload.

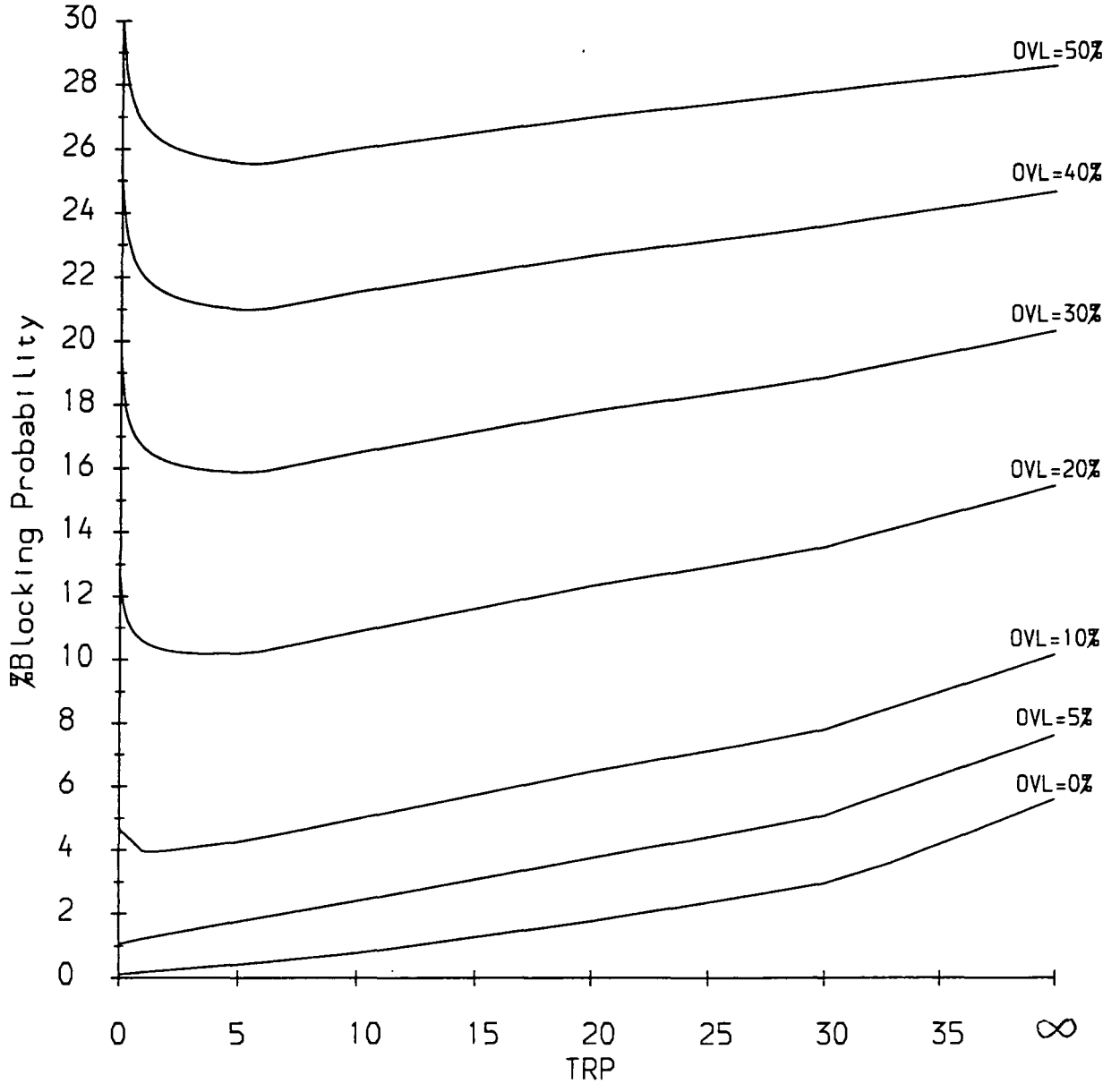


Fig 6.26 Theoretical results for LRP & DAR. BT network with the morning traffic matrix and different overlaods.

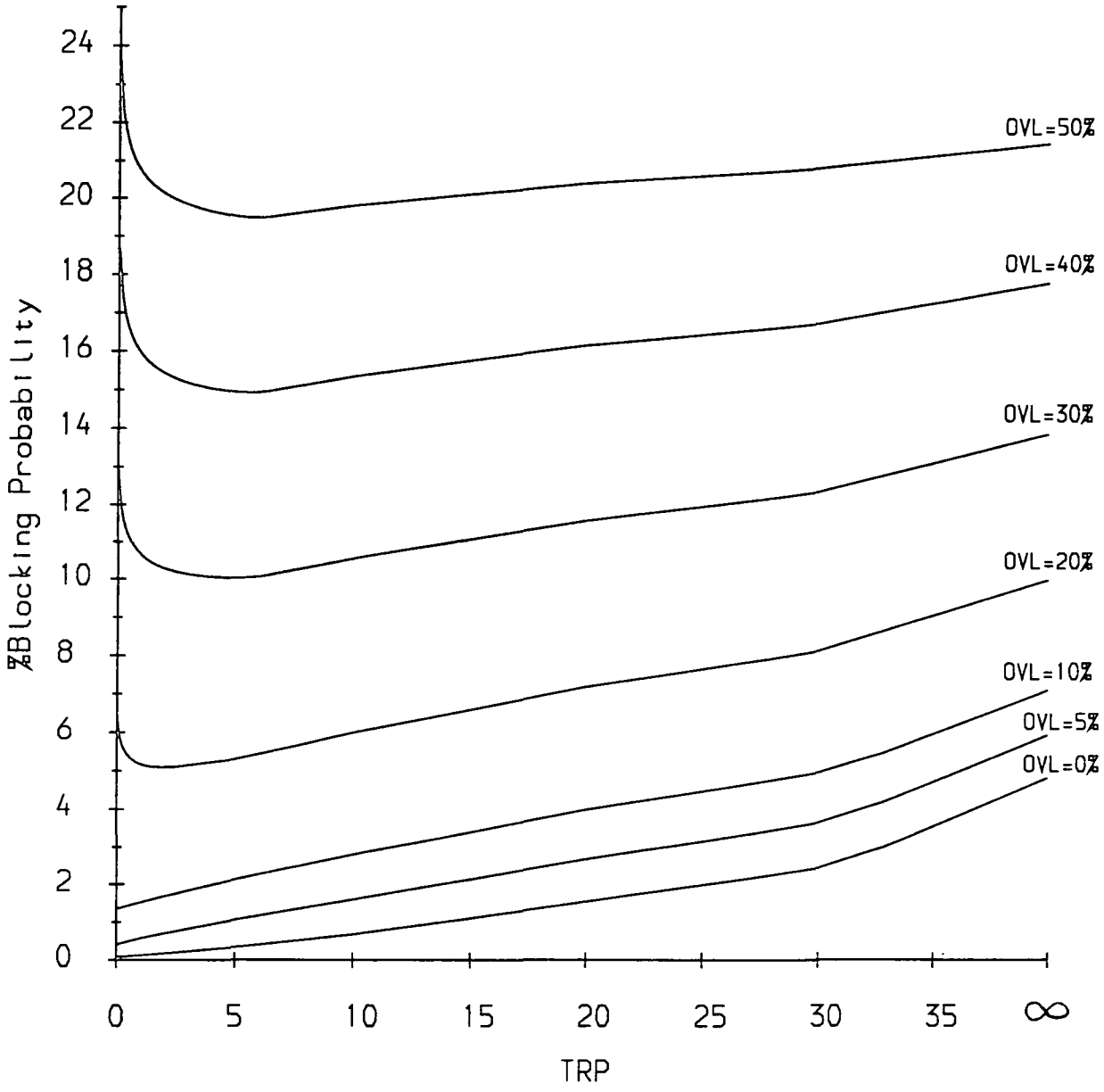


Fig 6.27 Theoretical results for LRP & DAR. BT network with the evening traffic matrix and different overlaods.

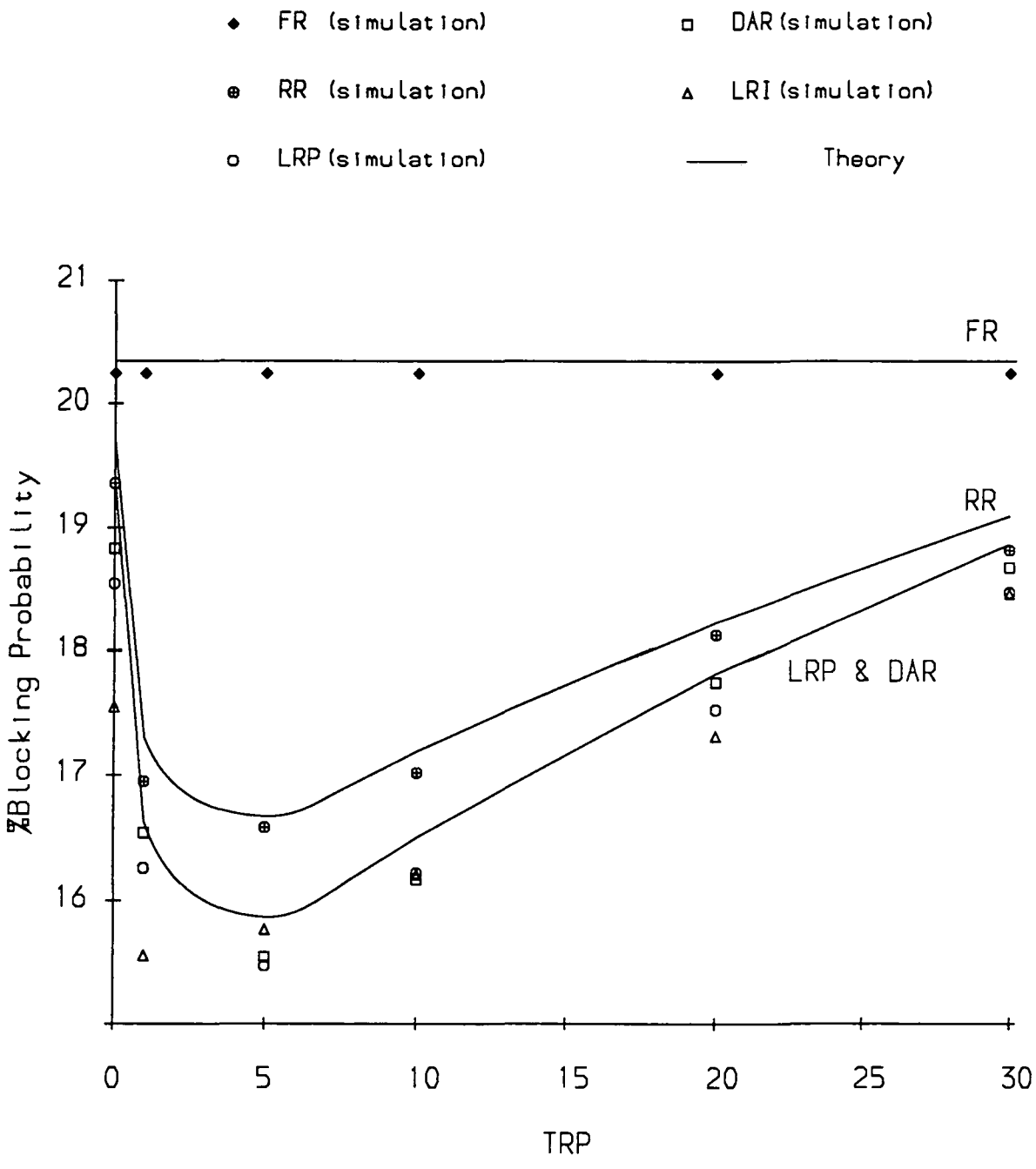


Fig 6.28 Theoretical and simulation results for the BT network with the morning traffic matrix, overload=30%.

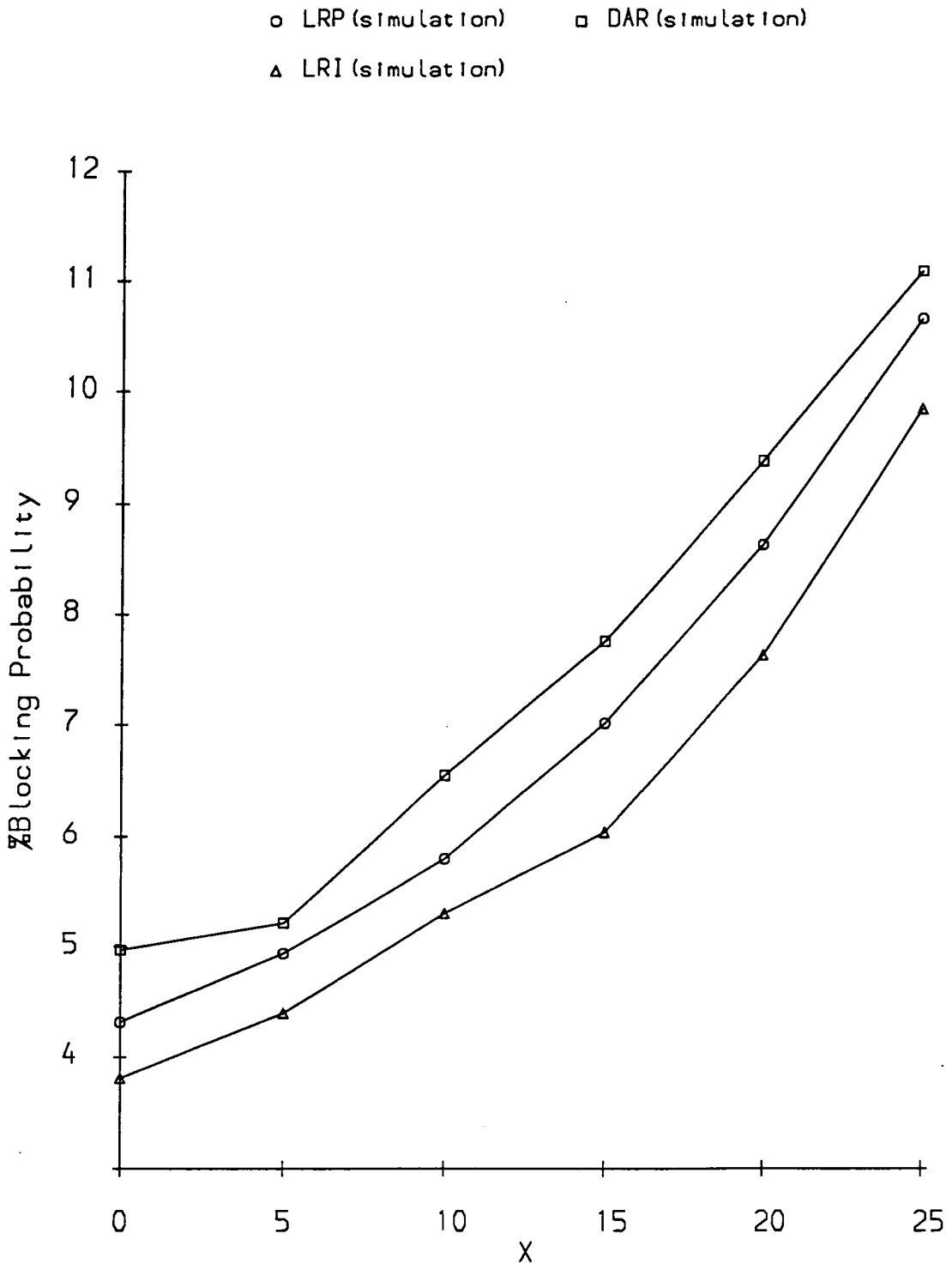


Fig 6.29 BT network with abnormal traffic conditions.

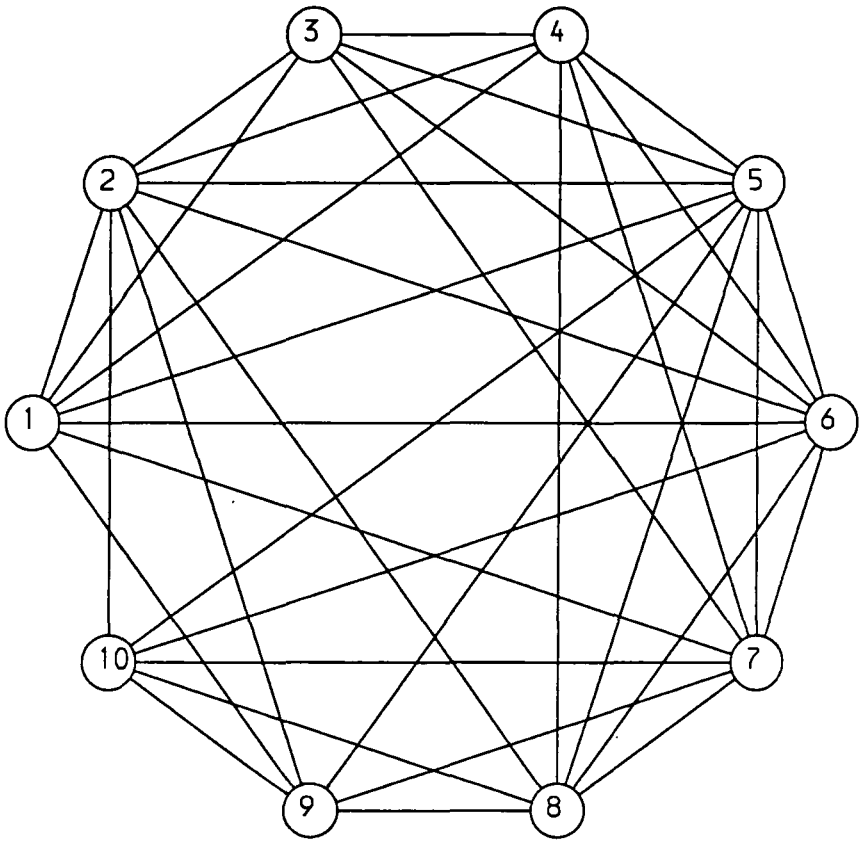


Fig 6.30 The 10 node Bell network.

Chapter Seven

Conclusions and further work

In this thesis we considered dynamic routing strategies in non-hierarchical circuit-switched networks. Following an introductory survey chapter two presented the fundamentals of stochastic learning automata. Two mathematical models were considered to predict the behaviour of different linear algorithms in a changing environment. It was shown that an $L_{R-\epsilon P}$ scheme attempts to equalize penalty probabilities while an L_{R-P} scheme tends to equalize penalty rates. The load equalization property of the L_{R-P} scheme was later used in chapter four to predict the theoretical overall blocking probability of a fully connected circuit-switched network.

In chapter three two simulation packages were developed. The first package can simulate a fully connected circuit-switched network of any size and the second package is similar to the first one but with a graphics front end. These packages were used in the subsequent chapters to perform experimental studies on the use of various dynamic routing strategies in circuit-switched networks.

Chapter four presented a theory for the blocking probabilities of FR, RR, L_{R-P}

and DAR and compared the theoretical predictions to simulation results for a five node fully connected network for different values of overload and TRP. An excellent agreement between analytical and simulation results was found for $TRP > 5$. For all routing strategies the results illustrate the sensitivity of the performance to the parameter TRP. In each case provided TRP is greater than 10 the results are invariant for further increase in TRP. Comparison of routing policies indicate that with TRP=0, all dynamic routing schemes (RR, LBA, L_{R-P} and DAR) are inferior to FR while with TRP=10 they all performed as well as FR. The test network was designed for FR and as a result any attempt to introduce alternate routing, only deteriorates the performance in these circumstances. TRP improves the network performance by reducing the amount of alternate routing. For TRP>10 a small amount of alternative routing takes place and dynamic routing schemes behave close to FR. Therefore under normal conditions for the network considered, FR is the best routing policy as it is less complicated than a dynamic routing plus trunk reservations. Next in this chapter the routing algorithms were compared under failure conditions and general overload. Once more with TRP=0 all dynamic routing schemes are inferior to FR but with TRP=10 the results for the various strategies are very similar. However in terms of the number of accepted calls for the failed link, TRP=10 results showed a significant deterioration compared to TRP=0.

In chapter five we compared LA with DAR on networks which were designed to force dynamic routing. First the influence of the learning parameter on the performance of the L_{R-I} scheme was investigated. It was shown that the algorithm can be

tuned for an optimal performance. Next the L_{R-I} and the DAR schemes were compared on a four node network with a single traffic source. The traffic source has two possible alternatives. The network was simulated for different ratios of the effective link capacities of the two alternate paths. It was shown that when the ratio is one, L_{R-I} outperforms DAR and when the ratio increases, the advantages of the L_{R-I} scheme are reduced. In the case of DAR a high ratio of link capacities on the two alternative paths is a favorable condition. Under these circumstances less switching will be required between the alternative paths for the DAR algorithm. It should be recalled that each time the DAR algorithm switches, a call is lost. Clearly as the ratio of the capacity of the two alternative paths approaches unity, then increasing switching will be required and therefore more calls will be lost. Results on five node networks deliberately designed to force dynamic routing illustrated an improved performance with L_{R-I}. The failure condition experiment showed a significant superiority of L_{R-I} over DAR for the case studied. Finally it was shown that for a traffic matrix not designed for FR, FR resulted in higher blocking probabilities than dynamic routing and any attempt to introduce TRP adversely affected the network performance.

In chapter six we studied the behaviour of different dynamic routing algorithms on more realistic larger networks. In the first part of this chapter, we considered the problem of instability when dynamic routing is used in symmetric uniformly loaded nonhierarchical networks. Experiments using a mathematical model for the AAR showed the existence of network instabilities for a range of overload. Simulation results showed that the network can experience quasi-stable behaviour, flipping between

low and high congestion states which may each last for considerable periods of time. It was shown through analysis that carried load drops dramatically at overload. The instability effect was controlled by applying trunk reservation to first-routed traffic. Such a control strategy reduces the amount of alternate routing in the network under overloads, and permits 1-link calls to use the trunks more efficiently. This reduces the quasi-stable behaviour and results in a continuous increase in carried load with increasing offered load over the entire range of overloads considered. Next we extended our studies to L_{R-P}, L_{R-I} and DAR and showed both through analysis and simulations that these algorithms do not exhibit instability. However they do show a drop in carried load as the offered load is increased beyond a certain point. Using the mathematical model it was demonstrated that trunk reservation increased the carried load at overload. However at very light overloads, carried load may drop when trunk reservation is implemented. This suggests the use of a triggering mechanism to activate trunk reservation only at larger overloads.

In the second part of chapter six, the performance of different dynamic routing algorithms was studied on two large networks with realistic capacity and traffic matrices. The first network is a 10 node network subset of the main BT network with two traffic matrices for the morning and the evening. The results for the case of no traffic overloads illustrated that L_{R-P}, L_{R-I} and DAR performed close to each other and they all significantly outperformed RR and FR. Also the performance of the network deteriorates when TRP is implemented under no overload of traffic. However theoretical results for L_{R-P} and DAR under different values of traffic overloads showed that

the optimum TRP varies with overload. The simulation results for abnormal traffic conditions showed that L_{R-I} and L_{R-P} outperform DAR with the best performance provided by the L_{R-I} scheme. The second network considered was a 10 node network from Bell labs used elsewhere [25]. Comparison of the results in [25] with our results for FR, RR, DAR, L_{R-P} and L_{R-I} showed that $L_{R-P}^{(.1,.1)}$ is the best routing algorithm for this network.

Further work is proposed as follows. In section 6.2 we studied instability in a symmetric uniformly loaded network for several dynamic routing algorithms. There is a need to study instability when L_{R-P}, L_{R-I} and DAR are implemented in networks of more general types. Two such networks are given in section 6.3 where we considered larger networks and compared different dynamic routing algorithms. The same sequence of experiments as in section 6.2 may be repeated to study whether these networks flip between low and high congestion states and whether the carried load drops at overload. Repeat attempts by customers may also intensify the unstable behavior. It is expected that at high congestion state, repeat attempts increase congestion and consequently increase the time that the network remains at that state.

In chapter three the simulation of a fully connected network with identical routing schemes operating at each node was considered. The simulator can be extended to hybrid routing by employing different routing schemes at different nodes of a network. More work is needed to investigate the practicality of hybrid routing and the selection of a proper routing algorithm at each node. Another possible extension to the simulator is the implementation of different TRPs for different links. The work

on the BT network in chapter six showed that there exists an optimum TRP at each overload value. Optimum TRP may be different for different links depending on the link capacity and arrival rate. Improved performance at overloads may be achieved by selecting an optimum TRP for each link of the network.

Conventionally a single automaton with a fixed set of actions is placed at each node of a network. Other forms of automata may be implemented in circuit-switched networks. For example in [33] a new LA algorithm with changing number of actions is suggested. In such automata, the set of available actions at every stage need only be a subset of the complete set of actions and could change from stage to stage. For the implementation to circuit-switched networks consider two levels of control at each node. At the bottom level, a number of automata are situated, each having a desired set of actions. At the top level, a mechanism is connected to all automata at the lower level and selects an automaton on the basis of a fixed probability distribution. This probability distribution can be changed with time. This form of control is particularly useful in large countries with different time zones where a different set of routes is considered according to the time of day or season of the year.

The convergence speed of the LA schemes is related to the size of the reward and penalty parameters. A fast convergence can be achieved by choosing large reward and penalty parameters but the results are expected to show large variances in steady state action probabilities. A possible solution may be the use of adaptive stepsize techniques where large reward and penalty parameters are selected initially and as the algorithm converges, the learning parameters are adjusted to smaller values to decrease the

steady state variances. In general the steady state probabilities will not be known and the choice of an adaptive strategy is non-trivial. In large networks the number of actions for a learning automaton becomes large and the convergence becomes slow. In addition, when all initial probabilities are chosen to be equal, each probability starts from a low value and therefore it takes a long time for the probabilities to converge to optimal values. Under such circumstances, a hierarchical structure of automata [34] may be placed at each node of the network to improve speed. This structure consists of several levels, each comprising of automata. Each action of an automaton at a certain level triggers an automaton directly below it. Research is needed to implement such systems of automata in communication networks and compare the speed of convergence to the case where a single automaton is placed at each node.

There is now an international trend towards Integrated Service Digital Networks (ISDNs). Research is urgently needed to study the interaction between circuit and packet switched traffic in such networks. Specifically, questions to be asked are how to route the types of traffic in a network. Should they follow the same paths or should each have its own set of paths for every source destination node pair? Moreover there is a need to study and compare the performance of different dynamic routing schemes in these classes of networks and to consider priority schemes for different classes of information.

References

[1] G. R. Ash, R. H. Cardwell, R. P. Murray, 'Design and Optimization of Networks with Dynamic Routing', The Bell System Technical Journal, Vol. 60, No. 8, October 1981, pp. 1787-1820.

[2] G. R. Ash, A. H. Kafker, and K. R. Krishnan, 'Servicing and real time control of networks with dynamic routing', The Bell System Technical Journal, Vol. 60, No. 8, October 1981, pp. 1821-1845.

[3] W. H. Cameron et al., 'Dynamic Routing for Intercity Telephone Networks', Tenth International Teletraffic Congress, Montreal, June 1983, Session 3.2, paper 3.

[4] F. Caron, 'Results of the Telecom Canada High Performance Routing Trial', 12th International Teletraffic Congress, Turino, June, 1988, pp. 2.2A.2.1-2.2A.2.10.

[5] R. G. Ash, 'Use of a trunk status map for real-time DNHR', 11th International Teletraffic Congress, 1985, paper 4.4 A-4.

[6] J. Lottin and J. P. Forestier, 'A Decentralized Control Scheme for Large Telephone Networks', Proceedings of the IFAC symposium, Toulouse, France, June, 1980, pp. 497-502.

[7] A. Girard and S. Hurtubise, 'Dynamic Routing and Call Repacking in Circuit-Switched Networks', IEEE Transactions on Communications, Vol. 31, No. 12, December 1983, pp. 1290-1294.

[8] B. R. Hurley, C. J. R. Seidl, W. F. Sewell, 'A Survey of Dynamic Routing Methods

for Circuit-Switched Traffic', IEEE Communication Magazine, Vol. 25, No. 9, September 1987, pp. 13-21.

[9] K. S. Narendra and P. Mars, 'The use of learning algorithms in Telephone Traffic Routing-A methodology', Automatica, Vol. 19, No. 5, pp. 495-502,1983.

[10] K. S. Narendra and R. M. Thathachar, 'On the behavior of a learning automaton in a changing environment with application to telephone traffic routing', IEEE Trans. Syst. Man Cybern., Vol. SMC-10, No. 5, May 1980, pp. 262,269.

[11] R. J. Gibbens, F. P. Kelly and P. B. Key, 'Dynamic Alternative Routing - Modelling and Behaviour', ITC12, Turino Italy, June 1988, Paper 3.4A.3.

[12] M. L. Tsetlin, 'On the behaviour of finite automata in random media', Avtomatika i Telemekhanika, Vol. 22, No. 10, Moscow, April 1961, pp. 1345-1354.

[13] V. I. Varshavskii and I. P. Vorontsova, 'On the behaviour of stochastic automata with variable structure,' Automaton and Remote Control, Vol. 24, No. 3, October 1963, pp. 327-333.

[14] K. S. Narendra and M. A. L. Thathachar, 'Learning automata-a survey', IEEE Trans. Syst. Man Cybern., Vol. SMC-4, No. 4, July 1974, pp323-324.

[15] S. Lakshmivarahan, 'Learning algorithms-theory and applications', New York: Springer-Verlang, 1981.

[16] M. F. Norman, 'Markov processes and learning models', Academic press, New York.

[17] P. R. Srikanta Kumar and K. S. Narendra, 'A Learning model for routing in telephone networks', Siam J. Control and optimization, Vol. 20, No. 1, January

1982, pp. 34-57.

[18] G. Gordon, 'System simulation', Prentice-Hall, Inc., New Jersey, 1978.

[19] M. Schwartz, 'Telecommunication networks', Addison-Wesley, 1987.

[20] J. Banks and J. S. Carson, 'Discrete-event System simulation', Prentice-Hall, Inc., New Jersey, 1984.

[21] R. R. Stacey and D. J. Songhurst, 'Dynamic Alternative routing in the British Telecom Trunk Network', Int Switching Symposium, March 1987, Phoenix, pp. B12.4.1-B12.4.5.

[22] K. S. Narendra, E. A. Wright, L. G. Mason, 'Application of Learning Automata to Telephone Traffic Routing and Control', IEEE Trans. on SMC, Vol. SMC-7, No.11, November 1977, pp. 785-792.

[23] K. S. Narendra and D. M. McKenna, 'Simulation Study of Telephone Traffic Routing Using Learning Algorithms - Part I', Yale University, New Haven, Ct., Technical Report S & IS 7806, December 1978.

[24] K. S. Narendra, P. Mars and M. S. Chrystal, 'Simulation Study of Telephone Traffic Routing Using Learning Algorithms - Part II', Yale University, New Haven, Ct., Technical Report S & IS 7907, October 1979.

[25] M. S. Chrystal, 'Adaptive Control of Communication Networks using Learning Automata', PhD thesis, Robert Gordon's Institute of Technology, Aberdeen, 1982.

[26] R. S. Krupp, 'Stabilization of alternate routing networks', IEEE International Conference on Communications, Philadelphia, June 1982, pp. 31.2.1-31.2.5.

[27] J. M. Akinpelu, 'The overload performance of engineered networks with non-

hierarchical and hierarchical routing', AT&T Bell Laboratories Technical Journal, Vol. 63, No. 7, September 1984, pp. 1261-1281.

[28] R. G. Gibbens, 'Dynamic Routing In Circuit-Switched Networks: The Dynamic Alternative Routing Strategy', PhD thesis, Cambridge University, July 1988.

[29] W-Y. NG, 'Dynamic Routing in Circuit-switched Networks: Simulation of the Routing Rule of Least Busy Alternative with Trunk Reservation', Part II Project, Dept of Engineering, University of Cambridge, 1985.

[30] Y. Nakagome and H. Mori, 'Flexible routing in the global communication network', Seventh International Teletraffic Congress, Stockholm, June 1973, paper 426.

[31] R. G. Ackerley, 'Hysterisis-type behaviour in networks with extensive overflow ', Br. Telecom Technol. J., Vol. 5, No. 4, October 1987, pp. 42-50.

[32] U. Korner and B. Walstrom, 'On symmetrical two-link routing in circuit-switched networks.', ITC 11, Kyoto 1985, pp. 855-860.

[33] M. A. L. Thathachar and B. R. Harita, 'Learning automata with changing number of actions', IEEE Trans. on SMC, Vol. SMC-17, No. 6, Nov/Dec 1987, pp. 1095-1100.

[34] M. A. L. Thathachar and K. R. Ramakrishnan, 'A hierarchical system of learning automata', IEEE Trans. on SMC, Vol. SMC-11, No. 3, March 1981, pp. 236-241.

Appendix 1

Simplified equations for B_1 and Q_2

To simplify the expressions for B_1 and Q_2 we first simplify the equation for p_0.

$$p_0 = [\sum_{j=0}^{m} \frac{\nu^j}{j!} + \sum_{j=m+1}^{c} \frac{A^{j-m}\nu^m}{j!}]^{-1} \qquad (4-12)$$

$$\frac{1}{p_0} = \sum_{j=0}^{m} \frac{\nu^j}{j!} + \frac{A\nu^m}{(m+1)!} + \frac{A^2\nu^m}{(m+2)!} + \ldots + \frac{A^{c-m}\nu^m}{c!}$$

$$\frac{1}{p_0} = \sum_{j=0}^{m} \frac{\nu^j}{j!} + \frac{\nu^m}{m!}\left(\frac{A}{(m+1)} + \frac{A^2}{(m+1)(m+2)} + \ldots + \frac{A^{c-m}}{(m+1)(m+2)\ldots c}\right)$$

$$\frac{1}{p_0} = \sum_{j=0}^{m} \frac{\nu^j}{j!} + \frac{\nu^m}{m!} \times \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}$$

B_1 is given by:

$$B_1 = \frac{A^{c-m}\nu^m}{c!} p_0 \qquad (4-10)$$

$$B_1 = \frac{\nu^m A^{c-m}}{m!(m+1)(m+2)\ldots c} p_0$$

Replacing the expression for p_0 into the above equation and dividing both numerator and denominator by $\nu^m/m!$ we get:

$$B_1 = \frac{\frac{A}{m+1}\frac{A}{m+2}\frac{A}{m+3}\cdots\frac{A}{c}}{\frac{\sum_{j=0}^{m}\frac{\nu^j}{j!}}{\frac{\nu^m}{m!}} + \sum_{i=1}^{c-m}\prod_{k=1}^{i}\frac{A}{(m+k)}}$$

$$B_1 = \frac{\prod_{k=1}^{c-m} \frac{A}{(m+k)}}{\frac{\sum_{j=0}^{m} \frac{\nu^j}{j!}}{\frac{\nu^m}{m!}} + \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}}$$

Let:

$$E(\nu, m) = \frac{\frac{\nu^m}{m!}}{\sum_{j=0}^{m} \frac{\nu^j}{j!}}$$

The expression for B becomes:

$$B_1 = \frac{\prod_{k=1}^{c-m} \frac{A}{(m+k)}}{\frac{1}{E(\nu,m)} + \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}}$$

Q_2 is given by:

$$Q_2 = \sum_{j=0}^{m-1} \frac{\nu^j}{j!} p_0 \qquad (4-11)$$

The expression for Q_2 can be simplified as follows:

$$Q_2 = (\sum_{j=0}^{m-1} \frac{\nu^j}{j!} + \frac{\nu^m}{m!} - \frac{\nu^m}{m!}) p_0$$

Replacing the expression for p_0 into the above equation:

$$Q_2 = \frac{\sum_{j=0}^{m} \frac{\nu^j}{j!} - \frac{\nu^m}{m!}}{\sum_{j=0}^{m} \frac{\nu^j}{j!} + \frac{\nu^m}{m!} \times \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}}$$

dividing both nemerator and denominator by $\sum_{j=0}^{m} \frac{\nu^j}{j!}$ we get:

$$Q_2 = \frac{1 - E(\nu, m)}{1 + E(\nu, m) \times \sum_{i=1}^{c-m} \prod_{k=1}^{i} \frac{A}{(m+k)}}$$

Appendix 2

The BT network specification

10 node capacity matrix

	2	3	4	5	6	7	8	9	10
1	30	30	60	90	90	120	30	810	420
2	0	60	90	90	90	90	30	720	240
3	0	0	30	60	90	60	30	90	60
4	0	0	0	90	120	30	30	60	120
5	0	0	0	0	510	120	90	120	180
6	0	0	0	0	0	180	30	150	240
7	0	0	0	0	0	0	60	120	120
8	0	0	0	0	0	0	0	60	90
9	0	0	0	0	0	0	0	0	60

10 node morning traffic matrix

	2	3	4	5	6	7	8	9	10
1	0.00	19.84	27.09	96.87	105.95	50.49	23.87	763.56	398.04
2	0.00	18.33	32.67	97.21	108.38	52.93	28.14	694.98	232.54
3	0.00	0.00	0.00	18.72	34.53	0.00	8.75	23.02	15.77
4	0.00	0.00	0.00	31.34	60.22	0.00	6.08	38.02	66.84
5	0.00	0.00	0.00	0.00	477.30	48.45	62.86	165.86	239.40
6	0.00	0.00	0.00	0.00	0.00	85.80	0.00	203.69	224.15
7	0.00	0.00	0.00	0.00	0.00	0.00	17.42	65.78	79.25
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	37.40	87.06
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

10 node evening traffic matrix

	2	3	4	5	6	7	8	9	10
1	0.00	41.70	67.39	61.45	82.24	112.05	19.96	662.27	251.57
2	0.00	32.18	81.90	64.46	83.69	125.11	27.00	588.81	146.13
3	0.00	0.00	0.00	28.30	70.11	0.00	17.48	48.61	9.91
4	0.00	0.00	0.00	84.72	144.79	0.00	25.63	85.61	42.05
5	0.00	0.00	0.00	0.00	204.10	108.20	42.08	111.52	150.28
6	0.00	0.00	0.00	0.00	0.00	198.62	0.00	156.66	141.65
7	0.00	0.00	0.00	0.00	0.00	0.00	40.38	145.63	49.93
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	35.31	54.81
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Mean Call holding time $\frac{1}{\mu} = 1$ time unit / call

Appendix 3

The Bell network specification

10 node capacity matrix

	2	3	4	5	6	7	8	9	10
1	43	52	12	10	3	15	0	19	0
2	0	40	17	1	11	0	20	17	8
3	0	0	13	26	1	12	0	0	0
4	0	0	0	0	18	8	11	0	0
5	0	0	0	0	172	148	28	28	8
6	0	0	0	0	0	13	11	0	5
7	0	0	0	0	0	0	18	8	6
8	0	0	0	0	0	0	0	72	36
9	0	0	0	0	0	0	0	0	49

Traffic matrix I

	2	3	4	5	6	7	8	9	10
1	.706	.001	.001	1.066	.42	1.319	.642	1.966	.534
2	0.00	.165	.001	.848	.186	.696	.958	1.241	.473
3	0.00	0.00	.001	.588	.489	1.057	.406	.001	.001
4	0.00	0.00	0.00	.425	.84	.763	.588	.225	.219
5	0.00	0.00	0.00	0.00	16.93	12.407	2.366	2.289	.634
6	0.00	0.00	0.00	0.00	0.00	.346	1.173	.001	.567
7	0.00	0.00	0.00	0.00	0.00	0.00	1.577	.471	.609
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.762	1.274
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.399

Traffic matrix II

	2	3	4	5	6	7	8	9	10
1	.001	2.074	.001	1.019	.53	1.092	.001	1.179	.61
2	0.00	3.526	.001	1.399	.353	.074	.00	2.77	.669
3	0.00	0.00	.687	.48	1.007	.001	.227	.308	.158
4	0.00	0.00	0.00	.472	1.09	1.242	.743	.168	.222
5	0.00	0.00	0.00	0.00	9.372	11.313	1.061	2.447	.691
6	0.00	0.00	0.00	0.00	0.00	.362	1.287	.001	.57
7	0.00	0.00	0.00	0.00	0.00	0.00	.78	.378	.338
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.684	1.186
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.754

Traffic matrix III

	2	3	4	5	6	7	8	9	10
1	.825	3.715	.001	1.919	.001	1.323	.631	1.922	1.166
2	0.00	1.406	.001	.48	.001	.001	.407	1.02	.455
3	0.00	0.00	.001	.568	1.271	.001	.669	.15	.001
4	0.00	0.00	0.00	.001	1.761	1.26	1.091	.011	.416
5	0.00	0.00	0.00	0.00	6.425	6.79	.484	.477	.468
6	0.00	0.00	0.00	0.00	0.00	.363	1.235	.001	.921
7	0.00	0.00	0.00	0.00	0.00	0.00	2.483	.001	1.171
8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.71	.555
9	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.943

Mean Call holding time $\frac{1}{\mu} = 10$ time units / call

Appendix 4

Simulation package I

```
C*********************************************************************
C This program allows simulation of a fully connected network of up to
C 10 nodes.
C Any of the following routing schemes can be selected:
C 1-FR  2-DAR  3-LRI  4-LRP  5-AAR 6-RR 7-LBA
C*********************************************************************
C      NO,ND,PROB    Arrays used in deciding identity of arriving calls,
C                    such that probability that the arriving call being
C                    for origin-destination pair NO(L) and ND(L) is
C                    PROB(L)-PROB(L-1)
C      TRAFIC        TRAFIC(L) is the call arrival rate in Erlang for
C                    origin-destination pair NO(L) and ND(L)
C      CAPAC         CAPAC(I,J) is the number of channels between nodes
C                    I and J.
C      LD            LD(I,K,J) is the current number of calls on
C                    origin-tandem-destination combination I,K,J
C      NACEPT        NACEPT(I,K,J) is the number of accepted calls for
C                    origin-tandem-destination combination I,K,J
C      NRJECT        NRJECT(I,J) is the number of rejected calls for
C                    origin-destination pair I,J
C      PLA           PLA(L,I,J) for LRP and LRI schemes is the probability
C                    of selecting the Lth alternative for O-D pair I,J
C      TDAR          TDAR(I,J) for DAR scheme is the current tandem node
C                    for O-D pair I,J
C      STR           Total traffic
C      OVF           Overload factor
C      RN            Time to output results (results are outputed every
C                    unit as RN is incremented by one each time)
C      TR            Time of call arrival
C      TINT          Inter-arrival time
C      UNIT          Maximum simulation time
C      TRP           Trunk Reservation Prameter
C      RW            Reward parameter
C      PN            Penalty parameter
C      SCH           Routing scheme code number:
C                    FR=1 DAR=2 LRI=3 LRP=4 AAR=5 RR=6 LBA=7
C      NN            Number of nodes
C
       DIMENSION  NO(45), ND(45), LD(10,0:10,10),PROB(45)
       DIMENSION NRJECT(9,10), NACEPT(9,0:10,10)
       DIMENSION PLA(8,9,10),TRAFIC(45)
       INTEGER SEED(3)
       INTEGER TDAR(10,10)
       INTEGER CAPAC(10,10),TRP,NN,NODP,SCH
       DOUBLE PRECISION RN,TR,UNIT
       INTEGER ORIG, DEST
       INTEGER TLOAD,AAR,RANODE,DAR,RR,TEST
       DATA TR,RN/0.0,1.0/
       DATA NRJECT,NACEPT/1080*0/
       DATA LD/1100*0/
       DATA CAPAC/100*0/
       DATA TRAFIC/45*0/
       DATA NO/45*0/
       DATA ND/45*0/
       DATA PROB/45*0/
       DATA PLA/720*0/
```

```
C************************** Main Program ********************************
       CALL INPUT(NN,NODP,OVF,X,SCH,CAPAC,TRAFIC,TRP,UNIT,IREF,
     1             RW,PN,SEED)
       CALL SETUP(NN,SCH,NODP,PROB,NO,ND,PLA,TDAR,SEED(3),TRAFIC,STR)
10     CALL ARIVAL(TR,TINT,ORIG,DEST,NO,ND,PROB,STR,SEED(1),NODP)
       IF (TR.GE.RN) THEN
            CALL OUTBP(NACEPT,NRJECT,RN,NN)
            RN=RN+1.
       ENDIF
       IF(RN.GT.UNIT) GO TO 1000
       CALL DEPART(LD,TINT,SEED(1),ORIG,DEST,NN)
       CALL ROUTE(LD,CAPAC,SEED(2),SEED(3),PLA,TDAR,NN,SCH,NACEPT,
     1             NRJECT,RW,PN,TRP,ORIG,DEST)
       GOTO 10
 1000  WRITE(8,*)'End of simulation.'
       END
C***********************************************************************
C Returns a random number between 0 and 1.
C***********************************************************************
       FUNCTION UNIF(IS)
       REAL UNIF
       IS=IS*65539
       IF (IS.LT.0) IS=IS+2147483647+1
       UNIF=IS*4.656613E-10
       RETURN
       END
C***********************************************************************
C This subroutine calculates: 1-origin and destination NI and NJ for the
C next call arrivall and  2-the interarrival time SR.
C***********************************************************************
       SUBROUTINE ARIVAL(TR,SR,NI,NJ,II,JJ,P,SUMLD,IX,NTR)
       DIMENSION II(45),JJ(45),P(45)
       DOUBLE PRECISION TR
       INTEGER IX
C
       RAND=UNIF(IX)
       SR=-1*LOG(RAND)/SUMLD
       TR=TR+SR
C
       RAND=UNIF(IX)
       DO 10 L=1,NTR
             IF (RAND.LT.P(L)) GOTO 20
10     CONTINUE
20     NI=II(L)
       NJ=JJ(L)
       RETURN
       END
C***********************************************************************
C Clears down calls.
C***********************************************************************
       SUBROUTINE DEPART(NX,SR,IW,NI,NJ,NN)
       DIMENSION NX(10,0:10,10)
C
       DO 10   I=1,NN-1
          DO 20   J=I+1,NN
             DO 30   K=0,NN
               IF (K.EQ.I.OR.K.EQ.J) GOTO 30
               IF (NX(I,K,J).EQ.0) GOTO 30
```

```
                RLMDA=NX(I,K,J)*SR
                P=EXP((-1)*RLMDA)
                Q=1.0
                II=0
40              RAND=UNIF(IW)
                Q=Q*RAND
                IF(Q.GE.P) THEN
                   II=II+1
                   GOTO 40
                ENDIF
                NX(I,K,J)=NX(I,K,J)-II
                NX(I,K,J)=MAX(0,NX(I,K,J))
                NX(J,K,I)=NX(I,K,J)
30            CONTINUE
20         CONTINUE
10      CONTINUE
        RETURN
        END
C*************************************************************************
C Returns total load carried on link (NI,NJ).
C*************************************************************************
        INTEGER FUNCTION TLOAD(NX,NI,NJ,NN)
        DIMENSION NX(10,0:10,10)
        INTEGER NXIJ
        I=MIN(NI,NJ)
        J=MAX(NI,NJ)
        NXIJ=NX(I,0,J)
        DO 10 L=I+1,NN
            IF(L.EQ.J) GOTO 10
            NXIJ=NXIJ+NX(I,J,L)
10      CONTINUE
        DO 20 L=1,J-1
            IF(L.EQ.I) GOTO 20
            NXIJ=NXIJ+NX(L,I,J)
20      CONTINUE
        DO 30 L=1,I-1
            NXIJ=NXIJ+NX(L,J,I)
30      CONTINUE
        DO 40 L=J+1,NN
            NXIJ=NXIJ+NX(J,I,L)
40      CONTINUE
        TLOAD=NXIJ
        RETURN
        END
C*************************************************************************
C This subroutine routes the call directly if possible and if not tries
C to route the call alternately using a dynamic routing scheme.
C*************************************************************************
        SUBROUTINE ROUTE(NX,C,IY,IZ,A,TDAR,NN,SCH,NACEPT,NRJECT,
     1                   RW,PN,TRP,NI,NJ)
        DIMENSION NX(10,0:10,10),A(8,9,10)
        DIMENSION NACEPT(9,0:10,10),NRJECT(9,10)
        INTEGER C(10,10),TRP,SCH,TDAR(10,10)
        INTEGER AAR,TLOAD,DAR,RR
        INTEGER IZ,IY
C
        NXIJ=TLOAD(NX,NI,NJ,NN)
        IF (NXIJ.LT.C(NI,NJ)) THEN
```

```
                NX(NI,0,NJ)=NX(NI,0,NJ)+1
                NACEPT(NI,0,NJ)=NACEPT(NI,0,NJ)+1
        ELSE
C
                IF(SCH.EQ.1)THEN
                    KK=-1
                ELSEIF(SCH.EQ.2)THEN
                    KK=DAR(NI,NJ,NX,C,TRP,IZ,TDAR,NN)
                ELSEIF(SCH.EQ.3)THEN
                    KK=LRI(NI,NJ,NX,C,TRP,IY,A,RW,NN)
                ELSEIF(SCH.EQ.4)THEN
                    KK=LRP(NI,NJ,NX,C,TRP,IY,A,RW,PN,NN)
                ELSEIF(SCH.EQ.5)THEN
                    KK=AAR(NI,NJ,NX,C,TRP,NN)
                ELSEIF(SCH.EQ.6)THEN
                    KK=RR(NI,NJ,NX,C,TRP,IZ,NN)
                ELSEIF(SCH.EQ.7)THEN
                    KK=LBA(NI,NJ,NX,C,TRP,NN)
                ENDIF
                IF (KK.EQ.-1) THEN
                    NRJECT(NI,NJ)=NRJECT(NI,NJ)+1
                ELSE
                    NX(NI,KK,NJ)=NX(NI,KK,NJ)+1
                    NX(NJ,KK,NI)=NX(NI,KK,NJ)
                    NACEPT(NI,KK,NJ)=NACEPT(NI,KK,NJ)+1
                ENDIF
        ENDIF
        RETURN
        END
C*********************************************************************
C Returns a tandem node using DAR strategy.
C*********************************************************************
        INTEGER FUNCTION DAR(I,J,NX,C,TRP,IZ,TDAR,NN)
        DIMENSION NX(10,0:10,10),N(8)
        INTEGER C(10,10),TRP,TDAR(10,10)
        INTEGER TLOAD,RANODE
        INTEGER IZ
        L=0
        DO 10 K=1,NN
           IF(K.EQ.I.OR.K.EQ.J) GOTO 10
           L=L+1
           N(L)=K
10      CONTINUE
        M=TDAR(I,J)
        K=N(M)
        NXIK=TLOAD(NX,I,K,NN)
        N1=C(I,K)-NXIK
        NXKJ=TLOAD(NX,K,J,NN)
        N2=C(K,J)-NXKJ
        MM=MIN(N1,N2)
        IF (MM.LE.TRP)THEN
            DAR=-1
            TDAR(I,J)=RANODE(NN,IZ)
        ELSE
            DAR=K
        ENDIF
        RETURN
        END
```

```
C*********************************************************************
C Returns a tandem node using LRI strategy.
C*********************************************************************
      FUNCTION LRI(I,J,NX,C,TRP,IZ,A,RW,NN)
      DIMENSION NX(10,0:10,10),N(8),A(8,9,10)
      INTEGER C(10,10),TRP
      INTEGER TLOAD
      INTEGER IZ
C
      L=0
      DO 10 K=1,NN
         IF(K.EQ.I.OR.K.EQ.J) GOTO 10
         L=L+1
         N(L)=K
10    CONTINUE
      M=LANODE(IZ,A,I,J,NN)
      K=N(M)
C
      NXIK=TLOAD(NX,I,K,NN)
      N1=C(I,K)-NXIK
      NXKJ=TLOAD(NX,K,J,NN)
      N2=C(K,J)-NXKJ
      MM=MIN(N1,N2)
      IF (MM.LE.TRP) THEN
C PUNISH:
         LRI=-1
      ELSE
C REWARD:
         LRI=K
         S=0.
         DO 30 IL=1,NN-2
            IF(IL.EQ.M) GOTO 30
            A(IL,I,J)=(1-RW)*A(IL,I,J)
            S=S+A(IL,I,J)
 30      CONTINUE
         A(M,I,J)=1.-S
      ENDIF
      RETURN
      END
C*********************************************************************
C Returns a tandem node using LRP strategy.
C*********************************************************************
      FUNCTION LRP(I,J,NX,C,TRP,IZ,A,RW,PN,NN)
      DIMENSION NX(10,0:10,10),N(8),A(8,9,10)
      INTEGER C(10,10),TRP
      INTEGER TLOAD
      INTEGER IZ
C
      L=0
      DO 10 K=1,NN
         IF(K.EQ.I.OR.K.EQ.J) GOTO 10
         L=L+1
         N(L)=K
10    CONTINUE
      M=LANODE(IZ,A,I,J,NN)
      K=N(M)
C
      NXIK=TLOAD(NX,I,K,NN)
```

```
        N1=C(I,K)-NXIK
        NXKJ=TLOAD(NX,K,J,NN)
        N2=C(K,J)-NXKJ
        MM=MIN(N1,N2)
        IF (MM.LE.TRP)THEN
C PUNISH:
           LRP=-1
           S=0
           DO 3 IL=1,NN-2
              IF(IL.EQ.M) GOTO 3
              A(IL,I,J)=A(IL,I,J)+PN*A(M,I,J)/(NN-3)
              S=S+A(IL,I,J)
 3         CONTINUE
           A(M,I,J)=1.-S
           GOTO 44
        ELSE
C  REWARD:
           LRP=K
           S=0.
           DO 30 IL=1,NN-2
              IF(IL.EQ.M) GOTO 30
              A(IL,I,J)=(1-RW)*A(IL,I,J)
              S=S+A(IL,I,J)
 30        CONTINUE
           A(M,I,J)=1.-S
        ENDIF
 44     RETURN
        END
C*************************************************************************
C Returns a tandem node using AAR strategy.
C*************************************************************************
        INTEGER FUNCTION AAR(I,J,NX,C,TRP,NN)
        DIMENSION NX(10,0:10,10),N(8)
        INTEGER C(10,10),TRP
        INTEGER TLOAD
C
        AAR=-1
        L=0
        DO 10 K=1,NN
           IF(K.EQ.I.OR.K.EQ.J) GOTO 10
           L=L+1
           N(L)=K
10      CONTINUE
        DO 1 M=1,NN-2
           K=N(M)
           NXIK=TLOAD(NX,I,K,NN)
           N1=C(I,K)-NXIK
           NXKJ=TLOAD(NX,K,J,NN)
           N2=C(K,J)-NXKJ
           MM=MIN(N1,N2)
           IF (MM.GT.TRP) THEN
               AAR=K
               GOTO 43
           ENDIF
 1      CONTINUE
        AAR=-1
 43     RETURN
        END
```

```
C*************************************************************************
C Returns a tandem node using RR strategy.
C*************************************************************************
      INTEGER FUNCTION RR(I,J,NX,C,TRP,IZ,NN)
      DIMENSION NX(10,0:10,10),N(8)
      INTEGER C(10,10),TRP
      INTEGER TLOAD,RANODE
      INTEGER IZ
      L=0
      DO 10 K=1,NN
         IF(K.EQ.I.OR.K.EQ.J) GOTO 10
         L=L+1
         N(L)=K
10    CONTINUE
      M=RANODE(NN,IZ)
      K=N(M)
      NXIK=TLOAD(NX,I,K,NN)
      N1=C(I,K)-NXIK
      NXKJ=TLOAD(NX,K,J,NN)
      N2=C(K,J)-NXKJ
      MM=MIN(N1,N2)
      IF (MM.LE.TRP)THEN
          RR=-1
      ELSE
          RR=K
      ENDIF
      RETURN
      END
C*************************************************************************
C Returns a tandem node using LBA strategy.
C*************************************************************************
      INTEGER FUNCTION LBA(I,J,NX,C,TRP,NN)
      DIMENSION NX(10,0:10,10),N(10)
      INTEGER C(10,10),TRP
      INTEGER TLOAD
      DO 10 L=1,NN
         N(L)=0
10    CONTINUE
      DO 20 K=1,NN
        IF(K.EQ.I.OR.K.EQ.J)GOTO 20
      NXIK=TLOAD(NX,I,K,NN)
      N1=C(I,K)-NXIK
      NXKJ=TLOAD(NX,K,J,NN)
      N2=C(K,J)-NXKJ
      N(K)=MIN(N1,N2)
 20   CONTINUE
      MM=MAX(N(1),N(2),N(3),N(4),N(5),N(6),N(7),N(8),N(9),N(10))
      DO 30 K=1,NN
        IF(K.EQ.I.OR.K.EQ.J)GOTO 30
        IF(MM.EQ.N(K))GOTO 40
 30   CONTINUE
 40   IF (MM.LE.TRP)THEN
          LBA=-1
      ELSE
          LBA=K
      ENDIF
      RETURN
      END
```

```
C*********************************************************************
C Selects a random node for LRP or LRI.
C*********************************************************************
      FUNCTION LANODE(IZ,A,IO,ID,NN)
      DIMENSION A(8,9,10), AP(8)
      RAND=UNIF(IZ)
      SUM=0
      DO 1 I=1,NN-2
         SUM=SUM+A(I,IO,ID)
         IF(RAND.LT.SUM) GOTO 3
 1    CONTINUE
      WRITE(8,*)'ERROR'
 3    LANODE=I
      RETURN
      END
C*********************************************************************
C Returns a random node.
C*********************************************************************
      INTEGER FUNCTION RANODE(N,IY)
      DIMENSION DUMMY(8)
      INTEGER IY
      RAND=UNIF(IY)
      I=RAND*(N-2)
      RANODE=I+1
      RETURN
      END
C*********************************************************************
C Reads number of nodes, capacity matrix, traffic matrix and routing
C scheme code number for a nonsymmetric network, from a file.
C*********************************************************************
      SUBROUTINE INPUT(N,NODP,OVF,X,SCH,C,TRF,TRP,UNIT,IREF,RW,PN,SEED)
      INTEGER C(10,10),SCH,TRP
      INTEGER SEED(3)
      DOUBLE PRECISION UNIT
      DIMENSION D(10,10)
      DIMENSION TRF(45)
      CHARACTER *60 TEXT
      READ(8,101)TEXT
      READ(8,*)SCH,OVF,X,UNIT,TRP
      OPEN (UNIT=5, FILE='DATSIMT',STATUS='OLD')
      READ(5,101)TEXT
      READ(5,*)N,RW,PN,SEED(1),SEED(2),SEED(3)
      WRITE(8,*)'N,RW,PN,SEED1,SEED2,SEED3,SCH,OVF,X,UNIT,TRP'
      WRITE(8,*)N,RW,PN,SEED(1),SEED(2),SEED(3),SCH,OVF,X,UNIT,TRP
      READ(5,101)TEXT
 101  FORMAT(A60)
      WRITE(8,*)TEXT
      READ(5,101)TEXT
      WRITE(8,*)TEXT
      NODP=0
      DO 1 I=1,N-1
          NODP=NODP+I
 1    CONTINUE
      DO 2 I=1,N-1
       K=I+1
       READ(5,*)(C(I,J),J=K,N)
       WRITE(8,*)(C(I,J),J=K,N)
 2    CONTINUE
```

```
      DO 30 I=1,N-1
       DO 40 J=I+1,N
         C(J,I)=C(I,J)
 40     CONTINUE
 30    CONTINUE
       DO 3 I=1,N-1
        K=I+1
        READ(5,*)(D(I,J),J=K,N)
        WRITE(8,*)(D(I,J),J=K,N)
 3     CONTINUE
       K=1
       DO 4 I=1,N-1
        DO 5 J=I+1,N
          TRF(K)=OVF*D(I,J)
          K=K+1
 5      CONTINUE
 4     CONTINUE
       RETURN
       END
C*********************************************************************
C This subroutine initializes P, PLA and TDAR.
C P(I) = probability that a call arrives for the Ith O-D pair.
C PLA(K,I,J) = Probability of selecting the Kth tandem node for O-D pair
C (I,J) for LRP or LRI.
C TDAR(I,J) = the current tandem node for O-D pair (I,J) for DAR.
C*********************************************************************
       SUBROUTINE SETUP(N,SCH,NODP,P,II,JJ,PLA,TDAR,IY,TRF,SUMLD)
       DIMENSION PLA(8,9,10),II(45),JJ(45),P(45)
       INTEGER TDAR(10,10)
       DIMENSION TRF(45)
       INTEGER N,NODP,SCH
       INTEGER RANODE
       INTEGER IY
C
       SUMLD=0
       DO 11 I=1,NODP
          SUMLD=SUMLD+TRF(I)
 11    CONTINUE
       WRITE(8,*)'TOTAL TRAFFIC=',SUMLD
C
       P(1)=TRF(1)/SUMLD
       DO 1 I=2,NODP
          P(I)=TRF(I)/SUMLD+P(I-1)
  1    CONTINUE
       IF (P(NODP).NE.0.) THEN
       P(NODP)=1.000
       ENDIF
       K=1
       DO 2 I=1,N-1
        DO 3 J=I+1,N
          II(K)=I
          JJ(K)=J
          K=K+1
 3      CONTINUE
 2     CONTINUE
C  INITIALIZE PLA: SET EQUAL PROBABILITIES
       DO 4 I=1,N-1
       DO 5 J=I+1,N
```

```
      DO 6 K=1,N-2
         PLA(K,I,J)=1./(N-2)
 6    CONTINUE
 5    CONTINUE
 4    CONTINUE
C  INITIALIZE TDAR CHOOSE A RANDOM NODE
         DO 7 I=1,N-1
         DO 8 J=I+1,N
            TDAR(I,J)=RANODE(N,IY)
 8       CONTINUE
 7       CONTINUE
      WRITE(8,59)
 59   FORMAT(2X,'%BP',T15,'  DIRECT   ',T30,'  CARRIED ',T45,' PDC'
     1 ,T60,'  UNIT')
      RETURN
      END
C*********************************************************************
C This subroutine calculates and prints the percentage of the overall
C blocking probability, total calls routed directly, total calls carried
C and proportion of directly routed calls at the end of each time unit.
C*********************************************************************
      SUBROUTINE OUTBP(NACEPT,NRJECT,RN,NN)
      DIMENSION NACEPT(9,0:10,10),NRJECT(9,10)
      INTEGER OFFERD,DIRECT,CARRID,REJECT
C
      N=INT(RN)
      DIRECT=0
      CARRID=0
      REJECT=0
      DO 10 I=1,NN-1
         DO 20 J=I+1,NN
            REJECT=REJECT+NRJECT(I,J)
            NRJECT(I,J)=0
            DIRECT=DIRECT+NACEPT(I,0,J)
            DO 30 K=0,NN
               IF (K.EQ.I.OR.K.EQ.J) GOTO 30
               CARRID=CARRID+NACEPT(I,K,J)
               NACEPT(I,K,J)=0
 30         CONTINUE
 20       CONTINUE
 10   CONTINUE
      OFFERD=CARRID+REJECT
      BP= 100*(REAL(REJECT)/REAL(OFFERD))
      PDC= REAL(DIRECT)/REAL(CARRID)
      WRITE(8,50)BP,DIRECT,CARRID,PDC,N
 50   FORMAT(1X,F8.4,T15,I6,T30,I6,T45,F8.5,T60,I4)
      RETURN
      END
```

Appendix 5

Simulation package II

```
/*********************************************************************/
/*                          graphics Front End                       */
/* This program draws a window on the screen and allows the user to */
/* draw  and edit a network in that window and to enter the         */
/* necessary parameters for simulation. The left and midddle mouse  */
/* buttons provide a menu each and the right mouse button is        */
/* reserved for drawing purposes. To simulate a network, this       */
/* program calls another module named pro.c.                        */
/*********************************************************************/
#define NMAX 10
#define TMAX 45
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/scrollbar.h>
#define RAD   20
#define PI    3.1415927
/**************************/
extern void simulate();
static void my_event_proc();
void number_node();
void num_node_start();
void draw_node();
void circle(),line(),bar();
void draw_line_between_two_nodes();
void draw_link(),write_string();
int  xy_is_near_node();
double sin(),cos(),sqrt();
void init_array();
void move_node();
void delete_node(),delete_link();
void get_traffic();
void get_capacity();
void print_node_connections(),write_link_data();
void load_tandem_link(),set_scheme(),sim_menu();
char *ecvt(), *itoa();
void help(),set_flags_to_zero(),set_flag_to_one();
struct {
        int  num;
        int  x;
        int  y;
        int  con[NMAX];
      } node[NMAX+1];
struct {
       int orig;
       int dest;
       int tandem[NMAX-1];
       int capac;
       float traffic;
       double prob;
       int scheme;
        }link[TMAX+1];
Menu menu;
Menu menu2;
Pixwin *pw;
Frame   frame;
Canvas   canvas;
Event  *event;
int  number_of_OD_pairs;
```

```
int  number_of_nodes=0;
int number_of_node_numbers=0;
/*********************************************************************/
main()
{

init_array();
frame = window_create(NULL, FRAME,
     FRAME_LABEL,   "<<Graphics-front-end>>",
     WIN_X,          500,
     WIN_Y,           0,
     WIN_WIDTH,       600,
     WIN_HEIGHT,      900,
     0);
canvas = window_create(frame, CANVAS,
          CANVAS_AUTO_SHRINK,    FALSE,
          CANVAS_WIDTH,          2000,
          CANVAS_HEIGHT,         1000,
          WIN_CONSUME_KBD_EVENT,
          WIN_ASCII_EVENTS,
          WIN_CONSUME_PICK_EVENT,
          WIN_MOUSE_BUTTONS,
          WIN_EVENT_PROC,     my_event_proc,
          WIN_HORIZONTAL_SCROLLBAR,
          scrollbar_create(SCROLL_PLACEMENT, SCROLL_SOUTH,0),
          0);
menu2=menu_create(MENU_STRINGS,"print node connections",
                  "print link data","simulation info",
                  "LRP","LRI","DAR",
                  "RR","AAR","FR",0,0);
menu=menu_create(MENU_STRINGS,"node","link",
                  "number node","delete node","move node",
                  "delete link","capacity","traffic",
                  "simulate",0,0);
pw=canvas_pixwin(canvas);
window_main_loop(frame);
exit(0);
}

/*********************************************************************/
/* This void detects an event on the graphics window. Two menus are */
/* displayed, one for left mouse button and the other for right      */
/* mouse button events. The user can select options from these       */
/* menus. When an option is selected then this void calls the        */
/* appropriate routine. The right mouse button is for drawing. When */
/* right mouse button event is detected, the current option is       */
/* checked and an appropriate drawing routine is called.             */
/*********************************************************************/
static void   my_event_proc(canvas,event)
Canvas canvas;
Event  *event;
{
static int node_flag=0, link_flag=0,move_flag=0,
         num_flag=0,num_start=0,del_node_flag=0,
         start_link_flag=0,del_link_flag=0,del_link_start=0;
static int seq_num=0,asci_count;
char *string;
static int TRP=0;
```

```
static float unit=10.0,t_inc=1.0;
static float rw_para=.1, pn_para=.1;
static int seed[3]={123456789,100876521,111345703};
string = NULL;
switch (event_id(event))
{
  case MS_LEFT:
    switch((int)menu_show(menu,canvas,event,0))
    {
    case 1:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    set_flag_to_one(&node_flag);
    help("Node"," ");
    break;

    case 2:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    set_flag_to_one(&link_flag);
    help("Link"," ");
    break;

    case 3:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    set_flag_to_one(&num_flag);
    num_start=1;
    help("Press a Node"," ");
    break;

    case 4:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    set_flag_to_one(&del_node_flag);
    help("Delete a Node"," ");
    break;

    case 5:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    move_flag=1;
    help("Press the node","to be moved");
    break;

    case 6:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    set_flag_to_one(&del_link_flag);
    help("delete link"," ");
    break;

    case 7:
    set_flags_to_zero(&node_flag,&link_flag,&num_flag,
    &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
    help("Enter capacity"," ");
    get_capacity();
    load_tandem_link(number_of_nodes,number_of_OD_pairs);
```

```
   break;

   case 8:
   set_flags_to_zero(&node_flag,&link_flag,&num_flag,
   &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
   help("Enter traffic"," ");
   get_traffic();
   break;

   case 9:
   set_flags_to_zero(&node_flag,&link_flag,&num_flag,
   &del_node_flag,&move_flag,&start_link_flag,&del_link_flag);
   help("Simulation"," ");
   number_of_OD_pairs=get_number_of_OD_pairs();
   load_tandem_link(number_of_nodes,number_of_OD_pairs);
   simulate(number_of_nodes,number_of_OD_pairs,TRP,
            unit,t_inc,seed,rw_para,pn_para);
   break;
   /******/
   default :
   break;
   } /* end of switch(menu_show) */

break;

case MS_RIGHT:
{
  if(event_is_down(event))
  {
     if (node_flag==1 && number_of_nodes<NMAX)
       draw_node(event_x(event),event_y(event));
     if (num_flag==1)
     {
          num_node_start(event_x(event),event_y(event),
                         &num_flag,&seq_num);
          num_start=0;
          if(num_flag==1)
          {
             help("Enter Node Number","Press Return");
          }
     }
      if(move_flag==1)
         move_node(event_x(event),event_y(event));
      if (link_flag==1)
         draw_link(event_x(event),event_y(event),&start_link_flag);
      if (del_link_flag==1)
         delete_link(event_x(event),event_y(event),&del_link_start);
     if(del_node_flag==1)
         delete_node(event_x(event),event_y(event));
  }
break;
} /* end case MS_RIGHT */
/********/
case MS_MIDDLE:
   switch((int)menu_show(menu2,canvas,event,0))
     {
       case 1:
          print_node_connections();
```

```
            break;
            case 2:
               write_link_data();
            break;
            case 3:
               sim_menu(&TRP,&unit,&t_inc,&rw_para,&pn_para,seed);
            break;
            case 4:
               set_scheme('4',"LRP");
            break;
            case 5:
               set_scheme('3',"LRI");
            break;
            case 6:
               set_scheme('2',"DAR");
            break;
            case 7:
               set_scheme('1',"RR");
            break;
            case 8:
               set_scheme('5',"AAR");
            break;
            case 9:
               set_scheme('0',"FR");
            break;
            default:
            break;
         }
   break;
   default:
   break;
   }  /* end of switch(id_event) */
   if(event_id(event)>='0' && event_id(event)<='9' && num_flag==1
            && num_start==0)
     {
        window_set(canvas,WIN_IGNORE_PICK_EVENT,WIN_MOUSE_BUTTONS,0);
        number_node(seq_num,event_id(event),asci_count);
        asci_count+=1;
     }
     if(event_id(event)==13)
     {
        window_set(canvas,WIN_CONSUME_PICK_EVENT,WIN_MOUSE_BUTTONS,0);
        asci_count=0;
        help("Press a Node"," ");
     }
}

/*********************************************************************/
void draw_node(x_node,y_node)
int x_node, y_node;
{
int node_sequence_number;

if(xy_is_near_node(&x_node,&y_node,60.0,&node_sequence_number)!=1)
 {
     number_of_nodes+=1;
     node[number_of_nodes].x=x_node;
     node[number_of_nodes].y=y_node;
```

```
      circle(node[number_of_nodes].x,node[number_of_nodes].y,1);
  }
}
/*********************************************************************/
void line(x1,y1,x2,y2,attrib)
int x1,y1,x2,y2,attrib;
{
   pw_vector(pw,x1,y1,x2,y2, PIX_SRC,attrib);
}
/*********************************************************************/
void circle(xc,yc,attribute)
int xc, yc,attribute;
{
int  x1,y1,x2,y2,xd,yd;
int i,x[13],y[13];
double ang;
float r;
ang=0.;
r=(float)RAD;
for(i=0; i<13; ++i)
 {
   x[i]= r*sin(ang);
   y[i]= r*cos(ang);
   ang=ang+PI/6.;
 }
x[0]=x[0]+xc;
y[0]=y[0]+yc;
for(i=1; i<13; ++i)
  {
    x[i]+=xc;
    y[i]+=yc;
    x1=x[i-1];
    y1=y[i-1];
    x2=x[i];
    y2=y[i];
    line(x1,y1,x2,y2,attribute);
  }
}
/*********************************************************************/
void draw_link(x_link,y_link,sline)
int x_link,y_link,*sline;
{
static int  xl[3],yl[3];
int n,i;
static int origin;
int destination;
int sequence_number;
if(xy_is_near_node(&x_link,&y_link,(float)RAD,&n))
  {
    sequence_number=n;
    *sline +=1;
    xl[*sline]=x_link;
    yl[*sline]=y_link;
    switch(*sline)
     {
       case 1:
       origin=sequence_number;
       break;
```

```
        case 2:
         {
          *sline=0;
          destination=sequence_number;
          if(origin!=destination)
           {
             draw_line_between_two_nodes(xl[1],yl[1],xl[2],yl[2],1);
             for(i=1; i<number_of_nodes; ++i)
              {
                if(node[origin].con[i]==destination)
                    break;
                if(node[origin].con[i]==0)
                  {
                    node[origin].con[i]=destination;
                    break;
                  }
              }
             for(i=1; i<number_of_nodes; ++i)
              {
               if(node[destination].con[i]==origin)
                 break;
               if(node[destination].con[i]==0)
                {
                  node[destination].con[i]=origin;
                  break;
                }
              }
           }
         }
         break;
         defult:break;
      }
   }
   else *sline=0;
}
/*********************************************************************/
void delete_link(x_link,y_link,sline)
int x_link,y_link,*sline;
{
static int  xl[3],yl[3];
int n,i;
static int origin;
int destination;
int sequence_number;
if(xy_is_near_node(&x_link,&y_link,(float)RAD,&n))
  {
    sequence_number=n;
    *sline +=1;
    xl[*sline]=x_link;
    yl[*sline]=y_link;
    switch(*sline)
     {
      case 1:
      origin=sequence_number;
      break;
      case 2:
      {
         *sline=0;
```

```
          destination=sequence_number;
          if(origin!=destination)
          {
            draw_line_between_two_nodes(xl[1],yl[1],xl[2],yl[2],0);
            for(i=1; i<number_of_nodes; ++i)
              {
                if(node[origin].con[i]==destination)
                  {
                    node[origin].con[i]=0;
                    break;
                  }
              }
           for(i=1; i<number_of_nodes; ++i)
           {
             if(node[destination].con[i]==origin)
             {
               node[destination].con[i]=0;
               break;
             }
           }
          }
       }
       break;
       defult:break;
      }
  }
  else *sline=0;
}
/*********************************************************************/
void draw_line_between_two_nodes(x1,y1,x2,y2,att)
int x1,y1,x2,y2,att;
{
int x0,y0,dummy;
double r1;
dummy=((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
r1= (double)(RAD)/ sqrt((double) dummy);
x0= r1*(x2-x1);
y0= r1*(y2-y1);
line(x1+x0,y1+y0,x2-x0,y2-y0,att);
}
/*********************************************************************/
xy_is_near_node(x,y,range,node_sequence_number)
int *x, *y, *node_sequence_number;
double range;
{
int i, dummy, flag;
double r1;
flag=0;
*node_sequence_number=0;
for (i=1; i<number_of_nodes+1; ++i)
 {
   dummy=((*x-node[i].x)*(*x-node[i].x)+
         (*y-node[i].y)*(*y-node[i].y));
   r1=sqrt((double) dummy);
   if(r1 <=range)
     {
       flag=1;
      *x=node[i].x;
```

```
        *y=node[i].y;
        *node_sequence_number=i;
        break;
       }
  }
return(flag);
}
/*************************************************************************/
void num_node_start(x,y,num_flag,seq_num)
int x,y,*num_flag,*seq_num;
{
int nn;
if(xy_is_near_node(&x,&y,(float)RAD,&nn))
 {
   *seq_num=nn;
   write_string(x-4,y+4,"| ");
 }
else
   *num_flag=0;
}
/*************************************************************************/
void number_node(seq_number,asci_code,asci_count)
int seq_number,asci_code,asci_count;
{
int node_number;
char c;
char s[3];
static char sd[3];
int x,y;
c=asci_code;
if(asci_count==0)
  {
      number_of_node_numbers+=1;
      s[0]=c;
      s[1]=0;
      sd[0]=s[0];
      x=node[seq_number].x;
      y=node[seq_number].y;
      write_string(x-4,y+4,"  ");
      write_string(x-4,y+4,&s[0]);
      node_number=atoi(s);
      node[seq_number].num=node_number;
  }
if(asci_count==1)
  {
      s[0]=sd[0];
      s[1]=c;
      s[2]=0;
      printf("s=%s\n",&s[0]);
      x=node[seq_number].x;
      y=node[seq_number].y;
      write_string(x-4,y+4,"  ");
      write_string(x-4,y+4,&s[0]);
      node_number=atoi(s);
      node[seq_number].num=node_number;
  }
}
/*************************************************************************/
```

```
atoi(s)
char s[];
{
int i,n;
n=0;
for (i=0; s[i]>='0' && s[i]<='9'; ++i)
     n=10*n + s[i] - '0';
return(n);
}
/*********************************************************************/
void write_string(xs,ys,st)
int xs,ys;
char *st;
{
  pw_text(pw,xs,ys,PIX_SRC,0,st);
}
/*********************************************************************/
void init_array()
{
int i,j;
for (i=0; i<NMAX+1; ++i)
 {
   node[i].num=0;
   node[i].x=0;
   node[i].y=0;
   for(j=0; j<NMAX; ++j)
     node[i].con[j]=0;
 }
 for (i=0; i<TMAX+1; ++i)
   {
     link[i].orig=0;
     link[i].dest=0;
     link[i].capac=0;
     link[i].traffic=0;
     link[i].scheme='4';     /* LRP */
   }
}
/*********************************************************************/
get_number_of_OD_pairs()
{
int i,k,j;
int number_of_OD_pairs=0;
for (i=1; i<number_of_nodes; ++i)
  number_of_OD_pairs+=i;
  k=1;
  for(i=1; i<number_of_nodes; ++i)
    {
      for(j=i+1; j<number_of_nodes+1; ++j)
        {
          link[k].orig=i;
          link[k].dest=j;
          k+=1;
        }
    }
return(number_of_OD_pairs);
}
/*********************************************************************/
void move_node(x,y)
```

```
int x,y;
{
int i;
char *s;
static int x_old, y_old,nn;
static int move_start=0;
if(move_start==0)
 {
   if(xy_is_near_node(&x,&y,(float)RAD,&nn))
    {
      move_start=1;
      x_old=x;
      y_old=y;
      circle(x,y,0);
      write_string(x-4,y+4,"  ");
      for(i=1; i<number_of_nodes; ++i)
        {
          if(node[nn].con[i] !=0)
          draw_line_between_two_nodes(x,y,node[node[nn].con[i]].x,
                                      node[node[nn].con[i]].y,0);
        }
      help("press the mouse on","the destination");
    }
   else
   move_start=0;
   return;
 }
if(move_start==1)
 {
   circle(x,y,1);
   s=itoa((double)node[nn].num);
   s[2]=0;
   write_string(x-4,y+4,s);
   node[nn].x=x;
   node[nn].y=y;
   for(i=1; i<number_of_nodes; ++i)
    {
      if(node[nn].con[i] !=0)
      draw_line_between_two_nodes(x,y,node[node[nn].con[i]].x,
                                  node[node[nn].con[i]].y,1);
    }
   move_start=0;
   help("Press the node","to be moved");
 }
}
/*********************************************************************/
void bar(x_start,y_start,width,att)
int x_start,y_start,width,att;
{
  line(x_start,y_start,x_start+width,y_start,att);
  line(x_start+width,y_start,x_start+width,y_start+width,att);
  line(x_start+width,y_start+width,x_start,y_start+width,att);
  line(x_start,y_start+width,x_start,y_start,att);
}
/*********************************************************************/
void delete_node(x_delete,y_delete)
int x_delete,y_delete;
{
```

```
int del_node;
int i,j,k;
int dummy[NMAX];
int x1,y1,x2,y2;
if(xy_is_near_node(&x_delete,&y_delete,(float)RAD,&del_node))
  {
    circle(x_delete,y_delete,0);
    write_string(node[del_node].x-4,node[del_node].y+4,"  ");
    /**/
    x1=node[del_node].x;
    y1=node[del_node].y;
    for(i=1; i<number_of_nodes; ++i)
      if(node[del_node].con[i]!=0)
        {
          x2=node[node[del_node].con[i]].x;
          y2=node[node[del_node].con[i]].y;
          draw_line_between_two_nodes(x1,y1,x2,y2,0);
        }
      /**/
    node[del_node].num=0;
    node[del_node].x=0;
    node[del_node].y=0;
    for(i=1; i<number_of_nodes; ++i)
      node[del_node].con[i]=0;
      /**/
      for(i=del_node+1; i<number_of_nodes+1; ++i)
        node[i-1]=node[i];
        for(i=1; i<number_of_nodes; ++i)
          {
            k=1;
            for(j=1; j<number_of_nodes; ++j)
              dummy[j]=0;
              for(j=1; j<number_of_nodes; ++j)
                {
                  if(node[i].con[j]==del_node)
                    node[i].con[j]=0;
                  if(node[i].con[j]>del_node)
                    node[i].con[j]-=1;
                  if(node[i].con[j]!=0)
                   {
                     dummy[k]=node[i].con[j];
                     k+=1;
                   }
                }
              for(j=1; j<number_of_nodes; ++j)
                {
                  node[i].con[j]=dummy[j];
                  dummy[j]=0;
                }
          }
        /**/
        node[number_of_nodes].num=0;
        node[number_of_nodes].x=0;
        node[number_of_nodes].y=0;
        for(i=1; i<number_of_nodes; ++i)
          node[number_of_nodes].con[i]=0;
          /**/
          number_of_nodes-=1;
```

```
            number_of_OD_pairs=get_number_of_OD_pairs();
            number_of_node_numbers-=1;
    }
}
/*************************************************************************/
void get_capacity()
{
int i,j,k,m;
number_of_OD_pairs=get_number_of_OD_pairs();
k=1;
printf("\n Enter Link Capacities");
for(i=1; i<number_of_nodes; ++i)
 {
   for(j=i+1; j<number_of_nodes+1; ++j)
    {
      for(m=1; m<number_of_nodes; ++m)
        if(node[i].con[m]==j)
         {
           printf("\n C(%d %d)=",node[i].num,node[j].num);
           scanf("%d",&link[k].capac);
         }
      k+=1;
    }
 }
}
/*************************************************************************/
void get_traffic()
{
int i,j,k,m;
number_of_OD_pairs=get_number_of_OD_pairs();
k=1;
printf("\n Enter Traffic Arrivals");
for(i=1; i<number_of_nodes; ++i)
 {
   for(j=i+1; j<number_of_nodes+1; ++j)
     {
       printf("\n TRF(%d %d)=",node[i].num,node[j].num);
       scanf("%f",&link[k].traffic);
       k+=1;
     }
 }
}
/*************************************************************************/
void load_tandem_link(N,NODP)
int N,NODP;
{
int i,k,j;
for(i=1; i<NODP+1; ++i)
  {
    k=0;
    for(j=1; j<N+1; ++j)
      {
        if(j!=link[i].orig && j!=link[i].dest)
          {
            k+=1;
            link[i].tandem[k]=j;
          }
      }
```

```
  }
}
/*********************************************************************/
void set_scheme(scheme,string)
int scheme;
char *string;
{
int i;
help(string," ");
for (i=1; i<TMAX+1; ++i)
   link[i].scheme=scheme;
}
/*********************************************************************/
void print_node_connections()
{
int i,j;
help("Connections"," ");
for(i=1; i<number_of_nodes+1; ++i)
    {
       printf("node[%d].num=%d\n",i,node[i].num);
       for(j=1; j<number_of_nodes; ++j)
       printf("node[%d].con[%d]=%d\n",i,j,node[node[i].con[j]].num);
    }
}
/*********************************************************************/
char *itoa(Number)
double Number;
{
char *st;
int ndigit,decpt,sign;
ndigit=4;
st=ecvt(Number,ndigit,&decpt,&sign);
*(st+decpt)=0;
return(st);
}
/*********************************************************************/
void help(string1,string2)
char *string1,*string2;
{
write_string(10,15,"                    ");
write_string(10,35,"                    ");
write_string(10,15,string1);
write_string(10,35,string2);
}
/*********************************************************************/
void set_flags_to_zero(flag1,flag2,flag3,flag4,flag5,flag6,flag7)
int *flag1,*flag2,*flag3,*flag4,*flag5, *flag6,*flag7;
{
help(" "," ");
*flag1=0; *flag2=0; *flag3=0; *flag4=0; *flag5=0;
*flag6=0; *flag7=0;
}
/*********************************************************************/
void set_flag_to_one(flag)
int *flag;
{
*flag=1;
}
```

```
/*********************************************************************/
void sim_menu(trunk_res_para,max_time,time_interval,
          rw_para,pn_para,seed)
int *trunk_res_para;
float  *max_time,*time_interval;
float *rw_para, *pn_para;
int seed[3];
{
int option = -1;
int  dummy;
float fdummy;
help(" "," ");
printf("\n 1 - trunk reservation parameter = %d\n",*trunk_res_para);
printf("\n 2 - maximum simulation time =%5.1f\n",*max_time);
printf("\n 3- time between displaying results = %3.1f\n",*time_interval);
printf("\n 4- reward parameter = %6.4f\n",*rw_para);
printf("\n 5- penalty parameter = %6.4f\n",*pn_para);
printf("\n 6- seed[1]=%d seed[2]=%d seed[3]=%d\n",seed[0],seed[1],seed[2]);
printf("\n 0 - No change and exit\n");
while(option!=0)
 {
   printf("\n Enter an option=");
   scanf("%d",&option);
   switch(option)
     {
      case 1 :
         printf("\n Enter trunk reservation parameter=");
         scanf("%d",&dummy);
         *trunk_res_para=dummy;
        break;
      case 2 :
        printf("\n Enter maximum simulation time=");
        scanf("%f",&fdummy);
        *max_time=fdummy;
        break;
      case 3 :
        printf("\n Enter time between displaying results=");
        scanf("%f",&fdummy);
        *time_interval=fdummy;
        break;
      case 4 :
        printf("\n Enter reward parameter=");
        scanf("%f",&fdummy);
        *rw_para=fdummy;
        break;
      case 5 :
        printf("\n Enter penalty parameter=");
        scanf("%f",&fdummy);
        *pn_para=fdummy;
        break;
      case 6 :
        printf("\n Enter first seed value=");
        scanf("%f",&fdummy);
        seed[0]=fdummy;
        printf("\n Enter second seed value=");
        scanf("%f",&fdummy);
        seed[1]=fdummy;
        printf("\n Enter third seed value=");
```

```
        scanf("%f",&fdummy);
        seed[2]=fdummy;
        break;
      defult:
        break;
     }
   }
}
/*********************************************************************/
void write_link_data()
{
int i,j,k;
help(" "," ");
number_of_OD_pairs=get_number_of_OD_pairs();
for(i=1; i<number_of_OD_pairs+1; ++i)
   printf("orig=%d dest=%d capacity=%d traffic=%5.2f scheme=%c\n",
         link[i].orig,link[i].dest,link[i].capac,
         link[i].traffic,link[i].scheme);
   printf("\n");
}
/*********************************************************************/
```

```
/*********************************************************************/
/*                        Simulation Module                          */
/* This module should be called from the program graph.c.            */
/* This module simulates a fully connected network of any size. All */
/* necessary information for simulation is passed to this module by */
/* program graph.c. Simulations can be performed for the following  */
/* routing policies: 1-FR, 2-DAR, 3-LRI, 4-LRP, 5-AAR, 6-RR, 7-LBA. */
/*********************************************************************/

#define MIN(x,y)    (((x) < (y)) ? (x) : (y))
#define MAX(x,y)    (((x) > (y)) ? (x) : (y))
#define FR    '0'
#define RR    '1'
#define DAR   '2'
#define LRI   '3'
#define LRP   '4'
#define AAR   '5'
#define NMAX  10
#define TMAX  45
double rand2();
int ranode(), lanode();
void simulate();
void clear_calls(), output(),set_up();
void outBP();
int generate_call(),total_load_on_ODP();
void route_call();
int direct_is_possible();
void accept_call(),reject_call();
int FR_scheme(),RR_scheme(),DAR_scheme(),LRI_scheme();
int LRP_scheme(),AAR_scheme();
extern  struct{
        int  orig;
        int  dest;
        int tandem[NMAX-1];
        int  capac;
        float traffic;
        double prob;
        int scheme;
           }link[TMAX+1];
/*********************************************************************/
void simulate(num_of_nodes,num_of_OD_pairs,TRP,max_simulation_time,
              t_inc,seed0,rw_para,pn_para)
int num_of_nodes,num_of_OD_pairs,TRP;
float max_simulation_time,t_inc;
int seed0[3];
float rw_para,pn_para;
{
float time_of_call_arrival=0.0, time_to_output_result=1.0;
float inter_arrival_time=0.0;
float sumld=0.;
int c[NMAX+1][NMAX+1];
float P_LA[NMAX-1][NMAX][NMAX+1];
int T_DAR[NMAX][NMAX+1];
int current_calls[NMAX][NMAX+1][NMAX+1];
int nrject[NMAX][NMAX+1],nacept[NMAX][NMAX+1][NMAX+1];
int ODP;
int seed[3];
/**********/
```

```
set_up(P_LA,T_DAR,num_of_OD_pairs,num_of_nodes,&sumld,
       current_calls,nacept,nrject,c,seed,seed0);
start:
  ODP=generate_call(&time_of_call_arrival,&inter_arrival_time,
                    sumld,num_of_OD_pairs,seed);
  if(time_of_call_arrival>=time_to_output_result)
     {
     outBP(nacept,nrject,time_to_output_result,num_of_nodes);
     time_to_output_result + = t_inc;
     }
  if(time_to_output_result>max_simulation_time)
      goto end;
  clear_calls(current_calls,inter_arrival_time,num_of_nodes,seed);
    route_call(num_of_nodes,ODP,current_calls,P_LA,T_DAR,c,
              rw_para,pn_para,nacept,nrject,seed,TRP);
   goto start;
end:
  printf("unit=%f\n",max_simulation_time);
}
/*********************************************************************/
double rand2(SEED)
int  *SEED;
{
  double ra;
  *SEED= *SEED * 65539;
   if(*SEED<0)
      *SEED += 2147483647 +1;
  ra= *SEED * 4.656613e-10;
  return(ra);
}
/*********************************************************************/
ranode(N,seed)
int N;
int seed[3];
{
 double ra;
 int i;
 ra=rand2(&seed[2]);
 i=ra*(N-2);
 return(i+1);
}
/*********************************************************************/
lanode(orig,dest,P_LA,N,seed)
int orig,dest,N;
float P_LA[NMAX-1][NMAX][NMAX+1];
int seed[3];
{
  double ra;
  float sum;
  int i;
  ra=rand2(&seed[1]);
  sum=0;
  for(i=1; i<N-1; ++i)
   {
     sum+=P_LA[i][orig][dest];
     if(ra<sum)
       break;
   }
```

```
 return(i);
}
/*************************************************************************/
int generate_call(tr,sr,sumld,NODP,seed)
  float *tr, *sr;
  float sumld;
  int NODP;
  int seed[3];
{
  int i;
  double log();
  double ra;
  ra=rand2(&seed[0]);
  *sr= -1 * log(ra)/sumld;
  *tr= *tr+ *sr;
  ra = rand2(&seed[0]);
  for (i=1; i<NODP+1; ++i)
       if(ra < link[i].prob)
            break;
  return(i);
}
/*************************************************************************/
void clear_calls(nx,sr,N,seed)
  int nx[NMAX][NMAX+1][NMAX+1],N;
  float sr;
  int seed[3];
{
  int i,j,k,ii=0;
  float p, q;
  double exp();

  for(i=1; i<N; ++i)
  {
       for(j=i+1; j<N+1; ++j)
       {
          for(k=0; k<N+1; ++k)
          {
            if(k==i || k==j )
                goto br;
            if(nx[i][k][j]==0)
                goto br;
              p=exp(-sr * nx[i][k][j]);
              q=1.0;
              ii=0;
con:
              q=q * rand2(&seed[0]);
              if(q>=p)
              {
                 ii=ii+1;
                 goto con;
              }
              nx[i][k][j]=nx[i][k][j] - ii;
              nx[i][k][j]=MAX(nx[i][k][j],0);
br:
          printf("");
        }
   }
}
```

```
}
/*************************************************************************/
int total_load_on_ODP(orig,dest,nx,N)
int orig, dest;
int nx[NMAX][NMAX+1][NMAX+1],N;
{
  int i,j,l;
  int nxij;
   i=MIN(orig,dest);
   j=MAX(orig,dest);
  nxij = nx[i][0][j];
  for(l=i+1; l<N+1; ++l)
    {
    if(l != j)
        nxij+= nx[i][j][l];
    }

  for(l=1; l<j; ++l)
  {
    if(l != i)
        nxij+= nx[l][i][j];
  }

  for(l=1; l<i; ++l)
        nxij+= nx[l][j][i];

  for(l=j+1; l<N+1; ++l)
        nxij+= nx[j][i][l];
  return(nxij);
  }
/*************************************************************************/
void route_call(N,ODP,nx,P_LA,T_DAR,c,
                rw_para,pn_para,nacept,nrject,seed,TRP)
int nx[NMAX][NMAX+1][NMAX+1],N,ODP;
float P_LA[NMAX-1][NMAX][NMAX+1];
int T_DAR[NMAX][NMAX+1];
int c[NMAX+1][NMAX+1];
float rw_para, pn_para;
int nrject[NMAX][NMAX+1],nacept[NMAX][NMAX+1][NMAX+1];
int seed[3];
int TRP;
{
 int ni,nj,tnode,nxij;
  ni=link[ODP].orig;
  nj=link[ODP].dest;
  if(direct_is_possible(ni,nj,nx,N,c))
        accept_call(ni,nj,0,nx,nacept);
  else
    {
      switch(link[ODP].scheme)
        {
          case FR:
            tnode= -1;
          break;
          case RR:
            tnode=RR_scheme(N,ODP,c,nx,seed,TRP);
          break;
          case DAR:
```

```
            tnode=DAR_scheme(N,ODP,c,nx,T_DAR,seed,TRP);
          break;
          case LRI:
            tnode=LRI_scheme(N,ODP,c,nx,rw_para,P_LA,seed,TRP);
          break;
          case LRP:
            tnode=LRP_scheme(N,ODP,c,nx,rw_para,pn_para,P_LA,seed,TRP);
          break;
          case AAR:
            tnode=AAR_scheme(N,ODP,c,nx,TRP);
          break;
          defult:
          break;
        }
      if(tnode==-1)
          reject_call(ni,nj,nrject);
      else
          accept_call(ni,nj,tnode,nx,nacept);
    }
 }
/*********************************************************************/
int direct_is_possible(ni,nj,nx,N,c)
int ni,nj, nx[NMAX][NMAX+1][NMAX+1],N;
int c[NMAX+1][NMAX+1];
{
  int nxij,flag=0;
  nxij=total_load_on_ODP(ni,nj,nx,N);
  if(nxij<c[ni][nj])
     flag=1;
  return(flag);
}
/*********************************************************************/
void accept_call(ni,nj,tandem,nx,nacept)
int ni,nj,tandem;
int nx[NMAX][NMAX+1][NMAX+1],nacept[NMAX][NMAX+1][NMAX+1];
{
  nx[ni][tandem][nj]+=1;
  nacept[ni][tandem][nj]+=1;
}
/*********************************************************************/
void reject_call(ni,nj,nrject)
int ni,nj,nrject[NMAX][NMAX+1];
{
  nrject[ni][nj]+=1;
}
/*********************************************************************/
void set_up(P_LA,T_DAR,NODP,N,sumld,nx,nacept,nrject,c,seed,seed0)
int NODP,N;
float P_LA[NMAX-1][NMAX][NMAX+1];
int T_DAR[NMAX][NMAX+1];
float *sumld;
int nx[NMAX][NMAX+1][NMAX+1], nrject[NMAX][NMAX+1];
int nacept[NMAX][NMAX+1][NMAX+1], c[NMAX+1][NMAX+1];
int seed[3],seed0[3];
{
int i,k,j,ni,nj;
for(i=0; i< 3; ++i)
  seed[i]=seed0[i];
```

```
/***initialize pla for LRP and LRI***/
  for (i=1; i<NODP+1; ++i)
    for(k=1; k<N-1; ++k)
       {
        ni=link[i].orig;
        nj=link[i].dest;
       P_LA[k][ni][nj]=1.0/((float)(N-2));
       }
/*** initialize Tandem for DAR ***/
    for (i=1; i<NODP+1; ++i)
       {
        ni=link[i].orig;
        nj=link[i].dest;
       T_DAR[ni][nj]=ranode(N,seed);
       }
/*** total load ***/
  for(i=1; i<NODP+1; ++i)
       *sumld= *sumld+link[i].traffic;
  printf("total traffic =%10.2f\n",*sumld);

/*** prob of each OD pair ***/
  link[1].prob=link[1].traffic/ *sumld;
  for(i=2; i<NODP+1; ++i)
     {
      link[i].prob=link[i].traffic/ *sumld +link[i-1].prob;
      }
/***** initialize nx, nrject, nacept *********************/
for(i=1; i<N; ++i)
  {
    for(j=i+1; j<N+1; ++j)
      {
        nrject[i][j]=0;
        for(k=0; k<N+1; ++k)
          if(k!=i && k!=j)
            {
              nacept[i][k][j]=0;
              nx[i][k][j]=0;
            }
      }
  }
/************ load capacity matrix ************************/
for(i=1; i<NODP+1; ++i)
   {
        ni=link[i].orig;
        nj=link[i].dest;
      c[ni][nj]=link[i].capac;
      c[nj][ni]=c[ni][nj];
   }
}
/*********************************************************************/
 int RR_scheme(N,ODP,c,nx,seed,TRP)
  int N,ODP,TRP;
  int c[NMAX+1][NMAX+1], nx[NMAX][NMAX+1][NMAX+1];
  int seed[3];
  {
  int l,tnode;
  int ni,nj,c1,c2,cmin;
   ni=link[ODP].orig;
```

```
    nj=link[ODP].dest;
    l=ranode(N,seed);
    tnode = link[ODP].tandem[l];
    c1=c[ni][tnode]- total_load_on_ODP(ni,tnode,nx,N);
    c2=c[tnode][nj]- total_load_on_ODP(tnode,nj,nx,N);
     cmin=MIN(c1,c2);
     if(cmin<=TRP)
        tnode = -1;
     return(tnode);
   }
/*********************************************************************/
int LRI_scheme(N,ODP,c,nx,rw_para,P_LA,seed,TRP)
 int N,ODP,TRP;
 int c[NMAX+1][NMAX+1], nx[NMAX][NMAX+1][NMAX+1];
 float P_LA[NMAX-1][NMAX][NMAX+1];
 float rw_para;
 int seed[3];
 {
   int l,tnode,c1,c2,cmin,ni,nj,i;
   float sum;
   ni=link[ODP].orig;
   nj=link[ODP].dest;
      l=lanode(ni,nj,P_LA,N,seed);
      tnode = link[ODP].tandem[l];
          c1=c[ni][tnode]- total_load_on_ODP(ni,tnode,nx,N);
          c2=c[tnode][nj]- total_load_on_ODP(tnode,nj,nx,N);
          cmin=MIN(c1,c2);
         if(cmin<=TRP)
             tnode = -1;
         else
           {
             sum=0.;
               for(i=1; i<N-1; ++i)
                if(i != l)
                  {
                     P_LA[i][ni][nj]*=(1-rw_para);
                     sum+=P_LA[i][ni][nj];
                  }
                P_LA[l][ni][nj]=1.0-sum;
           }
  return(tnode);
 }
/*********************************************************************/
int LRP_scheme(N,ODP,c,nx,rw_para,pn_para,P_LA,seed,TRP)
 int N,ODP,TRP;
 int c[NMAX+1][NMAX+1], nx[NMAX][NMAX+1][NMAX+1];
 float P_LA[NMAX-1][NMAX][NMAX+1];
 float rw_para,pn_para;
 int seed[3];
 {
  int l,tnode,c1,c2,cmin,ni,nj,i;
  float sum;
  ni=link[ODP].orig;
  nj=link[ODP].dest;
  l=lanode(ni,nj,P_LA,N,seed);
  tnode = link[ODP].tandem[l];
  c1=c[ni][tnode]- total_load_on_ODP(ni,tnode,nx,N);
  c2=c[tnode][nj]- total_load_on_ODP(tnode,nj,nx,N);
```

```
   cmin=MIN(c1,c2);
   if(cmin<=TRP)
     {
       tnode = -1;
       sum=0.;
       for(i=1; i<N-1; ++i)
       if(i != l)
        {
          P_LA[i][ni][nj]+=P_LA[l][ni][nj]*pn_para/(N-3);
          sum+=P_LA[i][ni][nj];
        }
          P_LA[l][ni][nj]=1.0-sum;
     }
   else
     {
       sum=0.;
       for(i=1; i<N-1; ++i)
         if(i != l)
           {
             P_LA[i][ni][nj]*=(1-rw_para);
             sum+=P_LA[i][ni][nj];
           }
             P_LA[l][ni][nj]=1.0-sum;
     }
  return(tnode);
  }
/*********************************************************************/
int DAR_scheme(N,ODP,c,nx,T_DAR,seed,TRP)
 int N,ODP,TRP;
 int c[NMAX+1][NMAX+1], nx[NMAX][NMAX+1][NMAX+1];
 int T_DAR[NMAX][NMAX+1];
 int seed[3];
 {
  int l,tnode,c1,c2,cmin,ni,nj;
  ni=link[ODP].orig;
  nj=link[ODP].dest;
  l=T_DAR[ni][nj];
  tnode = link[ODP].tandem[l];
  c1=c[ni][tnode]- total_load_on_ODP(ni,tnode,nx,N);
  c2=c[tnode][nj]- total_load_on_ODP(tnode,nj,nx,N);
  cmin=MIN(c1,c2);
    if(cmin<=TRP)
      {
        tnode = -1;
        T_DAR[ni][nj]=ranode(N,seed);
      }
  return(tnode);
 }
/*********************************************************************/
int AAR_scheme(N,ODP,c,nx,TRP)
 int N,ODP,TRP;
 int c[NMAX+1][NMAX+1], nx[NMAX][NMAX+1][NMAX+1];
{
  int l,tnode,c1,c2,cmin,ni,nj,nxij;
  int i;
  ni=link[ODP].orig;
  nj=link[ODP].dest;
  for(i=1; i<N-1; ++i)
```

```
    {
      tnode = link[ODP].tandem[i];
      c1=c[ni][tnode]- total_load_on_ODP(ni,tnode,nx,N);
      c2=c[tnode][nj]- total_load_on_ODP(tnode,nj,nx,N);
      cmin=MIN(c1,c2);
      if(cmin>TRP)
         break;
    }
   if(cmin<=TRP)
         tnode= -1;
   return(tnode);
}
/*********************************************************************/
void outBP(nacept,nrject,rn,N)
int nacept[NMAX][NMAX+1][NMAX+1],nrject[NMAX][NMAX+1],N;
float rn;
{
int time;
int i,j,k;
int total_directly_routed=0, total_carried_load=0, total_rejected=0;
float overall_blocking_probability;
time=(int)rn;
 for(i=1; i<N; ++i)
    {
      for(j=i+1; j<N+1; ++j)
        {
          total_directly_routed+=nacept[i][0][j];
          total_rejected+=nrject[i][j];
          nrject[i][j]=0;
          for(k=0; k<N+1; ++k)
             if(k!=i && k!=j)
               {
                  total_carried_load+=nacept[i][k][j];
                  nacept[i][k][j]=0;
               }
        }
    }
 overall_blocking_probability=100.0*((float)total_rejected)/
 ((float)total_rejected+(float)total_carried_load);
 printf("BP[%d]=%f\n",time,overall_blocking_probability);
}
```

Papers

1 - N.Eshragh and P.Mars, 'Performance evaluation of decentralized routing strategies in circuit-switched networks.', Fourth UK Teletraffic Symposium, Bristol, May 1987.

2 - N.Eshragh and P.Mars, 'Study of dynamic routing strategies in circuit-switched networks.', Third UK computer and Telecomunication Performance Engineering Workshop, Edinburgh, Sept 1987.

Performance Evaluation of Decentralized Routing Strategies in Circuit Switched Networks

N.Eshragh and P.Mars

School of Engineering and Applied Science

University of Durham

Abstract

The paper considers a comparative evaluation of decentralized dynamic routing strategies in circuit switched networks. Simulation results are presented for simple network configurations deliberately designed to force dynamic routing, using learning algorithm and the DAR algorithm. It is demonstrated that the learning algorithm provides superior performance characteristics.

1. Introduction

In a previous paper [1] simulation studies of telephone traffic routing in simple networks was considered. Specifically it was shown that a Linear Reward Inaction (L_{RI}) automaton scheme, when used in a simple network for call routing, performs at least as well as the optimum fixed rule (FR). The L_{RI} and Linear Reward Penalty (L_{RP}) schemes were compared to FR. It was concluded that both routing strategies always perform as well as the optimum FR while in simulations requiring mixed routing strategies they give superior performance.

A recent report [2] has investigated dynamic routing of fully connected circuit switched networks. The routing policy used is Least Busy Alternative (LBA) with Trunk Reservation (TR). It was concluded that LBA with TR is as good as FR but with the advantage of flexibility and spreading out of local overload. Subsequent research has critically compared LBA with Random Routing (RR), Fixed Routing (FR), Learning Automata and Dynamic Alternative Routing (DAR) [3] for a five node fully connected network. It has been shown that DAR and Learning Automata algorithms provide the best dynamic routing strategies [4]. In this paper the performance of the Learning Automata is compared by simulation with DAR. The algorithms are simulated on small networks with link capacities and traffic patterns chosen to force dynamic routing.

2. Routing Strategies

For all of the follwing routing policies except FR, calls are directly routed if possible, and if not an alternative is examined according to the policy of the implemented routing algorithm. Fig.1 shows what it means by direct and alternatives for a five node network.

Fixed Routing (FR)

Calls are routed directly and if blocked no alternative path is tried.

Least Busy Alternative (LBA)

This scheme examines all alternatives and selects the one with the most free links.

Random Routing (RR)

In this scheme all alternatives have equal probability P_i of being selected and $\sum_i P_i = 1$.

Dynamic Alternative Routing (DAR)

This scheme starts on a probabilistic basis: alternatives have equal probability

P_i and $\sum_i P_i = 1$ (The same as RR). When a tandem node is chosen and results in call completion, the node will be selected deterministically for the routing of the next call. If the call is blocked a random tandem node will again be selected.

Linear Reward Penalty (L_{RP})

In this scheme every alternative route is considered as an action α. At every stage the probability of choosing the ith action α_i is $P_i(n)$ and $\sum_i P_i(n) = 1$. When a particular action α_i is chosen and results in call completion the probabilities are updated as follows:

$$P_i(n+1) = P_i(n) + \sum_{j \neq i} (1 - A) P_j(n) \qquad 0.0 < A < 1.0$$

$$P_j(n+1) = A P_j(n)$$

When the call is blocked:

$$P_i(n+1) = B P_i(n) \qquad 0.0 < B < 1.0$$

$$P_j(n+1) = P_j(n) + \frac{(1-B)P_i(n)}{m}$$

Where m=number of actions - 1

A = Learning parameter

B = Penalty parameter

The next action is chosen using the modified probability distribution at stage (n+1).

Linear Reward Inaction (L_{RI})

This is a special case of L_{RP} with reward and no penalty. In this scheme every alternative path is considered as an action α. At every stage the probability of choosing the ith action α_i is $P_i(n)$ and $\sum_i P_i(n) = 1$. When a particular action α_i is chosen and results in call completion, the probabilities are updated as follows:

$$P_i(n+1) = P_i(n) + \sum_{j \neq i} (1 - A) P_j(n) \qquad 0.0 < A < 1.0$$

$$P_j(n+1) = A P_j(n)$$

Where A is the learning parameter.

When the call is blocked the probabilities remain unchanged.

Recent work [4] has critically compared the above first five routing strategies for a five node fully connected network. Under dynamic conditions the DAR and learning automata were shown to provide superior performance. The present paper further compares DAR and the L_{RI} algorithm for simple networks designed to force dynamic routing. L_{RI} is selected because it outperforms L_{RP} for the cases studied.

3. Experimental Results

Simulations were carried out for four and five node networks as shown in figures 2, 3 and 4. Network 1 is a four node network; network 2 is a five node fully connected network used elsewhere [5] and network 3 is a five node with links 1-3 and 3-5 being failed. Calls between nodes i and j are described by a Poisson process and exponential holding time. For each experiment the network simulator was run to equilibrium and then the total network blocking probability was measured defined as:

$$BP = \frac{\text{Total number of blocked calls}}{\text{Total number of offered calls}}$$

3.1. Tuning Dynamic Algorithms

Some dynamic algorithms have parameters that can influence their performance significantly. For example the time between updates for delta routing [5]. The performance of the L_{RI} scheme can be improved by properly tuning the learning parameter. Fig.5 shows the influence of the learning parameter on the performance of the algorithm. The test network and its traffic input are shown in the same figure.

From Fig.5 it can be seen that the blocking probability is very high for A < .99 and drops significantly as A increases with the minimum BP at A=.999. For A > .999 the blocking probability increases until A=1.0 which is actually RR (Random Routing).

The reason for the poor performance with A < .99 is that the algorithm converges to the selection of the path with a higher link capacity, path 1-2-3 (Fig.5) and as a result the available capacity of the other path, 1-4-3 will not be utilized. In other words when blocking probabilities are coarsely adjusted the algorithm locks on one path. The results presented in this paper were obtained for the optimized algorithm, i.e. where the value of learning parameter was selected to give the best performance for the problem studied.

3.2. Comparison between DAR and L_{RI}

Figs 6-8 compare the performance of the two algorithms for network 1 with a single traffic input(λ_{13}). Three sets of link capacities:

$C_{12} = C_{23} = 50, \quad C_{14} = C_{34} = 30$

$C_{12} = C_{23} = 60, \quad C_{14} = C_{34} = 20$

$C_{12} = C_{23} = 40, \quad C_{14} = C_{34} = 40$

have been considered with the total useful capacity being constant and λ_{13} varing over a range from about 30 to 100.

It can be seen that L_{RI} is superior to DAR for the intermediate range of traffic. When traffic is small the blocking probability is small for both schemes because there is sufficient capacity available. When traffic is high a large proportion of calls will be lost and the two schemes utilize the available capacity. For intermediate range of λ the routing algorithms play a more important role because the traffic should correctly be divided between the two paths. The superiority of L_{RI} over this range suggests that it has a better spread of traffic than DAR.

A Comparison of Figs 6-8 shows that when the ratio of the effective capacity of the upper to lower path increases the two algorithms perform closer. This reflects the fact that DAR performs better the higher the ratio of the link capacities of the two alternate paths. Under theses conditions DAR requires less switching between alternate paths.

The next series of experiments were carried out for network 2 with two base traffic matrices given in tables 1-2. The efficiency of the algorithms were investigated under a uniform scaling of the base traffic matrix by a scale factor OVL. For each value of OVL the network simulator was run to equilibrium and then the total blocking probability was measured. The results are shown in Figs 9-10 which illusttrate an improved performance with L_{RI}.

Finally simulations were performed for network 3 with the following base traffic matrix: $\lambda_{13} = 10$, $\lambda_{35} = 10$. The results are given in Fig.11 which shows a significant superiority of L_{RI} over DAR. L_{RI} performs better for two reasons, first it reduces the probability of selecting the failed links to zero i.e it learns not to attempt them, while DAR does not detect the failure and at the time of switching from one path to another, attempts the failed link and looses some calls. Second the traffic must be equally split between two possible paths which exists for each traffic source, eg paths 1-2-3 and 1-4-3 for the node pair 1-3. Under these conditions L_{RI} looses less calls than DAR when switching from one path to another.

4. Conclusions

The effect of the learning parameter on the performance of a simple four node network was shown. As the curve in fig.5 indicates the algorithm can be tuned for an optimal performance. Simulation results for network 1 shows that as the ratio of the effective capacity of the upper to lower path increases, the advantages of the L_{RI} scheme are reduced. In the case of DAR a high ratio of link capacities on the two

alternative paths is a favorable condition. Under these circumstances less switching will be required between the alternative paths for the DAR algorithm. It should be recalled that each time the DAR algorithm switches, a call is lost. Clearly as the ratio of the capacity of the two alternative paths approaches unity, then increasing switching will be required and therefore more calls will be lost.

Results on five node networks deliberately designed to force dynamic routing illustrated an improved performance with L_{RI}. Finally the failure condition experiment showed a significant superiority of L_{RI} over DAR for the case studied.

References

[1] K.S. Narendra and P. Mars "The Use of Learning Algorithms in Telephone Traffic Routing-A Methodology", Automatica, Vol.19, No.5, pp.495-502, 1983.

[2] W-Y NG, "Dynamic Routing in Circuit-Switched Networks: Simulation of the Routing Rule of Least Busy Alternative with Trunk Reservation", Cambridge University, Project Report, Feb.1985.

[3] R.J. Gibbens "Some Aspects of Dynamic Routing in Circuit-Switched Telecommunication Networks", Rayleigh Prize Essay (1986) University of Cambridge.

[4] N. Eshragh, "Simulation studies of telephone traffic routing: comparison of different routing techniques with Learning Automata", Durham University, Project Report, July 1986.

[5] A.Girard and S.Hurtubise, "Dynamic Routing and Call Repacking in Circuit-Switched Networks", IEEE Transactions on Communications, VOL.COM-31, NO.12, Dec 1983.

0.0	15.0	15.0	15.0	30.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	0.0

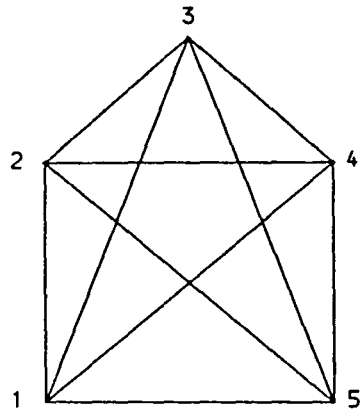
Table 1. Traffic matrix 1.

0.0	15.7	7.7	12.5	2.5
0.0	0.0	18.0	7.3	1.9
0.0	0.0	0.0	10.7	.9
0.0	0.0	0.0	0.0	1.2
0.0	0.0	0.0	0.0	0.0

Table 2. Traffic matrix 2.

List of Symbols

A	Learning parameter
B	Penalty parameter
BP	Blocking Probability
C_{ij}	Link capacity of the node pair (i,j)
DAR	Dynamic Alternative Routing
FL	Free Links
FR	Fixed Routing
LBA	Least Busy Alternative
L_{R-I}	Linear Reward Inaction
L_{R-P}	Linear Reward Penalty
OPT	Optimum
OVL	Overload factor
RR	Random Routing
λ_{ij}	Traffic intensity of the node pair (i,j) in call per unit time



CALL FROM 2 TO 4

DIRECT PATH 2-4

ALTERNATIVE 1 2-1-4

ALTERNATIVE 2 2-3-4

ALTERNATIVE 3 2-5-4

Fig. 1. Direct and Alternative paths for the node pair (2, 4).

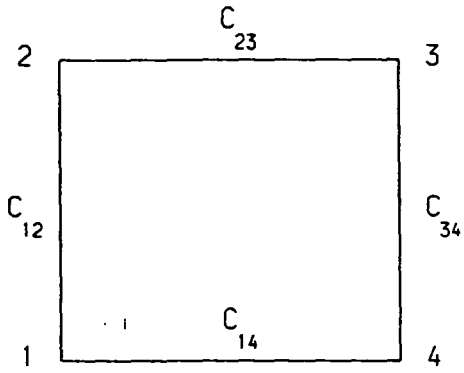


Fig. 2. Network 1: Four node

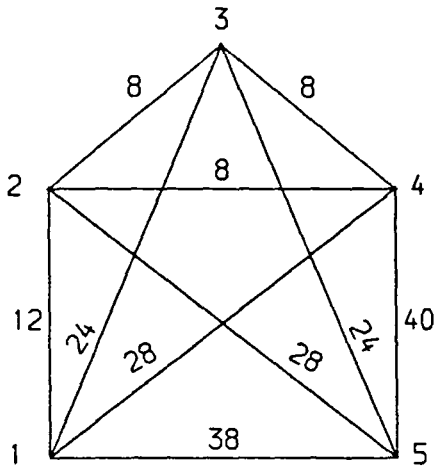


Fig. 3. Network 2: Five node

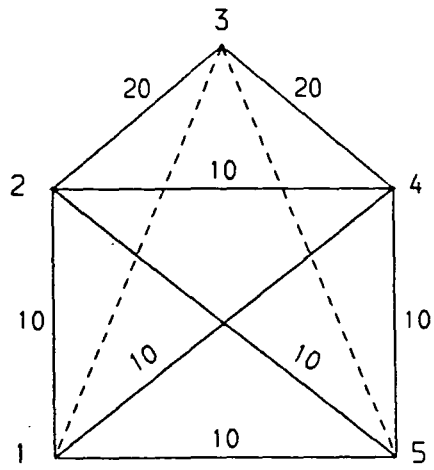


Fig. 4. Network 3: Five node

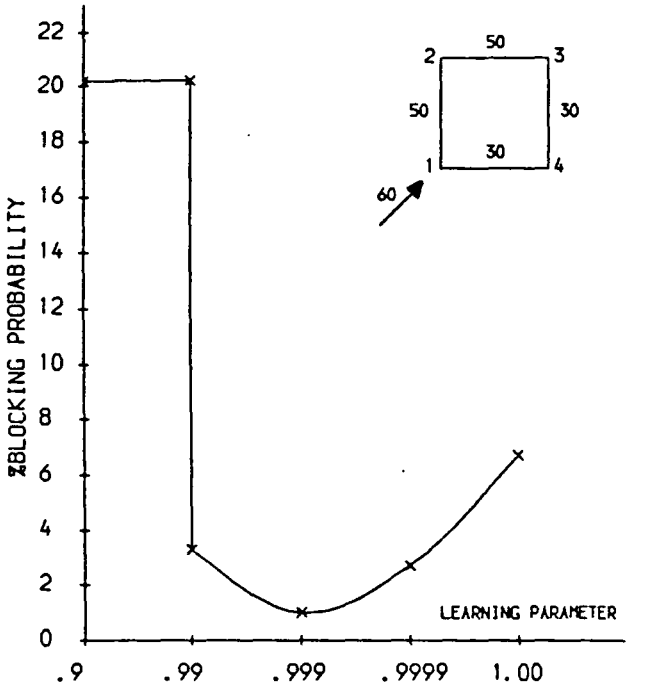


Fig.5 Effect of the Learning parameter on the performance of the LRI scheme.

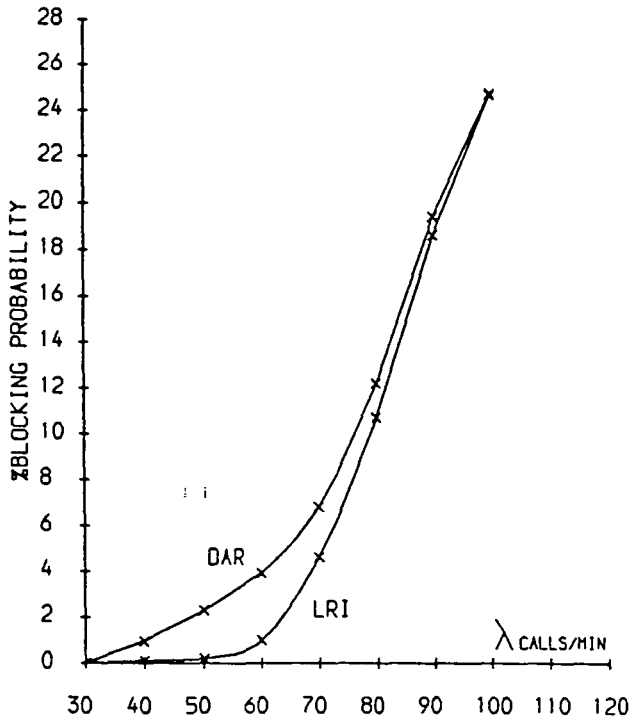


FIG. 6 Comparison between LRI and DAR. Network 1, link capacity ratio 50/30.

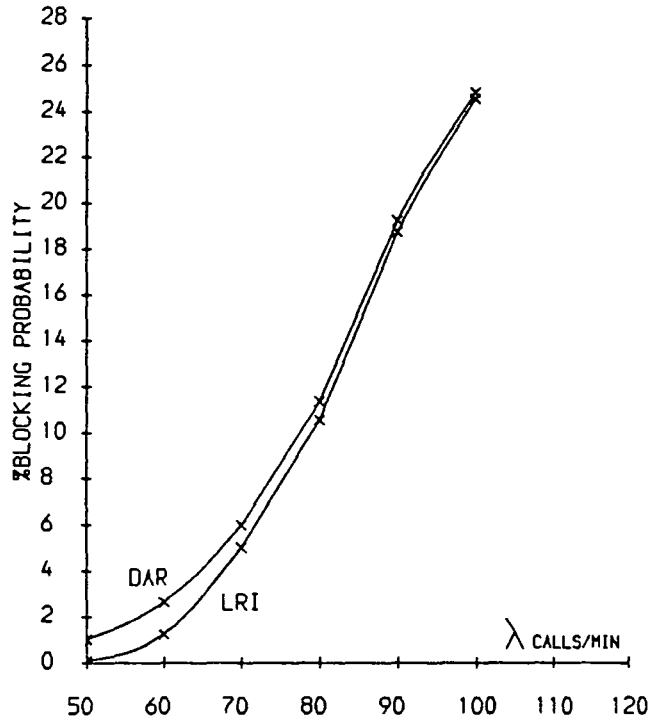


FIG. 7 Comparison between LRI and DAR. Network 1, link capacity ratio 60/20.

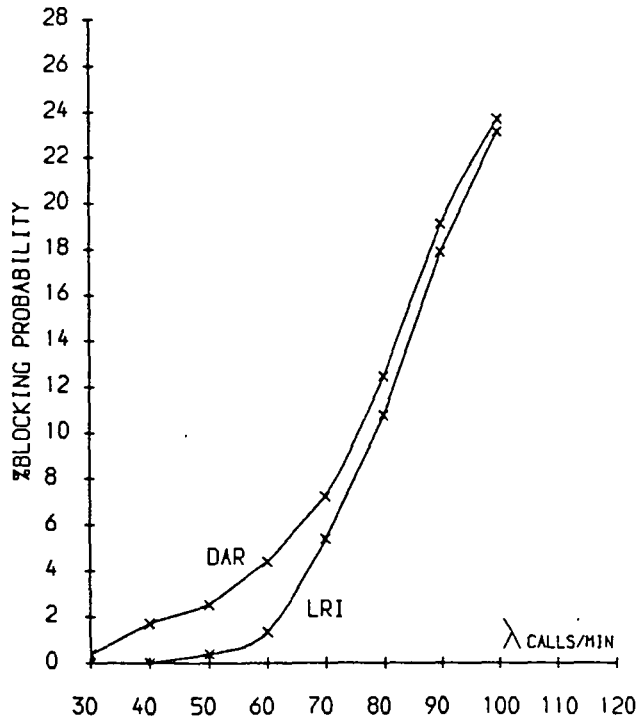


FIG. 8 Comparison between LRI and DAR. Network 1, link capacity ratio=40/40.

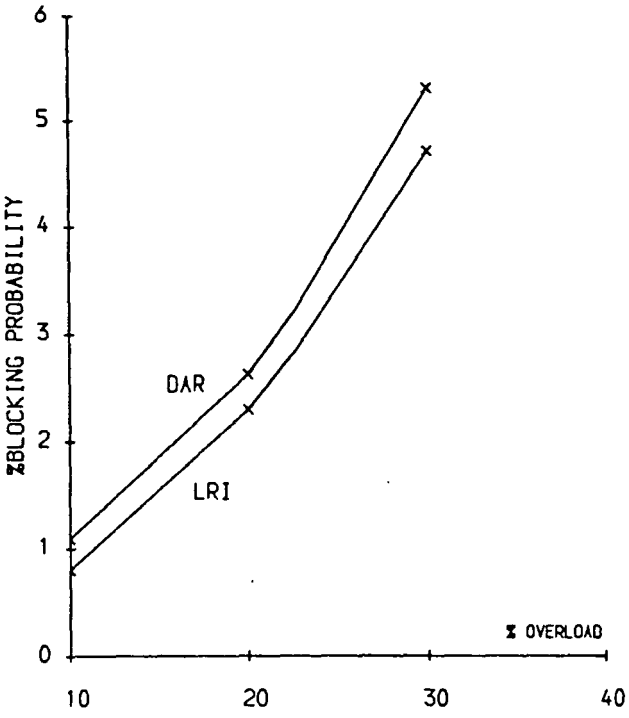


Fig. 9 Comparison between LRI and DAR. Network2, 2 traffic 1.

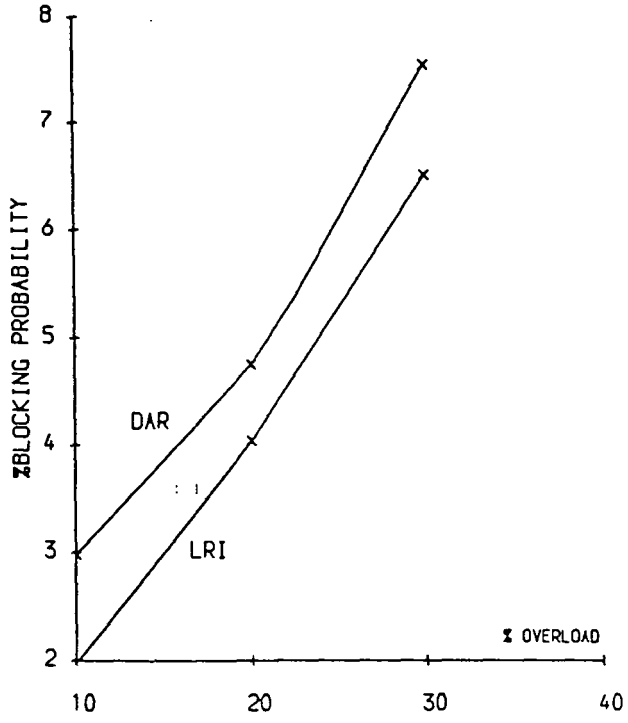


Fig. 10 Comparison between LRI and DAR. Network 2, traffic 2

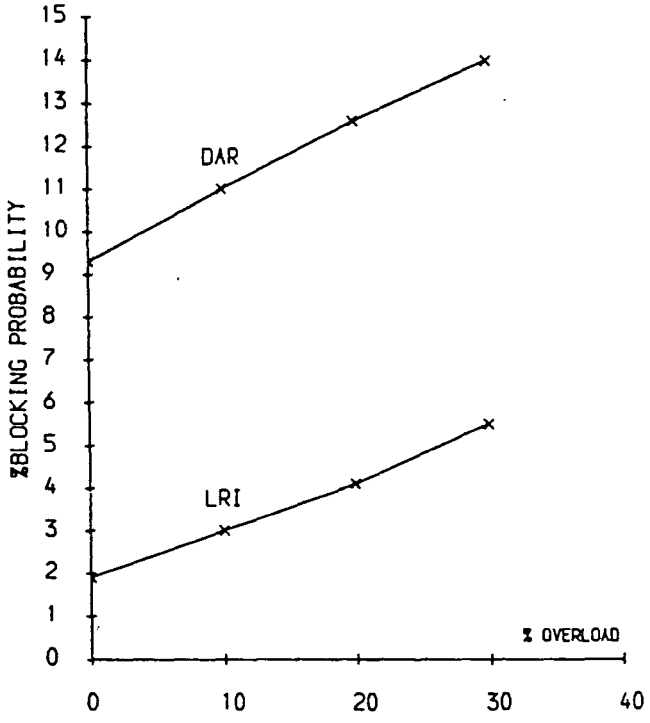


Fig. 11 Comparison of LRI and DAR under failure conditions.

Study of Dynamic Routing Strategies in Circuit Switched Networks

N.Eshragh and P.Mars

School of Engineering and Applied Science

University of Durham

South Road

Durham DH1 3LE

Abstract

The paper considers a comparative evaluation of decentralized dynamic routing strategies in circuit switched networks. Simulation results are presented for simple network configurations deliberately designed to force dynamic routing, using learning algorithm and the DAR algorithm. It is demonstrated that the learning algorithm provides superior performance characteristics.

1. Introduction

In a previous paper [1] simulation studies of telephone traffic routing in simple networks was considered. Specifically it was shown that a Linear Reward Inaction (L_{RI}) automaton scheme, when used in a simple network for call routing, performs at least as well as the optimum fixed rule (FR). The L_{RI} and Linear Reward Penalty (L_{RP}) schemes were compared to FR. It was concluded that both routing strategies always perform as well as the optimum FR while in simulations requiring mixed routing strategies they give superior performance.

A recent report [2] has investigated dynamic routing of fully connected circuit switched networks. The routing policy used is Least Busy Alternative (LBA) with Trunk Reservation (TR). It was concluded that LBA with TR is as good as FR but with the advantage of flexibility and spreading out of local overload. Subsequent research has critically compared LBA with Random Routing (RR), Fixed Routing (FR), Learning Automata and Dynamic Alternative Routing (DAR) [3] for a five node fully connected network. It has been shown that DAR and Learning Automata algorithms provide the best dynamic routing strategies [4]. In this paper the performance of the Learning Automata is compared by simulation with DAR. The algorithms are simulated on small networks with link capacities and traffic patterns chosen to force dynamic routing.

2. Routing Strategies

For all of the follwing routing policies except FR, calls are directly routed if possible, and if not an alternative is examined according to the policy of the implemented routing algorithm. Fig.1 shows what it means by direct and alternatives for a five node network.

Fixed Routing (FR)

Calls are routed directly and if blocked no alternative path is tried.

Least Busy Alternative (LBA)

This scheme examines all alternatives and selects the one with the most free links.

Random Routing (RR)

In this scheme all alternatives have equal probability P_i of being selected and $\sum_i P_i = 1$.

Dynamic Alternative Routing (DAR)

This scheme starts on a probabilistic basis: alternatives have equal probability

P_i and $\sum_i P_i = 1$ (The same as RR). When a tandem node is chosen and results in call completion, the node will be selected deterministically for the routing of the next call. If the call is blocked a random tandem node will again be selected.

Linear Reward Penalty (L_{RP})

In this scheme every alternative route is considered as an action α. At every stage the probability of choosing the ith action α_i is $P_i(n)$ and $\sum_i P_i(n) = 1$. When a particular action α_i is chosen and results in call completion the probabilities are updated as follows:

$$P_i(n+1) = P_i(n) + \sum_{j \neq i}(1 - A)P_j(n) \qquad 0.0 < A < 1.0$$

$$P_j(n+1) = AP_j(n)$$

When the call is blocked:

$$P_i(n+1) = BP_i(n) \qquad 0.0 < B < 1.0$$

$$P_j(n+1) = P_j(n) + \frac{(1-B)P_i(n)}{m}$$

Where m=number of actions - 1

A = Learning parameter

B = Penalty parameter

The next action is chosen using the modified probability distribution at stage (n+1).

Linear Reward Inaction (L_{RI})

This is a special case of L_{RP} with reward and no penalty. In this scheme every alternative path is considered as an action α. At every stage the probability of choosing the ith action α_i is $P_i(n)$ and $\sum_i P_i(n) = 1$. When a particular action α_i is chosen and results in call completion, the probabilities are updated as follows:

$$P_i(n+1) = P_i(n) + \sum_{j \neq i}(1 - A)P_j(n) \qquad 0.0 < A < 1.0$$

$$P_j(n+1) = AP_j(n)$$

Where A is the learning parameter.

When the call is blocked the probabilities remain unchanged.

Recent work [4] has critically compared the above first five routing strategies for a five node fully connected network. Under dynamic conditions the DAR and learning automata were shown to provide superior performance. The present paper further compares DAR and the L_{RI} algorithm for simple networks designed to force dynamic routing. L_{RI} is selected because it outperforms L_{RP} for the cases studied.

3. Experimental Results

Simulations were carried out for four and five node networks as shown in figures 2, 3 and 4. Network 1 is a four node network; network 2 is a five node fully connected network used elsewhere [5] and network 3 is a five node with links 1-3 and 3-5 being failed. Calls between nodes i and j are described by a Poisson process and exponential holding time. For each experiment the network simulator was run to equilibrium and then the total network blocking probability was measured defined as:

$$BP = \frac{\text{Total number of blocked calls}}{\text{Total number of offered calls}}$$

3.1. Tuning Dynamic Algorithms

Some dynamic algorithms have parameters that can influence their performance significantly. For example the time between updates for delta routing [5]. The performance of the L_{RI} scheme can be improved by properly tuning the learning parameter. Fig.5 shows the influence of the learning parameter on the performance of the algorithm. The test network and its traffic input are shown in the same figure.

From Fig.5 it can be seen that the blocking probability is very high for A < .99 and drops significantly as A increases with the minimum BP at A=.999. For A > .999 the blocking probability increases until A=1.0 which is actually RR (Random Routing).

The reason for the poor performance with A < .99 is that the algorithm converges to the selection of the path with a higher link capacity, path 1-2-3 (Fig.5) and as a result the available capacity of the other path, 1-4-3 will not be utilized. In other words when blocking probabilities are coarsely adjusted the algorithm locks on one path. This experiment has been repeated for another two sets of link capacities shown in Figs 6-7. In Fig.6 where the capacities of the two alternate paths are equal, the minimum blocking probability is at A=1.00 (Random Routing) as expected. Fig.7 shows that the minimum BP is at A=.999, therefore changing the capacities of alternate paths does not have a significant effect on the optimum value of the learning parameter.

The results presented in this paper were obtained for the optimized algorithm, i.e. where the value of learning parameter was selected to give the best performance for the problem studied.

3.2. Comparison between DAR and L_{RI}

Figs 8-10 compare the performance of the two algorithms for network 1 with a single traffic input(λ_{13}). Three sets of link capacities:

$C_{12} = C_{23} = 50, \quad C_{14} = C_{34} = 30$

$C_{12} = C_{23} = 60, \quad C_{14} = C_{34} = 20$

$C_{12} = C_{23} = 40, \quad C_{14} = C_{34} = 40$

have been considered with the total useful capacity being constant and λ_{13} varing over a range from about 30 to 100.

It can be seen that L_{RI} is superior to DAR for the intermediate range of traffic. When traffic is small the blocking probability is small for both schemes because there is sufficient capacity available. When traffic is high, a large proportion of calls will be lost and the two schemes utilize the available capacity. For intermediate range of λ the routing algorithms play a more important role because the traffic should correctly be divided between the two paths. The superiority of L_{RI} over this range suggests that it has a better spread of traffic than DAR.

A Comparison of Figs 8-10 shows that when the ratio of the effective capacity of the upper to lower path increases the two algorithms perform closer. This reflects the fact that DAR performs better the higher the ratio of the link capacities of the two alternate paths. Under theses conditions DAR requires less switching between alternate paths.

The next series of experiments were carried out for network 2 with two base traffic matrices given in tables 1-2. The efficiency of the algorithms were investigated under a uniform scaling of the base traffic matrix by a scale factor OVL. For each value of OVL the network simulator was run to equilibrium and then the total blocking probability was measured. The results are shown in Figs 11-12 which illusttrate an improved performance with L_{RI}.

Finally simulations were performed for network 3 with the following base traffic matrix: $\lambda_{13} = \lambda_{35} = \lambda$, $120 < \lambda < 200$. The results are given in Fig.13 which shows a significant superiority of L_{RI} over DAR. L_{RI} performs better for two reasons, first it reduces the probability of selecting the failed links to zero i.e it learns not to attempt them, while DAR does not detect the failure and at the time of switching from one path to another, attempts the failed link and looses some calls. Second the traffic must be equally split between two possible paths which exists for each traffic source, eg paths 1-2-3 and 1-4-3 for the node pair 1-3. Under these conditions

L_{RI} looses less calls than DAR when switching from one path to another.

3.3. Effect of Trunk Reservation Parameter (TRP)

In a previous research [4] effect of Trunk Reservation (TR) was studied on the performance of a five node fully connected network with different traffic matrices. The traffic matrices have been designed for FR and as a result that scheme gave the best performance. It was demonstrated that as TRP increased, the blocking probabilities improved for dynamic routings until a certain TRP for which all dynamic routings were as good as FR. The reason was that TR reduced the amount of dynamic routing in the network and above a certain TRP all dynamic routings behaved very similar to FR.

This experiment investigates the effect of TRP, when the traffic matrix is not designed for FR. Consider network 2 and traffic matrix 1. The table below shows steady state blocking probabilities for $0 < \text{TRP} < \infty$ and both DAR and L_{R-I} schemes. It can be seen that blocking probabilities increase as TRP increases until TRP=∞ (which is FR). Therefore different traffic patterns can amplify the difference between FR and dynamic routing. For the particular network considered any attempt to introduce TRP adversly affects the network performance. Clearly by increasing TRP more traffic will be routed directly and the differential between FR and dynamic routing will be reduced. However any attempt to introduce TRP for this class of networks increases blocking probability.

TRP	0	1	5	7	10	∞
DAR	1.09	1.23	2.96	4.45	5.83	6.49
L_{R-I}	.80	1.36	3.23	4.23	5.71	6.49

4. Conclusions

The effect of the learning parameter on the performance of a simple four node network was shown. As curves in Figs 5-7 indicate the algorithm can be tuned for an optimal performance. Simulation results for network 1 shows that as the ratio of the effective capacity of the upper to lower path increases, the advantages of the L_{RI} scheme are reduced. In the case of DAR a high ratio of link capacities on the two alternative paths is a favorable condition. Under these circumstances less switching will be required between the alternative paths for the DAR algorithm. It should be recalled that each time the DAR algorithm switches, a call is lost. Clearly as the

ratio of the capacity of the two alternative paths approaches unity, then increasing switching will be required and therefore more calls will be lost.

Results on five node networks deliberately designed to force dynamic routing illustrated an improved performance with L_{RI}. The failure condition experiment showed a significant superiority of L_{RI} over DAR for the case studied. Finally it was shown that for a traffic matrix not designed for FR, FR resulted in higher blocking probabilities than dynamic routing and any attempt to introduce TRP adversly affected the network performance.

References

[1] K.S. Narendra and P. Mars "The Use of Learning Algorithms in Telephone Traffic Routing-A Methodology", Automatica, Vol.19, No.5, pp.495-502, 1983.

[2] W-Y NG, "Dynamic Routing in Circuit-Switched Networks: Simulation of the Routing Rule of Least Busy Alternative with Trunk Reservation", Cambridge University, Project Report, Feb.1985.

[3] R.J. Gibbens "Some Aspects of Dynamic Routing in Circuit-Switched Telecommunication Networks", Rayleigh Prize Essay (1986) University of Cambridge.

[4] N. Eshragh, "Simulation studies of telephone traffic routing: comparison of different routing techniques with Learning Automata", Durham University, Project Report, July 1986.

[5] A.Girard and S.Hurtubise, "Dynamic Routing and Call Repacking in Circuit-Switched Networks", IEEE Transactions on Communications, VOL.COM-31, NO.12, Dec 1983.

0.0	15.0	15.0	15.0	30.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	15.0
0.0	0.0	0.0	0.0	0.0

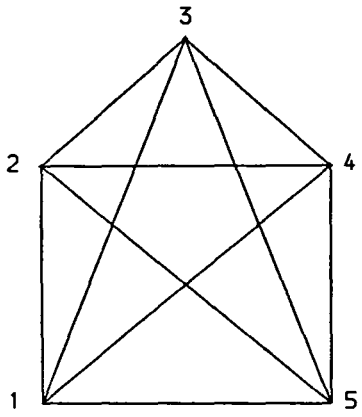
Table 1. Traffic matrix 1.

0.0	15.7	7.7	12.5	2.5
0.0	0.0	18.0	7.3	1.9
0.0	0.0	0.0	10.7	.9
0.0	0.0	0.0	0.0	1.2
0.0	0.0	0.0	0.0	0.0

Table 2. Traffic matrix 2.

List of Symbols

A	Learning parameter
B	Penalty parameter
BP	Blocking Probability
C_{ij}	Link capacity of the node pair (i,j)
DAR	Dynamic Alternative Routing
FL	Free Links
FR	Fixed Routing
LBA	Least Busy Alternative
L_{R-I}	Linear Reward Inaction
L_{R-P}	Linear Reward Penalty
OPT	Optimum
OVL	Overload factor
RR	Random Routing
TR	Trunk Reservation
TRP	Trunk Reservation Parameter
λ_{ij}	Traffic intensity of the node pair (i,j) in call per unit time



CALL FROM 2 TO 4

DIRECT PATH 2-4

ALTERNATIVE 1 2-1-4

ALTERNATIVE 2 2-3-4

ALTERNATIVE 3 2-5-4

Fig. 1. Direct and Alternative paths for the node pair (2,4).

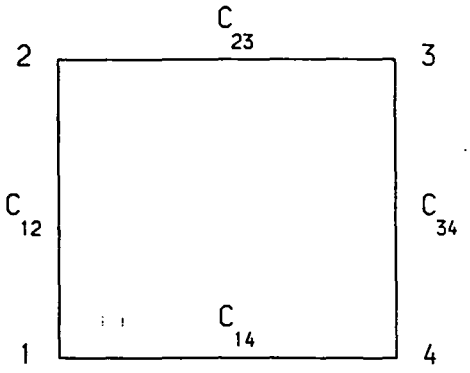


Fig. 2. Network 1: Four node

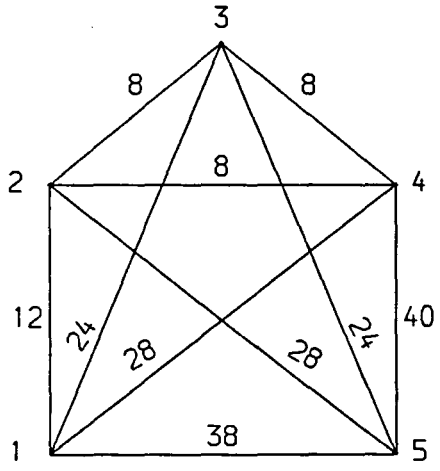


Fig. 3. Network 2: Five node

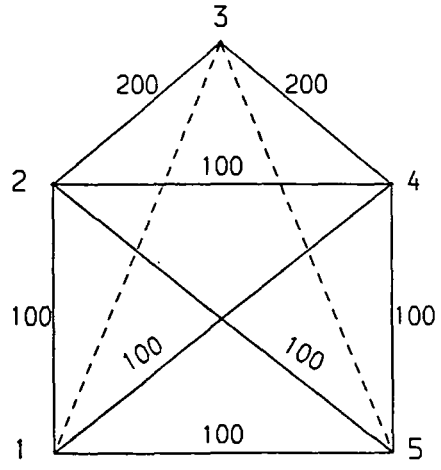


Fig. 4. Network 3: Five node

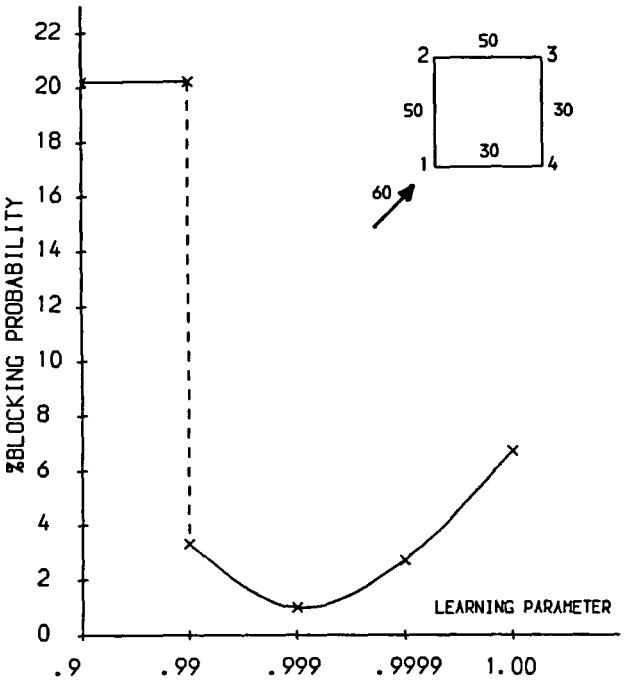


Fig. 5 Effect of the Learning parameter on the performance of the LRI scheme.

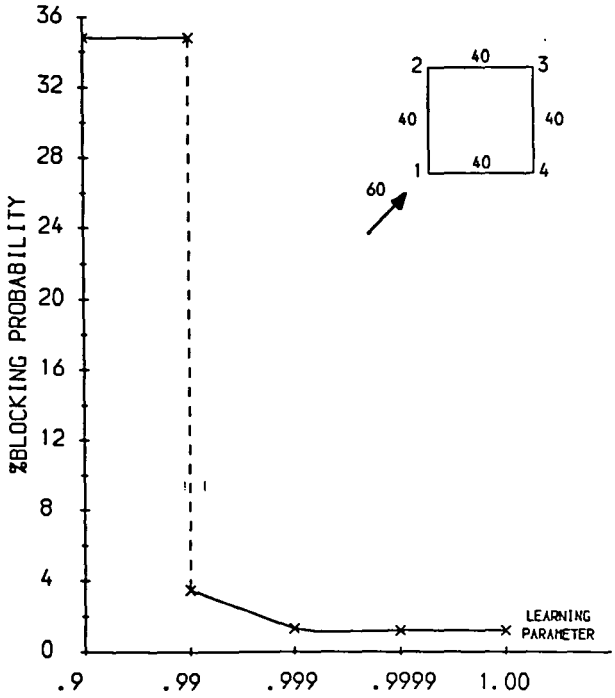


Fig. 6 Effect of the Learning parameter on the performance of the LRI scheme.

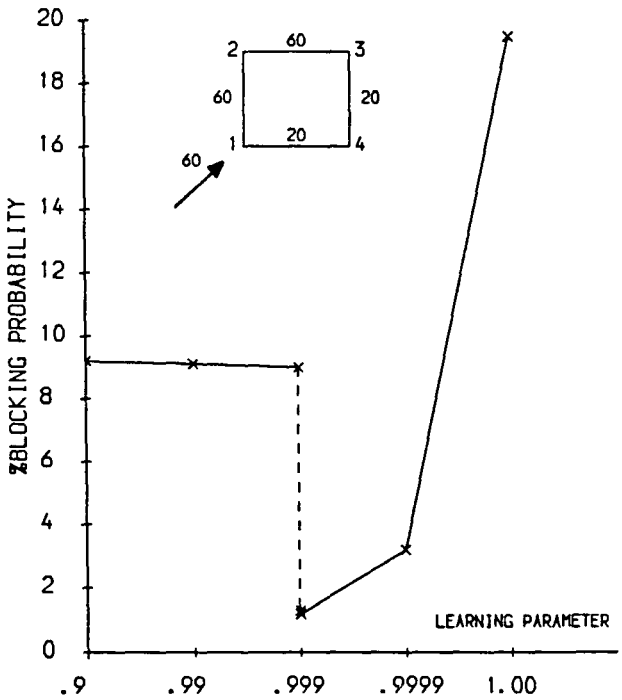


Fig. 7 Effect of the Learning parameter on the performance of the LRI scheme.

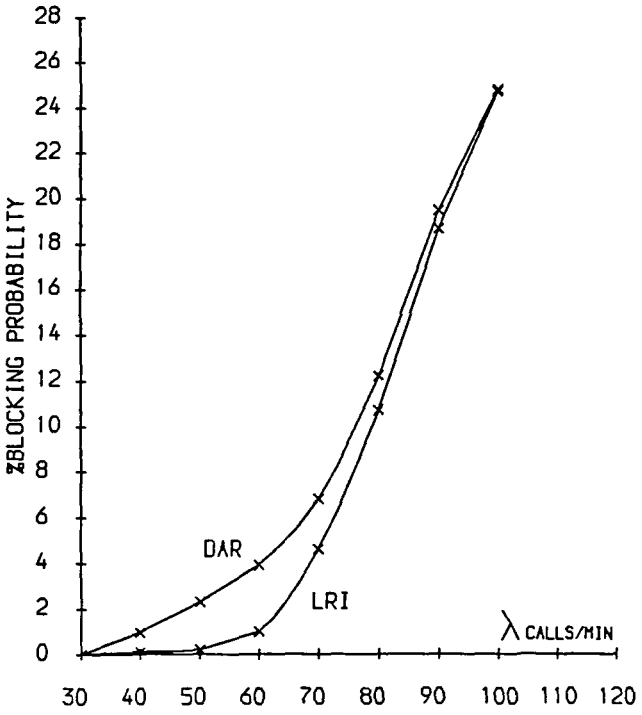


FIG. 8 Comparison between LRI and DAR.
Network 1, link capacity ratio 50/30.

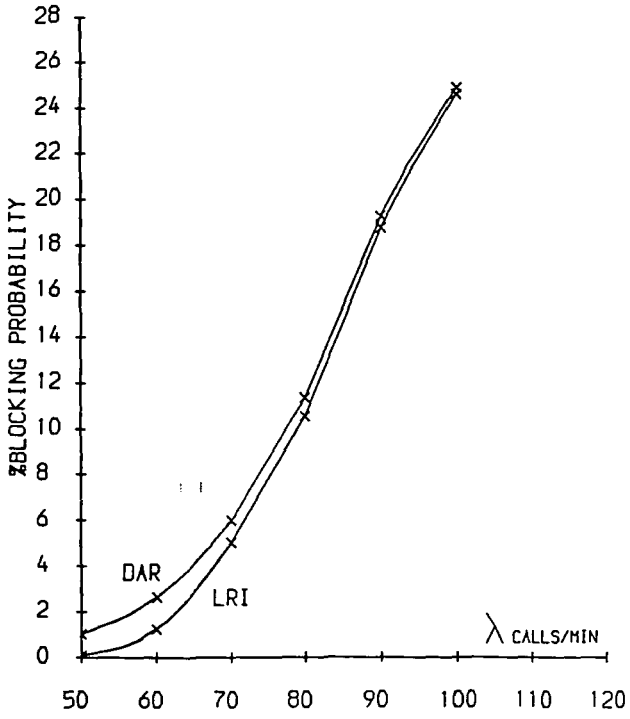


FIG. 9 Comparison between LRI and DAR.
Network 1, link capacity ratio 60/20.

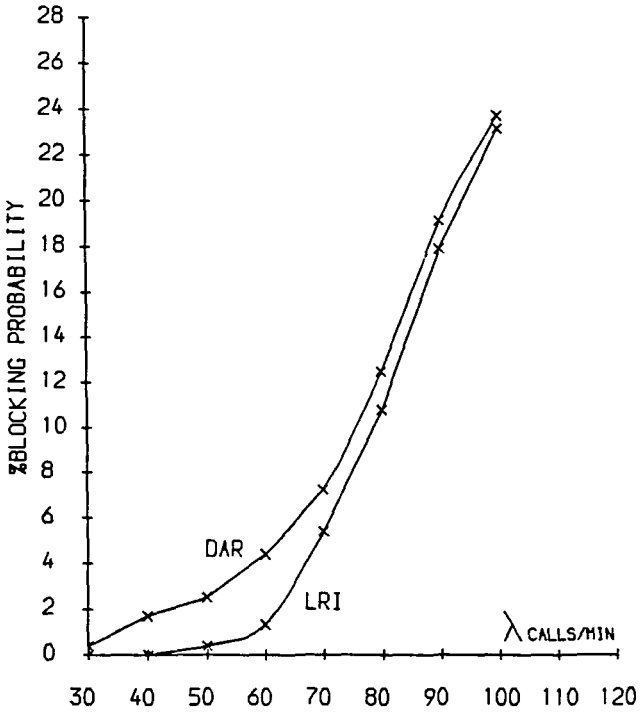


FIG. 10 Comparison between LRI and DAR.
Network 1, link capacity ratio=40/40.

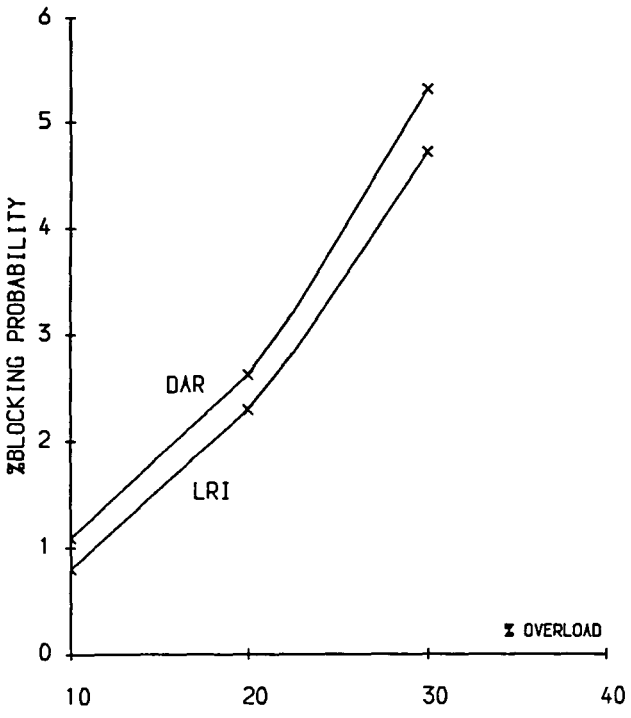


Fig. 11 Comparison between LRI and DAR. Network2, 2 traffic 1.

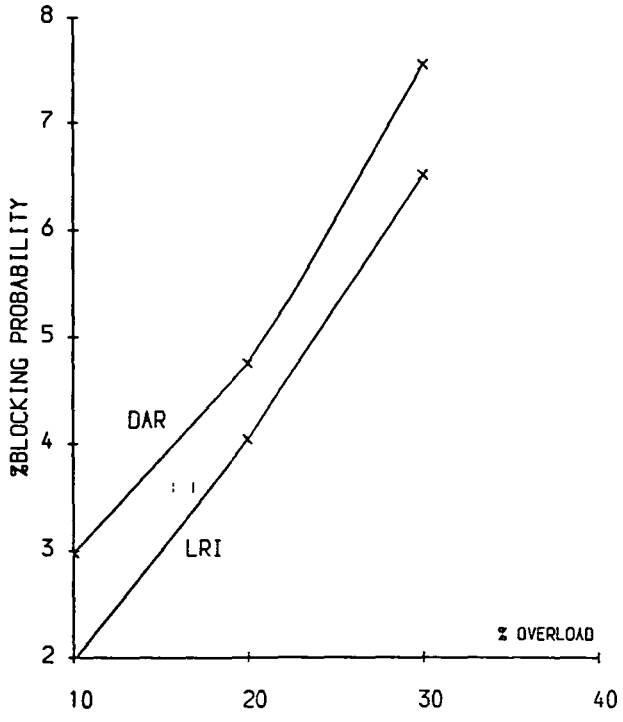


Fig. 12 Comparison between LRI and DAR. Network 2, traffic 2

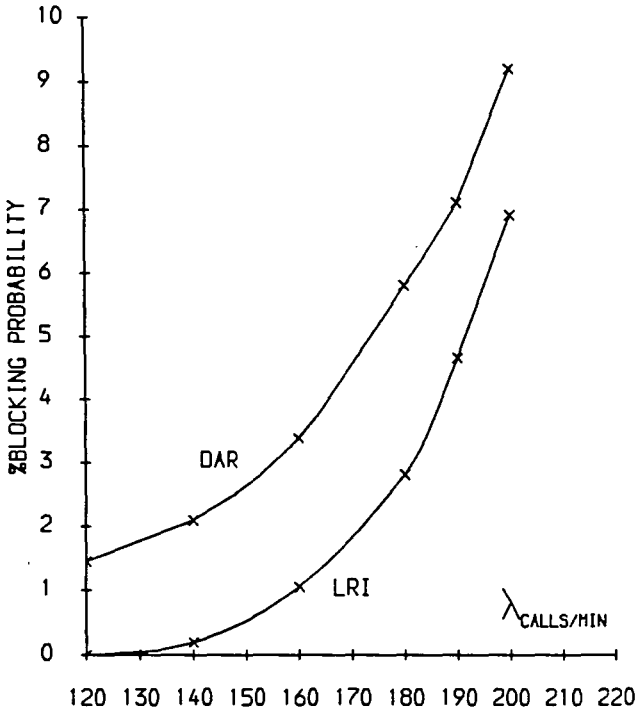


Fig. 13 Comparison of LRI and DAR under failure conditions.