Durham E-Theses

Optimal admission policies for small star networks

Antoniu, Nikolaos H.

OPTIMAL
ADMISSION POLICIES
FOR SMALL
STAR NETWORKS.

A thesis presented for the
degree of Doctor of Philosophy
at the University of Durham.

Nikolaos H. Antoniu

Department of Mathematical Sciences,
University of Durham,
Durham, DH1 3LE.
ENGLAND.

September 1994

1 3 JAN 1995

Abstract

In this thesis admission *stationary* policies for small *Symmetric Star* telecommunication networks in which there are two types of calls requesting access are considered. Arrivals form independent Poisson streams on each route. We consider the routing to be fixed. The holding times of the calls are exponentially distributed periods of time. Rewards are earned for carrying calls and future returns are discounted at a fixed rate. The operation of the network is viewed as a Markov Decision Process and we solve the *optimality equation* for this network model numerically for a range of small examples by using the policy improvement algorithm of Dynamic Programming. The optimal policies we study involve acceptance or rejection of traffic requests in order to maximise the Total Expected Discounted Reward. Our *Star* networks are in some respect the simplest networks more complex than single links in isolation but even so only very small examples can be treated numerically. From those examples we find evidence that suggests that despite their complexity, optimal policies have some interesting properties.

Admission Price policies are also investigated in this thesis. These policies are not optimal but they are believed to be asymptotically optimal for large networks. In this thesis we investigate if such policies are any good for small networks; we suggest that they are.

A reduced state-space model is also considered in which a call on a 2-link route, once accepted, is split into two independent calls on the links involved. This greatly reduces the size of the state-space. We present properties of the optimal policies and the *Admission Price* policies and conclude that they are very good for the examples considered. Finally we look at *Asymmetric Star* networks with different number of circuits per link and different exponential holding times. Properties of the optimal policies as well as *Admission Price* policies are investigated for such networks.

Nikolaos H. Antoniu ©MCMXCIV

Dedication

I dedicate this thesis to my parents Hristos and Elektra, my sister Helena, Antigone Karadimou and Melpomeni Petrani for their love and support.

This thesis is also dedicated to Catherine Louise Faulkner for being my *dream-keeper* and for making me realise that at one time in my life I might have had any number of stories, but now there is no other. This is the only story I will be able to tell.

Acknowledgements

The work in this thesis was carried out between January 1991 and September 1994 under the supervision of Dr. Iain M. MacPhee. I would like to express my graditude to Dr. Iain M. MacPhee for his help and guidance throughout the course of this work. Further acknowledgements are due to David Woof for his help in the computing work.

I would also like to thank the following: Dougal Wilson for being the other side of Nebs Communications, Barrie Hall because *the media is the message*, Simone for not being *die frau ohne schatten*, Prof. Protonotarios, Catherine, Ersie and Dimitrios Zevgolis, Dr. Panagiotis Dounis, Dr. Vicky Malandraki, Dr. Nikitas Vaptismas, Dr. Marina Moula and Kalliope Tsarouhi for being here when everything started, Elspeth Faulkner for sharing my love for Herge's *Tin-Tin*, Dr. Georgantopoulos for being the reason I first visited Durham, Alain Resnais, J.S. Bach for his *Triple Concert BWV 1044*, Mike and Mary Faulkner, Yiannis and Fotini, Bernarda and Fernanda de Utrera, Georgina Hillard, Mike, Ray and Roy, Barry Magarian for his Wagner.

Finally I would like to thank all the Members of Staff of the Department of Mathematical Sciences as well as the Secretarial Staff for their help and understanding.

Contents

Chapter 1

Introduction

1.1   Introduction

In recent years there has been a resurgence of interest in the mathematical theory of Telecommunication networks as well as the stochastic modelling of them, and in their application to the design and control of Telecommunication systems. This is due to

(1) the variety of new problems raised by advances in the technology of computers, electronics and communication systems; and

(2) the increased computing power available to researchers.

Throughout the century, problems from this field have provided an impetus to the development of probability theory, pure and applied.

Advances in the technology of modern Telecommunication systems have made it feasible to consider sophisticated schemes which can control the routing of calls within a network. Such schemes decide whether an arriving call should be accepted, and, if so, how it should be routed.

The answer to the question of how should calls be routed or capacity allocated so as to improve the performance of the network is not straightforward. An increase in

10

the offered traffic along a particular route will increase the blocking at links along that route; that will affect traffic carried along other routes that use these links, and also - in the case of alternative routing - along routes which act as alternatives. Such *hysterisis* or *knock-on* effects will generally propagate throughout the entire network. Nevertheless, it is more fundamental to ask whether a call should be accepted, since the routing of the call is in practice easier to answer and not so critically dependent on solving optimally. This is particularly true for modern computer and telecoms networks, which are able to respond to randomly fluctuating demands and failures by rerouting traffic and by reallocating resources. They do this so well, that in many respects, large-scale networks appear as coherent as intelligent organisms.

In this thesis models of a telecommunications, star-shaped, Loss network which consist of K links of the same capacity C linked through a common node are considered. Such networks are known as *Symmetric Star* networks.

There are two types of route on which calls can request admission: 1-link routes and 2-link routes involving any pair of the single links. Arrivals form independent Poisson streams on each route. The networks are circuit-switched in that before a request is accepted, it is first checked that sufficient resources are available to deal with the request. We also consider the routing to be fixed, that is a call has but one try to get through the network; otherwise, either rejected or denied a route, it is considered lost (Loss).

Every link contains circuits which the calls hold for some exponentially distributed periods of time. Dependency arises through occupancy of pairs of circuits. Both types of calls have the same exponential holding time. For every call carried we earn a reward; for different types of calls we earn different rewards. The rewards are discounted at a fixed rate.

The operation of the network is viewed as a Markov Decision Process which, when observed in time, is in one of a number of states. In the models we investigate *stationary* policies which choose actions depending on the state of the process at that time and hence our approach is *dynamic*. These actions involve whether to accept or reject traffic requests in order to maximise the Total Expected Discounted Reward (TEDR).

We solve the *optimality equation* for our networks numerically for a range of small examples by using the policy improvement algorithm of Dynamic Programming. The programs written can deal with a range of network sizes and offered traffics.

Generally, optimal policies are complex to describe and 'nobody' has exact solutions except for problems on a single link. *Star* networks are in some respect the simplest networks which aren't a single link or two in series but even so only very small examples can be treated numerically. For such examples we find evidence that suggest that the optimal policies have some interesting properties.

An obvious class of policies investigated also in our work, are the *Admission Price* policies. These policies are not optimal but they are believed to be asymptotically optimal for large networks; where the number of links grows to infinity. In this work we investigate if such policies are any good for small networks; we suggest that they are. Stationary distributions of blockings of links are also considered both theoretically and numerically. We then compare independent blocking assumptions - as presented in Kelly (1986, 1988) - with our numerically calculated 'exact' solution.

A reduced state-space model is also investigated in which a call on a 2-link route, once accepted, is split into two independent calls on the links involved. We comment on the size of the state-space and employ the *Admission Price* policies and conclude that they are very good for the examples considered. A comparison for the *Admission Price* policies between the full and the reduced model is carried out which shows that they are very similar. The policies are also compared to similar ones suggested in routing schemes proposed by Ott & Krishnan (1985, 1986) and Key (1990). Our reduced model is the next simplest network to that of two links in series with traffic which uses single links or use all the links; see Key (1990).

Finally we look at *Asymmetric* networks with different number of circuits per link and different exponential holding times. The *Admission Price* policies are investigated and compared to the ones for the full and reduced models. Properties of the policies of such networks are studied.

1.2 Telecommunication Networks: A Brief and Biased Introduction

Schemes that can control the acceptance and routing of calls in the network include *Dynamic* and *Adaptive Schemes*.

Dynamic Schemes are routing schemes which select a route for a call on the basis of the network state at the time of call-arrival and they are state-dependent. The purpose of such *Dynamic Schemes* is to:

(a) adjust routing patterns in accordance with varying and uncertain offered traffics;

(b) make better use of spare capacity in the network which may result from dimensioning upgrades or forecasting errors;

(c) provide extra flexibility in order to minimise network blocking; and

(d) provide robustness to respond to failures or overloads.

Two approaches in particular have received considerable attention. In the United States, AT&T has implemented a scheme called *Dynamic Nonhierarchical Routing (DNHR)* and in Canada, Bell-Northern Research has proposed a scheme called *Dynamic Controlled Routing (DCR)*.

The *DNHR* uses traffic forecasts for different times of the day in a large-scale optimisation procedure to predict a routing pattern. In *DNHR*, calls try their assigned paths in a pre-determined sequence, and the calls are blocked when all assigned paths are busy. The *DCR* proposes a central controller which receives information of the current state of all links at regular intervals of about 5-10 seconds to determine a routing pattern. The problem with the *DNHR* and *DCR* is that the former uses a large off-line calculation to advise on choices of alternative routes which can only change hourly, whereas the latter is centralised, time-delayed and requires detailed information about circuit occupancies and traffic arrivals.

Akinpelu (1984), studied the performance of *Nonhierarchical (NH)* and *Hierarchical (H)* networks under overloads using analytical and simulation models. Her work

13

assumes a *NH* network without controls in which its link sizes, offered loads and a fixed route for each pair are specified. In these networks the traffic offered is Poisson and the link blocking probabilities are independent. Calls blocked on a link of a path can always return to the switching system so that they can access the next path of their route. The basic idea of his analysis is to determine the offered load as a function of the link blocking. He provides equations which solved iteratively and starting with an initial estimate of the link blockings, determine the offered traffic and blocking probabilities.

The efficiency of *NH* and *H* networks was examined by Akinpelu (1984) in examples. She found that under no control, the *H* networks made more efficient use of their circuits under overload than do the *NH* networks; without control in *H* networks the calls carried increase steadily as offered load increases over the entire range of overloads considered. She also examined *NH* networks in which the control was *trunk reservation* for first routed traffic. Examples of both *NH* and *H* networks under *trunk reservation*[1] control are also examined only to conclude that such a control has an extremely beneficial effect on the performance of the network under overload in that by limiting the amount of multi-link and alternate-routing calls in the network, it allows more efficient use of the circuits for the one link calls.

Akinpeku (1984) and Ackerley (1987) have found that as the network load increases, there can only exist two states, a high blocking state with most of the calls carried on two link paths, and a low blocking state where most calls are carried on a single link.

Simulations for *Symmetric* networks carried out by Akinpelu (1984) and Ackerley (1987) show that a *maximal packing* strategy which accepts whenever it is possible to carry the call can have disastrous effects under general overload, creating instabilities and carrying much less traffic than if the dynamic policy was switched off. This was a very important result as it proposes a dichotomy between an individual and a 'social' or network optimum. This dichotomy suggests the idea of using a flexible *Dynamic Scheme* at low loads and turn it off at high loads. A simple way of doing this is to apply *trunk reservation*.

---

[1]Trunk reservation is presented and discussed in §1.6.

Dynamic Alternative Routing (DAR) was the routing scheme proposed by Gibbens, Kelly and Key (1989). *DAR* in contrast to *DNHR* and *DCR*, is decentralised and only uses local information with that being whether *trunk reservation* limits have been exceeded on a route, and the current recommended alternative route. Gibbens, Kelly and Key (1989), obtain bounds which hold for any *DAR* scheme and they compare the performance of *DAR* with such bounds. They developed a simple analytical model which enables *DAR* to be implemented on both large and small fully connected loss networks. Empirical validation of the model as well as a number of examples are discussed. They also show that *trunk reservation* controls the instability of dynamic routing and limits the extent of rerouting under various overloads.

Various aspects of dynamic routing in fully connected circuit-switched networks are considered by Gibbens and Kelly (1990). They consider networks of nodes and the calls use a single circuit/trunk between two nodes or can be rerouted via a tandem node.

In Gibbens and Kelly (1990), bounds on the overall performance of a dynamic routing scheme are derived under a fixed pattern of offered traffic. These bounds provide a measure of the extent to which it may be possible to improve on the performance of any given scheme. They study two types of bounds. The first they term *max-flow*[2] and holds under minimal assumptions concerning the stochastic structure of the system. The second bound they consider, called the *Erlang bound*, applies when streams of offered traffic are Poisson, and, is obtained by consideration of the random flows across the networks.

Gibbens and Kelly (1990) study *Symmetric* networks and use the *fixed-point* approximation general scheme to measure the effect of *trunk reservation*. A wide range of alternative routing is allowed. Gibbens and Kelly (1990), show that in networks with well-matched traffic and capacity, there is limited potential for dynamic routing to improve network performance. *DAR* is also studied. The problem of how the capacity of links within a network should be chosen is considered; this is known as the 'dimensioning problem'.

Routing schemes which select a route for a call on the basis of the network state at the time of an arrival in order to minimise network blocking were investigated by Ott

---

[2]The motivation for this bound was the fluid flow.

and Krishnan (1985) and Krishnan and Ott (1986). They consider circuit-switched networks with n links (trunkgroups) and C_k circuits in trunkgroup k, $k = 1, 2, .., n$. Their schemes have the properties that in order to compare the relative desirability of two routes, only information on trunkgroups in those routes is used, and that the comparison is simple enough to make the schemes implementable.

In their work, they make two assumptions about the stochastic behaviour of the network. The first is that all calls have exponentially distributed holding times with unit mean. The second is that as soon as a call has been accepted on an n-link route, it becomes n independent calls on the n links (trunkgroups) involved, each with an independent holding time as above. As a result, the state of the network at any time is reduced, and described by the number of occupied circuits on every link.

The offered link-loads are regarded as independent direct-routed Poisson loads offered to the links. At call arrivals, a routing decision needs to be made: in the non-alternate routing scheme, each arriving call is offered precisely to one admissible path; if the selected path is busy, no other path is tried and the call is blocked.

Their objective is to find a policy which minimises the average number of lost calls per unit time. Using results based on Howard's (1960) value determination and policy iteration methods, they show that many policies can be given in the form of a value function and when there is a way to compute such a function, we have in hand an implementable policy. To find such a value function they obtain a state-dependent routing scheme called *Separable Routing* when a policy iteration procedure of Markov Decision Theory on the finite state-space is used to improve upon a nominal scheme of direct routing. This procedure is carried out by considering a value cost of adding a call to a link; see also §1.7.1.

Krishnan and Ott (1985, 1986) compare their *Separable Routing* scheme with two other schemes: *DNHR* and the *Least-Loaded Routing (LLR)* using two network designs. *DNHR* is a state independent routing scheme and *LLR* uses state information. In *DNHR* a call is blocked if all assigned paths are busy, whereas in LLR, the blocked call is allowed rerouting to the path with the largest number of idle trunks. In comparison with *DNHR* and LLR schemes, Krishnan and Ott (1985, 1986) achieve a lower network blocking for a considerable range of overloads for the first design -

improvement occurs above a certain level of overload - but not for the second design; a detailed description of *DNHR* schemes can be found in Ash, Cardwell, and Murray (1981), *LLR* is examined in Ash (1985). Krishnan and Ott (1989) also observed that the use of their 'cost' formula appears to underestimate the true cost of a two-link call and thus cause *Separable Routing* to do more two-link routing than is desirable.

A modification to *Separable Routing* scheme was also presented in Krishnan and Ott (1989) as an attempt to accommodate alternate routing in which an arriving call has access to all admissible paths, and blocking on a single link or a path does not necessarily amount to call loss. This scheme is called *Forward-Looking Routing*, and achieves better performance than the other routing schemes such as *DNHR, LLR* and *Separable Routing* in that results to lower network blocking of calls over a wide range of loads. This scheme curbs the tendency for excessive two-link routing and thus reduces switch loads as well.

The question of minimising the cost of lost calls in networks of finite capacity which can carry calls of different types requiring different resources, was also investigated by Zachary (1988).

Zachary (1988) models the problem as a Markov decision process, using the objective of average cost function optimisation. He considers the application of the policy improvement method upon a 'base' policy, in order to find computationally tractable methods of determining nearly optimal policies; exact optimal policies are impossible to be computed due to the large state-space.

The 'base' policy considered is good in itself and allows the average cost function to be easily computed as a sum of components each of which usually depends only on the state of the network, at least to a good approximation: For any policy $\pi \in \Pi$, he considers the corresponding differential cost function of Markov Decision Theory, $\phi^\pi : X \to R$, normalised so that $\phi^\pi(0) = 0$, where X is the set of all possible states and Π is the set of all non-randomised *stationary* Markov policies. $\phi^\pi(x)$ is the cost of being in state x at a given time, relative to that of being in state 0 at that time, measured by the difference in expected future costs incurred in the network.

Zachary (1988) presents a theorem which shows that for any policy $\pi \in \Pi$, ϕ^π is the unique solution of a system of linear equations in $x \in X$. This cost function is then

used to determine the one-step improvement on the 'base' policy which is defined by the first iteration of the improvement method. This latter can be interpreted as an attempt, at each iteration, to define the 'optimal' policy, using the current differential cost function as an estimate of that at the optimum. Hence, the one-step improvement of a good initial good policy is general close to the optimum.

In this work existing calls are allowed rearrangement even when the arriving call is initially rejected. Zachary (1988) considers application of the theory to particular types of network, showing how good base policies might be defined with differential cost functions readily computable as described earlier. An example is a fully connected telephone network, with a direct link between each pair of nodes between which calls may arise. Examples of cellular radio networks are also examined. He finally considers briefly the extent to which the ideas mentioned in his work are applicable to more general networks and he mentions problems which most require further investigation in order to examine the robustness of his method with respect to deviations from the assumptions of the model.

1.3 The Network as a Markov Decision Process: Description and Mathematical Framework

1.3.1 Description of the Network

In this section, we consider the following description of a stochastic circuit-switched network, using the terminology of telephony and notation similar to that of Kelly (1986) and Key (1990).

There are K links, labelled $k = 1, 2, .., K$ and link i comprises C_i circuits/trunks. A subset $r \subset \{1, 2, .., K\}$ identifies a route. Let $\mathcal{R}$ be the set of possible routes. A call on route r uses A_{ir} circuits from link i, where $A_{ir} \in Z_+$. In the important special case where each element of the matrix $\mathbf{A} = (A_{ir}, i = 1, 2..., K; r \in \mathcal{R})$ is either 0 or 1 a route r can be identified with a subset of the set of links $\{1, 2, .., K\}$ - just set $r = \{i : A_{ir} = 1\}$.

In our work we consider $\mathcal{R}$ to be $\mathcal{R} = [\{i\}, \{i,j\}, \ i,j = 1,2,..,K; \ i \neq j]$, in other words all the one and two link routes. We also assume A_{ir} to be either 1 or 0 according to whether link i is a part of route r or not.

Assume that calls requesting route r arrive as a Poisson Process of rate ν_r, and that as r varies over $\mathcal{R}$ it indexes independent Poisson streams. Suppose that there is no control and therefore a call requesting route r is blocked and lost if on any link i in the route, there are less than A_{ir} circuits free. Otherwise the call is accepted and simultaneously holds A_{ir} circuits from links i for the holding period of the call. Holding periods of calls on route r are identically exponentially distributed with mean μ_r^{-1}.

Let $n_r(t)$ be the number of calls in progress at time t on route r, $\mathbf{n}(t) = (n_r(t), r \in \mathcal{R})$, and $\mathbf{C} = (C_1, C_2, .., C_K)$. The exponential holding times of Poisson arrivals make the sequence $(\mathbf{n}(t), t \geq 0)$ a Markov Process that is a stochastic process with the Markovian property which, when observed in time, is in one of a number of states. The stochastic process $(\mathbf{n}(t), t \geq 0)$ has a unique stationary distribution and under this distribution $\pi(\mathbf{n}) = P(\mathbf{n}(t) = n)$ is given by the product form

$$(1.1) \qquad \pi(\mathbf{n}) = G(\mathbf{C}) \prod_{r \in \mathcal{R}} \frac{(\nu_r/\mu_r)^{n_r}}{n_r!}, \quad \mathbf{n} \in \mathcal{S}(\mathbf{C}),$$

where

$$(1.2) \qquad \mathcal{S}(\mathbf{C}) = \{\mathbf{n} \in Z_+^{\mathcal{R}} : 0 \leq A\mathbf{n} \leq C\}$$

and $G(\mathbf{C})$ is the normalising constant or partition function

$$(1.3) \qquad G(\mathbf{C})^{-1} = \sum_{\mathbf{n} \in \mathcal{S}(\mathbf{C})} \prod_{r \in \mathcal{R}} \frac{(\nu_r/\mu_r)^{n_r}}{n_r!}.$$

The result (1.1) is easy to check if we view the process as a multi-dimensional birth and death process with equlibrium distribution given by (1.1).

Most quantities of interest can be written in terms of the distribution (1.1) or the

partition function (1.3). For example let L_r be the stationary probability that a call requesting route r is lost, then, since the arrival stream of calls requesting route r is Poisson,

$$(1.4) \qquad 1 - L_r = \sum_{\mathbf{n} \in \mathcal{S}(C - Ae_r)} \pi(\mathbf{n}) = G(\mathbf{C}) \, G(\mathbf{C} - \mathbf{A}e_r)^{-1},$$

where e_r is the unit vector from $\mathcal{S}(\mathbf{C})$ describing just one call in progress on route r.

The above simple explicit form (1.3) does not provide the complete solution because it is impractical for all but the smallest networks to compute G directly. Note that the number of routes $\mathcal{R}$ may grow as fast as exponentially with the number of links K. For a description of the problem see Harvey and Hills (1979) as well as Louth, Mitzenmader and Kelly (1994).

1.3.2 Reversible and Irreversible Processes

The distribution (1.1) satisfies the detailed balance equations

$$\pi(\mathbf{n}) \, \nu_r = \pi(\mathbf{n} + e_r) \, (n_r + 1), \qquad \mathbf{n}, \mathbf{n} + e_r \in \mathcal{S}(\mathbf{C})$$

where e_r is the unit vector describing just one call in progress on route r. This models a reversible process, and as Burman, Lehoczky & Lim (1984) and Whittle (1986) prove, the distribution (1.1) is insensitive to the form of the holding time distribution. In what follows we assume that the holding times have an exponential distribution with unit mean.

A classical example of the above model is a telephone network. The model also rises naturally in the study of local area networks, multi-processor interconnection architectures, database structures, mobile radio and broadband packet network; Kelly (1985) and Hui (1990, 1991).

Reversibility is a very attractive property, giving a closed product form which can be

approximated [see Kelly (1991)], and some authors have looked at optimal controls[3] which preserve reversibility; see for example Kelly (1986), Key (1988) and Zachary (1988). Probabilistically, thinning the arriving stream - by accepting say 1 in x calls - is such a control, as are 'admissible set policies' which restrict the sets that could be used; for example we might in addition to equation (1.2) require that $\mathbf{n} \in \Omega \subset \mathcal{S}(\mathbf{C})$. Such 'admissible' controls which preserve reversibility were studied by Foschini and Gopinath (1981). 'Co-ordinate convex' control [Ross and Tsang (1989)] which effectively truncates the state-space also preserves reversibility.

For our model we must decide at arrival times of calls whether to accept or reject them taking into account the state of the process. Thus we have a Markov decision process. To accept or reject arrivals we must follow some policy. A policy is any rule for choosing actions. Policies in general are mappings $\pi : \mathcal{S} \times R^+ \to \mathcal{A}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. In this work we consider policies which choose actions depending on the state of the process at that time and are mappings $\pi : \mathcal{S} \to \mathcal{A}$. Such policies are called *stationary*.

Blackwell (1965), shows that if the state-space and the action-space are finite, there is an optimal policy according to some control, and furthermore there is an optimal *stationary* one; see also Ross (1983).

If the process is in state $(\mathbf{n}(t))$ at time t and action a is chosen, then independent of the past, two things occur: (a) We receive a reward R_r immediately if an arrival of type r is accepted; (b) The process moves to a new state after an exponential time which is independent of the control policy employed but depends upon the state of the process. Thus the expected time between transitions is not constant. After *uniformisation* [Lippman (1975)], the Markov Decision process can be represented by an equivalent discrete-time Markov decision process n(t). *Uniformisation* makes the rate of transitions between states constant, and turns the problem into one that can handled discretely (and hence on the computer); see Chapter 2.

However, now that there is a *stationary* optimal state-dependent control, one that is deterministic and doesn't depend on time, and which says whether to accept or reject a call of type r depending upon the network state in order to maximise the average reward, the controlled process is not in general reversible.

---

[3]See §1.1 for the need of controls.

1.3.3 Optimality and Maximising the Reward

As pointed out by Ott & Krishnan (1986), Zachary (1988) and Key (1990), any policy which is expressed in the form of routing decisions is impractical to implement. Research for years have been focused on methods that are general enough to be desciptive and yet computationally feasible. The work of Bellman (1957) and Howard (1970) on the Dynamic Programming and Markov Chain Theory was pioneering in the field. The basic idea of their work is that Dynamic Programming can result a recursive procedure for calculating an optimal value function from a functional equation. Such research has supported the suggestion that many policies can be given in the form of a value function often called 'relative-value' and hence if there exists a mechanism to compute such functions, we have an implementable policy; see for example Howard (1960), Ott & Krishnan (1985,1986) and Ross(1983). The emphasis is therefore stressed on computing or approximating such value functions.

The functional equation is obtained from the principle of optimality, stating that an optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first transition - a principle being always valid when the number of states and the number of actions is finite. Therefore calculating the optimal value functions calculates the optimal policy.

Under the above assumptions one can look for an optimal control policy to maximise the average reward over an infinite or finite length of time in which the reward is accumulated. Rewards can be discounted or not. Discounting at rate $\xi > 0$ means that a reward R, received at time t has present value $R\exp(-\xi t)$. Apart from ensuring that the total reward is bounded, discounting suggests the fact that the rewards earned in the future are worth less than those earned immediately. This is very useful since in approximations one might not wish to look too far ahead. In case that the discounting is not considered and the time horizon is infinite, any policy that does not reject all the calls will result in an infinite reward, and therefore the criterion in this case to select among policies is the reward earned per unit time or average reward we seek to maximise; see Key (1990).

Let $V(z)$ denote the expected discounted reward earned over an infinite time length given the intial state is z, and $V_n(z)$ be the reward when n transitions remain. Using

uniformisation the Dynamic Programming enables the recursions or value iterations for V_n to be written down immediately under the optimal policy; see Lippman (1975), Ross (1983) and Key (1990). Under this, it is optimal to accept a type r call if and only if

$$R_r > W_n(z,r) = V_n(z) - V_n(z + e_r).$$

It can be shown that the recursions considered is a contraction mapping on the set of bounded functions on the state-space, and hence the function V_n converges to a limit V which satisfies the optimality equation, for all bounded initial V_0; see Ross (1970, 1983). This method is the method of successive approximation or value-iteration, which provides an efficient computational approach.

If $R(t)$ denotes the reward earned by time t with no discounting ($\xi = 0$) for a policy π and z_0 is the state at time $t = 0$, then the expected reward $\phi_\pi(z)$ given an initial state z is defined as

$$\phi_\pi(z) = \lim_{t \to \infty} E\left[\frac{R(t)}{t} | z_0 = z\right].$$

If for some policy f, $\phi_f(z) = \sup_\pi \phi_\pi(z)$, $\forall z$, then f is said to be average-reward optimal.

Let now $V_{f,n}^*(z)$ be the n-stage return under stationary policy f with no discounting and initial state z, then using the *uniformised* transition rates Λ gives another kind of expected average reward, $\phi_f(z)$,

$$\phi_f(i) = \lim_{n \to \infty} E\left[\frac{V_{f,n}^*}{n/\Lambda}\right].$$

Ross (1983) shows that the average cost does not depend upon the initial state and links the average-reward optimal and discounted reward via a theorem which proves that

$$(1.5) \qquad \lim_{\xi \to 0} \xi V(z) = \phi_\pi(z), \ \forall z.$$

In our work the optimality criterion we use to discriminate between policies is the total expected discounted return; see Chapter 2.

1.4   Size of Networks and the State-Space

In this work we look for optimal policies in time where the state of the network can be described by the state of all current routes. Thus, the optimal state-dependent routing in our case is a problem of optimal control of a Markov decision process in a huge state-space. Because of the size of the state-space, numerical calculations are very difficult and any policy which is given in the form of routing decisions is unimplementable.

To demonstrate the problem with the state-space size, consider a model of a reduced *Star* network with K links each of capacity C in which calls arrive randomly and according to a Poisson distribution. There are two types of route: 1-link routes and 2-link routes involving any pair of the single links. A 2-link route on pair (i,j), after its acceptance, is split into 2 independent single link routes on links i and j; this network is often called a reduced state-space network. Both types of calls have an *exponential* holding time with mean 1, (Exp(1)). For this example the state of the network at any time we observe it is the number of busy circuits in the K links. In this 'simple example' the size of the state-space $|\mathcal{S}|$ is given by the expression

$$(1.6) \qquad |\mathcal{S}| = (C+1)^K;$$

For a network with $K = 190$ and $C = 20$ (1.6) is $|\mathcal{S}| = 21^{190}$ which is considerably more than 10^{80}, the estimated number of elementary particles in the universe [Ott and Krishnan (1985)]; see Table 1.1.

Under the assumptions of the *Symmetric Star* network we investigate in Chapter 2, things are more complicated as there is no closed form expression in general for

calculating the size of the state-space. The rate of increase in the size is so rapid that direct numerical approaches can never deal and even in moderate sized systems there will always be a problem. In our work the size of the state-space was obtained by counting the number of states on a computer; see Appendix B.1.

The following table gives the size of the state-space for a range of small values of K, the number of links and C, the capacity of each link for the reduced state-space network model[4] as well as the full model[5].

K	C	$(C+1)^K$	$|\mathcal{S}|$
2	3	16	30
2	4	25	55
3	3	64	336
3	4	125	1023
3	5	216	2610
3	6	343	5860
3	7	512	11942
4	3	256	5142
4	4	625	28746
4	5	1296	124074
4	6	2401	442918
4	7	4096	1366806
5	3	1024	101368
5	4	3125	1131389
5	5	7776	8940840
5	6	16807	54653970
5	7	32768	273816800

Table 1.1: The size $|\mathcal{S}|$ for the state-space.

For our *Symmetric Star* network by permuting the ordering of the links, many of the states when K is large are equivalent.

For example, in a *Symmetric Star* network with $K = 4$, $C = 3$ and $|\mathcal{S}| = 5143$, we could have 4! equivalent states: If we represent the state of the network as a vector

$$\left(x_a, w_{ab}, w_{ac}, w_{ad}, x_b, .., x_d\right),$$

---

[4]Discussed in Chapter 5.
[5]Discussed in Chapters 2,3 and 4.

where x_i denotes the number of single link calls on link i, and w_{ij}, the number of two-link calls on pair (i, j), and look at the states

$$A = (1, 1, 0, 0, 0, 0, 2, 2, 0, 0), \quad B = (1, 1, 0, 0, 0, 2, 0, 0, 0, 2)$$

we can see that $w_{ad}(A) = w_{ac}(B)$, and $w_{bc}(A) = w_{bd}(B)$, and re-labelling c and d has no effect. Further research might highlight probable benefits of this equivalence, in order to reduce the size of the huge state-space. It seems to be very difficult to try to take advantage of this. There are many serious problems to be considered about for example how to debug and to be confident of the results and of course the restriction that such a benefit could only apply to *Symmetric* networks; our computing programs are made to work for asymmetric cases too.

The major problem that the large size of the state-space causes can be reviewed as follows: Our model is a Markov decision process in which our goal is to maximise the Total Expected Discounted Reward V and as such there is an *optimality equation* which is used in a recursive procedure for calculating optimal value functions V and for the calculation of the optimal policy itself. The optimality equation after *uniformisation* is linear in V but the state-space quickly becomes huge. If we define the iterative method to be

$$AV_p = R_p, \quad \text{for} \quad V_p,$$

where R_p is the rewards vector, we can see that A is $|\mathcal{S}|^2$ in size while V and R are size $|\mathcal{S}|$. We have efficient storage for the policies as well as the values V but not for A. The storage problem as well as the important fact that we are interested in the policies more than the values V were the reasons not to use the value-iteration peocedures in our calculations; see §1.3.3. A more detailed presentation can be found in Chapter 2.

Due to the large size of the state-space we have restricted our investigation for networks with:

(a) $K = 2$, $C \leq 7$,

(b) $K = 3$, $C \le 5$,

(c) $K = 4$, $C \le 4$,

(d) $K = 5$, $C = 3$.

In Appendix A we consider the difficulties arising in trying to calculate the state-space for the *Symmetric* networks considered in this thesis.

1.5 Approximation procedures and Asymptotic analysis

Blocking and Loss probabilities for calls in networks have been the subject of much research [see for example Gibbens, Girard, Kelly, Mitra, Key, Ziedins, Whitt]. The behaviour of such networks under limiting regimes has been studied and properties of the models have been presented. The above researchers modelled networks with increasing link capacity and offered traffic, but fixed network topology. That is because if capacities increase more quickly then all blocking probabilities will tend to 0, while if capacities increase more slowly then some blocking probabilities will approach 1.

A major advance has been the development of approximation procedures. One of the main reasons for that was the intractability of (1.1) - (1.4). Such approximations avoid computational problems and in some cases provide deeper insights. The most important is the *Erlang fixed-point* approximation developed and studied by Holtzman (1971), Lin et al (1978), Girard and Ouimet (1983), Heyman (1985), Whitt (1985), Kelly (1986, 1991) and others.

The *Erlang fixed-point* approximation we now describe. Let

$$(1.7) \qquad E(\nu, C) = \frac{\nu^C}{C!} \left[\sum_{n=0}^{C} \frac{\nu^n}{n!} \right]^{-1}$$

Erlang's formula for the loss probability of a single link of capacity C offered Poisson traffic at rate ν. Let E_i, $i = 1, 2, .., K$ solve the non-linear equations

27

$$(1.8) \qquad\qquad E_i = E(\rho_i, C_i), \quad i = 1, 2, .., K$$

where

$$(1.9) \qquad\qquad \rho_i = \sum_{r \in \mathcal{R}} A_{ir} \nu_r \prod_{j \in r - \{i\}} (1 - E_j)^{A_{jr}}.$$

Then an approximation for the loss probability on route r is given by

$$(1.10) \qquad\qquad 1 - L_r \approx \prod_{i \in r} (1 - E_i)^{A_{ir}}.$$

The reasoning behind this approximation is as follows. Suppose that a stream of rate ν_r is thinned by a factor $(1 - E_j)$ at each link $j \in r - \{i\}$ before being offered to link i. If these thinnings could be assumed independent both from link to link and over all routes passing through link i (they clearly are not), then the traffic offered to link i would be Poisson at rate (1.9), the blocking probability at link i would be given by (1.8) and the loss probability on route r would satisfy (1.10) exactly.

Heyman (1985) and Whitt (1985) call the *fixed-point* procedure the *reduced load* approximation.

The *fixed-point* approximation procedure is an important one for a number of reasons. Firstly, it has a long history in Telecommunications and has been found to be an effective approximation in a variety of circumstances. Secondly, it has recently been the subject of a number of theoretical analyses, establishing its accuracy under various limiting regimes [Whitt (1985), Kelly (1986), Ziedins and Kelly (1989)]. The approximation has a mathematical interest due to similarities between the approximated and asymptotic results as we shall see later. Thirdly, it can accommodate additional features such as alternative routing and *trunk reservation* discussed also in Kelly (1986, 1990, 1991).

In the following subsections a brief description of the work on *reduced load approximation* and limiting behaviour of large networks examined by Kelly is given along with his equations and results. The idea of *decentralisation* is also discussed. Other

results by other researchers are also featured.

1.5.1   Limiting Regimes and Fixed-Point Approximation

Kelly (1986) aims to show that the analysis of circuit-switched networks becomes simpler the larger the network. He shows that when the capacity and offered traffic are increased together in a network with fixed topology, a limiting regime emerges in which loss probabilities are as if the links block independently, with blocking given by the solution of a simple convex optimisation problem. Kelly considers the *fixed-point* approximation procedure based on solving Erlang's formula under the assumption of independent blocking. This procedure produces a unique solution with fixed routing and under the limiting regime the estimated loss probabilities obtained from the procedure converge to the correct values.

He first considered the optimisation problem of finding the most likely state n under the probability distribution $\pi(\mathbf{n})$. This is equivalent to

$$(1.11) \qquad \max \sum_r (n_r \log \nu_r - \log n_r!)$$

over $\mathbf{n} \in \mathcal{S}(\mathbf{C})$, a problem which is made difficult by the discrete nature of the state-space. To simplify this replace $\log n!$ by $n \log n - n$ by using Stirling's formula and replace the integer vector $\mathbf{n}$ by a real $\mathbf{x}$. The resulting problem is

$$(1.12) \qquad \max \quad \sum_r (x_r \log \nu_r - x_r \log x_r + x_r)$$
$$subject\ to \qquad \mathbf{x} \geq 0, \mathbf{A}\mathbf{x} \leq \mathbf{C}.$$

The dual of the above problem expressed in terms of a vector $\mathbf{y} = (y_1, y_2, .., y_K)$ is to

$$(1.13) \qquad \min \quad \sum_r \nu_r \exp\left(-\sum_i y_i A_{ir}\right) + \sum_i y_i C_i$$

29

$$\text{subject} \quad \text{to} \qquad \mathbf{y} \geq 0.$$

Unless otherwise specified, summations run over $r \in \mathcal{R}$ or $i \in \{1, 2, .., K\}$.

Kelly (1986) treated problems (1.12) and (1.13) rigorously and proved that there exists a unique optimum $\mathbf{z} = (z_r, r \in \mathcal{R})$ to problem (1.12). It can be expressed in the form

$$(1.14) \qquad z_r = \nu_r \prod_i (1 - B_i)^{A_{ir}}, r \in \mathcal{R}$$

where $B = (B_1, B_2, .., B_K)$ is any solution to the set

$$(1.15) \qquad \begin{aligned} \sum_r A_{ir} \nu_r \prod_j (1 - B_j)^{A_{jr}} &= C_i, \text{ if } B_i > 0 \\ \sum_r A_{ir} \nu_r \prod_j (1 - B_j)^{A_{jr}} &\leq C_i, \text{ if } B_i = 0 \\ B_1, B_2, ..., B_K &\in [0, 1). \end{aligned}$$

There always exists a solution to relations (1.15), and it is unique if $\mathbf{A}$ has rank K. There is a one-to-one correspondence between solutions to relations (1.15) and optima of problem (1.12), given by the transformation $1 - B_i = 1 - \exp(-y_i)$, where $\mathbf{y}$ is the vector of Lagrange multipliers. [A proof can be found in Kelly(1986)].

Relations (1.15) have a straightforward interpretation in terms of a continuous, or fluid, traffic. Suppose that an offered traffic of ν_r on route r is thinned by a factor $(1 - B_j)^{A_{jr}}$ on each link j, so that a traffic of

$$(1.16) \qquad \nu_r \prod_{j \in \mathcal{R}} (1 - B_j)^{A_{jr}}$$

remains on route r where one unit of traffic on route r uses A_{ir} at link i. Then relations (1.15) state that at any link i for which $B_i > 0$ the total capacity of that link, C_i, must be utilised by the superposition over $r \in \mathcal{R}$ of the traffic (1.16).

In Kelly's limiting regime the behaviour of loss probabilities L_r has a very simple description. There is a parameter $B_i \in [0, 1)$ associated with link i such that

$$(1.17) \qquad L_r \to 1 - \prod_{i \in r} (1 - B_i)^{A_{ir}}, \quad r \in \mathcal{R}.$$

The above limit is deduced as a law of large numbers; see Kelly (1986). This limit allows him to classify a link as overloaded, critically loaded or underloaded which has an impact for the process describing the number of free circuits at the link.

Note the similarity between (1.17) and (1.10).

It is as if links block independently, link i blocking with probability B_i. Now the various thinnings are clearly not independent, but the reasoning does suggest that the approximation (1.17) might be more accurate the more diverse the collection of routes passing through any given link.

Kelly (1986) and, Ziedins and Kelly (1989) provide theoretical evidence for this suggestion, by presenting an asymptotic analysis of networks exhibiting various forms of symmetry.

Heyman (1985) and Whitt (1985) obtain results similar to Ziedins and Kelly (1989) for *Symmetric* networks. Whitt's approach is to prove a functional law of large numbers for the process

$$\{[Q_K^1(t), \ldots Q_K^C(t)], t \geq 0\} \quad as \quad K \to \infty,$$

where $Q_K^i(t)$ represents the numbers of links with i busy circuits at time t in a *Symmetric* network with K links. This approach is very powerful: it provides results for the non-stationary behaviour of the system when holding times are exponential, and promises to be able to deal with networks where the product-form solution (1.1)-(1.4) is not available. In contrast, Ziedins and Kelly (1989) make heavy use of particular features of the partition function (1.3). They show that the *reduced load* approximation may give either an upper or a lower bound on the loss probability of a call in a *Symmetric* network; an upper bound on the error of the approximation is obtained in the special case when $C = 1$.

31

Heyman (1985), Kelly (1985), Mitra and Weinberger (1984) and Mitra (1985) all present results on blocking and loss probabilities under traffic conditions where arrival rates are normalised so that as network size increases blocking probabilities approach 0.

Key (1994, 1990) examined the problem of *Adaptive Control* in the context of limiting regimes, where we accept or reject probabilistically, with no knowledge of the underlying state. He describes the asymptotic regime and re-confirms the results of Kelly (1986), Whitt (1985) and Ziedins (1986) which show that approximations are asymptotically valid as networks increase in size in a certain way, or as the traffic and circuits grow larger. His results say that asymptotically, we can use *adaptive routing* to maximise our gain.

Key (1990, 1994) uses the limiting regime suggested by Kelly (1986) which assumes a sequence of problems where the offered traffic λ_r and capacities C are replaced by $\lambda(N) = (\lambda_r(N), r \in \mathcal{R})$, $C(N) = (C_j(N), j = 1, 2, .., K)$ for $N = 1, 2, ..,$, and where the following is satisfied as $N \to \infty$:

$$\frac{\lambda_r(N)}{N} \to \lambda_r, \quad \frac{C_j(N)}{N} \to C_j,$$

where $r \in \mathcal{R}$ and $j = 1, 2, ..., \mathcal{S}$.

He considers $\mathbf{x}$ to be the optimal solution to the following linear program which gives the *max-flow* bound [see Gibbens, Kelly and Key (1989)] on the optimal return under any dynamic routing,

(1.18)   **LP1** : max $\sum_{r \in \mathcal{R}} x_r R_r$

   subject to $x_r \leq \lambda_r$, $r \in \mathcal{R}$

   $x_r \geq 0$

   $\sum_{r \in \mathcal{R}} x_r A_{jr} \leq C_j.$

where x_r represents the carried traffic on route r, and R_r is the return associated with each call of type r. Let $\mathbf{x}(N)$ be the optimal solution to **LP1** when the traffic

32

and capacities $\lambda(N), \mathbf{C}(N)$.

Key (1990) shows that

$$(1.19) \qquad \frac{1}{N}\mathbf{x}(N) \to \mathbf{x}$$

and that this bound can be achieved asymptotically by the state-independent adaptive policy which rejects a proportion P_r of type r traffic, and routes the remainder of the traffic directly, where $P_r = (1 - \frac{x_r}{\lambda_r})$. In other words, if z_r is the carried traffic under this scheme, then

$$\frac{1}{N}z_r(N) \to \frac{1}{N}x_r(N) \to x_r$$

where

$$P_r(N) = 1 - \frac{x_r(N)}{\lambda_r(N)}.$$

Using Kelly's results, Key proves that

$$z_r(N) = (1 - P_r)\lambda_r \prod_j (1 - b_j)^{A_{jr}}, \qquad \circ$$

where b_j are the link blockings which satisfy (1.15) with $\nu_r = \lambda_r(1 - P_r)$.

He emphasises on the fact that if $b_j = 0$ for all j, then the equations that b_j satisfy are identical to the optimum $\mathbf{x}$ which solves (1.18).

His theorems say that asymptotically we can use adaptive routing to maximise our return, and that there is nothing to be gained by accepting more traffic into the network than the solution to the linear program (1.18).

The expected undiscounted rate of return from a network with no controls where the blocking for a link is calculated in Key (1990) as the unique solution to a set of *fixed-point* equations is examined, and results similar to those of Kelly (1986), Lin

33

(1978), Girard and Ouimet (1983) are reached.

1.5.2 Decentralisation

Another very interesting idea tackled by Kelly (1988, 1990) was to investigate how the routes in use affect the performance of the network and furthermore, if there are many substitute routes that could carry calls between two nodes to search if there is a criterion for comparing their efficiency.

A related issue in the literaure of telecommunications concerns the extent to which control can be *decentralised*. This asks whether control could be distributed over the switching nodes of the network, with computations and decisions made locally.

Over a period of time the form of the network or the demands placed on it may change, and routings may need to adapt accordingly. A single node could perhaps control this, receiving information from everywhere in the network and making all decisions about routing an obvious problem being possible node failures. Another idea would be to distribute the control over the links of the network, with computations and decisions made locally.

To progress with such questions, Kelly considered the following rate of return from a network under the *Erlang fixed-point* approximation to be

$$R(\nu; C) = \sum_r R_r \lambda_r,$$

where $\lambda_r = \nu_r \prod_{k \in r} (1 - E_k)$ where E is the Erlang *fixed-point* defined in §1.4.

He shows that there exist implicit shadow prices associated with each route and with each link of the network, and that the equations defining these prices have a local or decentralised character. He illustrates how his results can be used as the basis for such a decentralised routing scheme, responsive to changes in the demands placed on the network. In the networks considered, both alternative and fixed routing are considered. Kelly's (1988) scheme is adaptive in the sense that it attempts to respond to changes in network form or in arrival rates rather than to seek out and

utilise capacity left idle for very short periods.

A very important result of his work is that the distinction between the *Adaptive* and *Dynamic* approaches is mainly one of emphasis: with sufficiently many levels of priority at each link and a sufficiently broad pattern of alternative routes, *Adaptive* Schemes become *Dynamic*. Kelly's approach contrasts the *Dynamic* approach of Gibbens (1986), Key (1987), Krishnan and Ott (1985, 1986) and Zachary (1988).

1.6 Trunk reservation

As mentioned in §1.1, a *maximal packing* strategy of always accepting every call when there is room to fit it in can be very bad. This is a classical example of the dichotomy between a 'user' optimum and a 'social' optimum where we seek to limit an individual call's freedom in order to maximise the overall performance; see Lippman & Stidham (1977). Ideally we would like to use a flexible dynamic routing control at low loads and turn it off at high loads. Fortunatelly a simple way of doing this exists, *trunk reservation*.

Under *trunk reservation*, a bound m is specified for each link and alternate-routed calls attempting to occupy a circuit on the link are refused if the number of free circuits on the link is below the bound m. Only relatively small values of m, typically less than 10, are needed even for very large circuit quantities.

The advantages of *trunk reservation* were first suggested in the work of J. Weber [see Kelly (1990)], who used a series of simulation studies to examine the effectiveness of various alternate routing schemes. Songhurst (1980) has compared a number of service protection methods and concluded that *trunk reservation* is inherently the most efficient method under a range of traffic load patterns. Akinpelu (1987) has used the *fixed-point* approximation as well as simulation results to show that *trunk reservation* can suppress instabilities under overloads and that such a control allows more efficient use of the circuit for the routes for the one link routes; see §1.1.

Trunk reservation is a very attractive control because its idea is simple and can be easily implemented. Usually it is refered as a control which allows priority to be given to chosen traffic streams.

35

Kelly (1990) extended his work (1988) on routing and capacity allocation in circuit-switched networks to include *trunk reservation* and showed that *fixed-point* methods can apply to it. Such a generalisation is important due to the practical importance of *trunk reservation*. His routing scheme is assumed static over a period. When a call arrives it is routed (or lost) on the basis of a relatively restricted amount of information on the current network state. Under alternative routing for example, the route assigned to an arriving call is determined by information on which links are full or occupied above or below their *trunk reservation* parameters. Using such a simplified control, he shows that there exist implied costs associated with the priority and non-priority traffic through a link. The equations defining these costs have also a local or decentralised character and can be used as a basis for routing or capacity allocation strategies. In this work the routing can be fixed, or alternative and many levels of priority as well as fully connected networks are featured.

Some insight into the efficacy of *trunk reservation* is given by the Markov decision analysis of a single link offered two streams of traffic. Suppose that one of the streams has a priority and generates a larger reward. Suppose also that arrival rates are known and the decision on accepting or rejecting a call depends on the priority level and the history of the link. The aim is to maximise the long-run average expected reward per unit time. Assume also that the holding times are identical and exponential, then it is sufficient to consider policies which summarise the entire history of the link by a single integer, the number of calls currently in progress. But the optimal policy for the Markov decision process is to accept nonpriority calls provided the number of free circuits is above a certain integer. This is a *trunk reservation* policy, and Lippman (1975) has shown that this form of policy remains optimal under a wide variety of discount and finite length of time criteria.

When a single link is offered many types of call, each type with its own reward, the optimal policy is *trunk reservation* with multiple priority levels. The strict optimality of *trunk reservation* does not extend to networks involving more than one link, but is reasonable to expect that such policies will perform well; see Kelly (1990). There is an emphasis that a *trunk reservation* based routing scheme has to take into account *hysterisis* effects; in fact it can provide a method of control as it is known that unrestricted use of alternative routes can severely damage the performance of a network; see §1.2, Ott & Krishnan (1986), Gibbens & Kelly (1990)

and Key (1990).

Key (1990) compares the *trunk reservation* parameter with thinning of the the offered stream. His examples suggest that *trunk reservation* is a robust control and an important one in practice as offered traffic can be varying and imprecisely known. He also mentions cases in which *trunk reservation* is exactly optimal.

1.7 Reduced State-Space Models, Admission Policies and Properties of the Optimal Policy

In this section we briefly present some results of research carried out by Ott & Krishnan (1985, 1986) and Key (1990) on the reduced models. These results are of particular interest for us because they form the background on which we examine our *Admission price* policies and properties of them for both the full and reduced state-space networks of Chapters 2, 3, 4 and 5.

1.7.1 Separable Routing and Cost functions

As mentioned in §1.2, Ott & Krishnan (1985) and Krishnan & Ott (1986) investigated the reduced state-space model by using a state-dependent routing scheme called *Separable Routing* which is obtained by applying the policy iteration procedure of Markov decision process on a nominal routing scheme. Under the assumptions of their model (§1.2), Ott & Krishnan investigate the single link which is offered a stream of Poisson traffic at rate λ and where a reward 1 is earned for each accepted call. The policy iteration procedure is then carried out by considering the value cost of adding a call to the link: if link k has C trunks and is offered a Poisson load of λ erlangs, the 'cost' of adding a call to the link when it already has j calls in progress is given by

$$(1.20) \qquad \Delta(j,k) = \frac{E(\lambda, C)}{E(\lambda, j)}, \quad 0 \leq j \leq C$$

The above relative value/cost is the probability that if the trunkgroup is in state j at time 0 and a call is added, then at least one future call will be blocked on trunkgroup k during the lifetime of the call. $\Delta(j, k)$ is therefore an estimate of the expected increase in future blocked calls on the trunkgroup due to the addition of a call when j calls are already in progress.

Equation (1.20) can be extended to the case of a network with many links. As the links are regarded to be independent, the cost of adding a multi-link call which is worth 1 and uses a single circuit from each of the links $k_1, ..., k_m$, in the respective states $j_1, ..., j_m$, is then given by

$$(1.21) \qquad\qquad W = \sum_{i=1}^{m} \Delta(k_i, j_i).$$

Thus, the cost of a path r is *separable* into the costs of the constituent links, which accounts for the name *Separable Routing*.

Krishnan and Ott's (1985,1986) routing scheme (admission policy) can now be specified as follows: when a call arrives at path r, the cost W, in the current network state, of each admissible path that has at least one free circuit on each link in the path is calculated by the above expressions. If the minimum path cost $W \geq 1$, then the call is rejected; otherwise it is routed on the minimum-cost path.

1.7.2   The Single Link Model

Key (1990, 1994), considers a stochastic loss network with fixed routing patterns, constant Poisson arrivals, and holding times of negative exponential duration with the same mean. The notation used in the description of the Network is that of §1.3.1. The rewards considered are discounted and the target is to maximise the total reward. A reduced state-space is sometimes used, in which the resources are held independently; that is the state space is described by the vector x of the number of different types of calls in progress and the result is an irreversible system.

Key (1994, 1990, 1988) applies the theory of Markov decision processes to a single link network which is offered one or two streams of traffic. For a single link network

of capacity C which is offered Poisson traffic at rate λ he calculates the value function to be

$$(1.22) \qquad W_\xi(j) = \frac{W_\xi(0)}{F_\xi(j)}, \text{ with } W_\xi(0) = F_\xi(C).$$

F is such a function that when the discount factor $\xi \to 0$, $F_\xi(j) \to E(\lambda, j)$; see (1.7). Thus in this case equation (1.22) becomes (1.20), the formula that gives the Ott & Krishan 'cost' function. Key (1990) generalises the above results to a general birth and death process with state-dependent arrivals and holding times.

He then considers the case of a single link which is offered two streams of Poisson traffic, which one is worth more than the other. He uses the results of Lippman (1975) that the optimal policy is *trunk reservation* and investigates the choice of optimal *trunk reservation* parameter. He analyses the problem by starting with a policy that rejects all nonpriority traffic and applies policy improvement to change the *trunk reservation* parameter.

Key (1990) investigates approximations for routing schemes for networks. He considers *admissible controls* where the state-space is reduced to $\Omega \subset \mathcal{S}$. His 'link-based approach' calculates approximate relative values for individual links $W_j(x_j)$ and accepts a multilink call worth R_r which uses links $1, .., m$ on which there are currently $x_1, .., x_m$ calls in progress if

$$(1.23) \qquad W_1(x_1) + ... + W_m(x_m) \leq R_r.$$

where W are calculated by using (1.20), the Ott & Krishnan 'cost' formula. Key (1990) gives sufficient conditions for this simple approximation to be good: He proves a theorem which states that if the call types include all the single link routes on a link j, worth R_j of rate λ_j, n_j is the number of busy circuits on link j and $R_r \leq R_j, \forall r$ that use link j, then

$$W_j(x) \to R_j \frac{E(\lambda_j, C)}{E(\lambda_j, n_j)}, \text{ as } \frac{A_{jr}\lambda_r}{\lambda_j} \to 0.$$

The above relative values are R_j times the expected increase in blocked calls caused by adding a call to the route [compare with (1.20)].

Key extends the Ott and Krishnan Erlang's Loss formula results by proposing a decomposition method to approximate the policy in a large network by using the single link results.

1.8 Properties of the Optimal Policies

As Key (1990) states, the optimal policies in real networks will be constrained by the information available to us. Within the framework of his model the optimal policy - by taking into account the state of the network - is a deterministic one which will reject or accept a call or even send it in an alternative route according to the entire state of the network. Specifying such policies for large networks would be infeasible because of combinatorial explosion, apart from being unrealistic because of the amount of information required. Key's (1990) aim is to understand something of the nature of optimal policies by looking at a single link, and to apply some of this knowledge to obtain good strategies for larger networks.

Key (1990) describes some general properties about the nature of optimal policies, describing *monotonicity* for a certain class of networks.

Key's results hold true for the reduced discounted state-space model in which links are held independently.

For a single link offered different traffic streams, optimal policies are monotonic increasing [see Lippman (1975)], that is, if a call is rejected in state x, then it is optimal to reject the calls in states bigger than x; which proves that optimal policy is of a critical number type. As we later will investigate whether *monotonicity* and other properties for optimal polices hold for our optimal policy, it is necessary to briefly present Key's results in which we will refer to again in Chapter 3.

Let $W(x, r) = V(x) - V(x + e_r)$, where $V = V_\alpha$ is the expected discounted reward. What follows is a summary of Key's results on properties of the optimal policy and can be found in detail in Key (1990).

40

Properties

(I) Theorem 4.1 in Key (1990): If type i calls use fewer resources on each link in the network than type k calls, then for all the discount factors and states x, $V(x + e_i) \geq V(x + e_k)$.

(II) Corollary 4.1 in Key (1990): If type i calls use fewer resources than type k calls, then for all the discount factors and states x, $W(x, i) \leq W(x, k)$.

(III) Assumption A1 in Key (1990): Either type i calls are smaller than type k calls, or routes i and k do not use any common links.

(IV) Assumption A2 in key (1990): Calls of type larger than i are worth less i.e. $R_i \geq R_k$.

(V) Theorem 4.2 in Key (1990): Under the Assumptions A1 and A2 it is always optimal to accept all calls of type i.

(VI) Property P1 in Key (1990): If we reject a type i call in state x, then we reject it in state $x + k$ for calls i and k which are distinct and not disjoint.

(VII) Property P2 in Key (1990): If we reject type j call in $x + e_i$ then we reject it in x for calls i and j which are disjoint. Property P2 means that for calls which are disjoint, and thus could be widely separated in a network, in general, the more type i calls in progress, the less likely we are to reject type j calls, and vice-versa.

He then proves a theorem that shows that Properties P1 and P2 hold for all discount factors for a certain class of networks in which among other things type k calls are monotonic with respect to themselves i.e. if we reject a type k call in x, we reject it in $x + e_k$.

Theorem 4.1, Corollary 4.1 and Theorem 4.2. are all of special interest to us because as we shall see later on we assume that 2-link routes are worth less than the 1-link routes. In Chapter 3, we discuss properties of our optimal policies and see if the results of Key (1990) apply to our networks.

In our work we do not consider *trunk reservation*, admissible controls, alternative routing nor state-dependent arrivals.

Chapter 2

The Model,Definitions,Theorems, Computational Procedures

2.1 Introduction

We consider *Symmetric*, star-shaped, circuit-switched Loss networks which consist of K links of capacity C linked through a common node. Calls requesting routes arrive at the network randomly and according to a Poisson process. There are two types of route: 1-link routes and 2-link routes involving any pair of the single links. Requests for 1-link routes arrive at rate λ_1 on each link and requests for 2-link routes arrive at rate $\lambda_2/(K-1)$ on each 2-link pair. Both types of calls have an exponential holding time with mean 1, (Exp(1)).

2-link routes involving links i and j, where $i < j$ will be denoted by the ordered pair (i,j). As a matter of convenience, we say that 1-link calls on link i relate to pair (i,i).

For every 1-link call carried we earn reward R_1 and for every 2-link call carried we earn reward R_2. Rewards are bounded and earned immediately. In our networks we take R_2 to be less than twice R_1

$$R_2 < 2R_1.$$

The reason for this is that as we are considering the case where all 1-link routes are accepted, we think that it is reasonable to take $R_2 < 2R_1$; it is our assumption.

2.2 The Network as a Markov Decision Process

We consider the operation of the network as a process which, when observed in time, is in one of a number of states. The set of all possible states S is finite. Denote a typical state by $z \in S$.

The exponential holding times of accepted calls make the sequence of states z_t, $t \geq 0$, a continuous time Markov process. At arrival points, and after observing the state of the process, an action a must be chosen which accepts or rejects arrivals. To accept or reject arrivals we must follow some policy π. A policy π is any rule for choosing actions a. In our case, a rule for accepting or rejecting arrivals. In this work we consider policies which are non-randomised and choose actions depending on the state of the process at that time and are mappings $\pi : S \to A$, where S is the state space and A is the action space. Such policies are called *stationary*. *Stationary* policies are simple and Blackwell (1965) shows that it is not necessary to consider more complicated time-dependent policies; which is a great advantage in case where one is analysing a Markov process over an infinite length of time.

The times between consecutive decision epochs are not identical but are exponentially distributed with state-dependent mean. If the process is in state z at time t and policy $\pi(z)$ is chosen, then independent of the past, two things occur:

a) We receive a reward $R(z, \pi)$ immediately if an arrival is accepted.

b) The next state of the system is chosen according to the transition probabilities $P_\pi(z)$.

After *uniformisation* [Lippman (1975)], a procedure in which **null** trnsitions are introduced into the system in an appropriate state-dependent fashion, the continuous

time Markov decision process can be represented by an equivalent discrete time Markov decision process z in which the rate of transition between states is constant. Under *uniformisation* the times between transitions are independent of not only the control policy employed, but also the state of the process. The purpose of *uniformisation* is to enable us to replace the complex differential equation form of the *optimality equation* with a simpler iterative equation as we shall see later on. Note that both rewards earned and the transition probabilities are functions only of the last state and the subsequent action. *Uniformisation* is discussed in §2.3.

In investigating the performance of different policies a key consideration is *optimality*. The optimality criterion we use in this work is the **Total Expected Discounted Return (TEDR)** as defined in Ross (1983) and the goal is to exhibit optimal policies that can be implemented in the network and maximise the TEDR.

The above criterion assumes a discount factor α, $0 \leq \alpha < 1$, and among all policies π, attempts to maximise the TEDR

$$(2.1) \qquad V_\pi(z) = E_\pi \left[\sum_{t=0}^{\infty} R(z,\pi)_t \alpha^t | z(0) = z \right]$$

where E_π represents the conditional expectation given that policy π is employed; $V_\pi(z)$ is well defined [Ross, 1983]. (2.1) represents the TEDR after *uniformisation*.

The discount factor α is introduced because a reward to be earned in the future is less valuable than one earned today. This way we also ensure that the total reward is bounded.

As mentioned in §1.3, Blackwell (1965), shows that if the action space $\mathcal{S}$ and the state-space $\mathcal{A}$ are finite there is an optimal π^* such that, for every π, $V_{\pi^*}(z) \geq V_\pi(z)$, $\forall z \in \mathcal{S}$, in the set of all possible states. He also shows that if there is an optimal π, there is one which is stationary.

If we define $V(z)$ to be

$$V(z) = \sup_\pi V_\pi(z)$$

then a policy π^* is said to be α-optimal (or optimal) if

$$(2.2) \qquad V_{\pi^*}(z) = V(z), \quad \forall z \in \mathcal{S}$$

An optimal policy has the property that whatever the initial state and initial action are, the remaining actions must constitute an optimal policy with regard to the state resulting from the first transition - a principle being always valid when the number of states and the number of actions is finite. Now we can use Dynamic Programming results to calculate the optimal policy by considering an *optimality equation*; see §1.3.3.

In our work the *optimality equation* is given by the following expression

$$(2.3) \qquad V(z) = \max_{\pi} \left[E[R(z,\pi)] + \alpha \sum_{\bar{z}} P_{\pi}(\bar{z}|z)V(\bar{z}) \right]$$

where $z, \bar{z} \in \mathcal{S}$. [A proof can be found in Ross (1983)].

The *optimality equation* (derived by considering the network as a Markov decision process) will be our tool for the analysis of such problems for the main reason that under *uniformisation* it becomes a linear equation which can be solved recursively by Gauss-Seidel or some such iterative procedure; the iterative procedures we are using are presented in §2.4.

Proposition 1

$\forall$ stationary policies π, V_{π} is the unique solution of

$$(2.4) \qquad V_{\pi}(z) = E[R(z,\pi)] + \alpha \sum_{\bar{z}} P_{\pi}(\bar{z}|z)V_{\pi}(\bar{z})$$

Proof: A proof can be found in Ross (1983).

45

Let g be the stationary policy that when the process is in state z, selects the action which maximises the right hand side of (2.4), then

$$V_g(z) = V(z), \quad \forall z \geq 0$$

and hence g is α-optimal.

Proof: A proof can be found in Ross (1983).

Before we discuss the actual procedures for calculating our optimal policies, we give definitions of the parameters used to describe the network and the rates in which events happen as well as the transition probablities.

2.3   Definition of the Model

2.3.1   Preliminary Definitions

Definition 1

The number of all possible pairs (i, j), where $i \leq j$, is denoted by β, where

$$\beta = \frac{K(K+1)}{2}, \quad i, j = 1, 2, ..., K$$

Definition 2

The number of all possible 2-link pairs (i, j), where $i < j$ is denoted by γ, where

$$\gamma = \frac{K(K-1)}{2}$$

where $1 \le i < j \le K$.

<div align="center">

Lemma   1

</div>

We can index all the 2-link pairs (i,j) with the formula

$$l(i,j) = \frac{i(2K-1-i)}{2} - K + j$$

where $1 \le i < j \le K$.

Proof: To derive the above expression for $l(i,j)$ write out the pairs as follows

<div align="center">

(1,2), (1,3), ..., (1,K)

(2,3), (2,4), ..., (2,K)

....

(K-2,K-1), (K-2,K)

(K-1,K)

</div>

In the first row $l(1,j) = j - 1$; in the second $l(2,j) = K - 1 + j - 2$; and in row i

$$l(i,j) = (\sum_{n=K-i+1}^{K-1} n) + j - i$$

which can be rearranged to give the formula of $l(i,j)$.

<div align="center">

Definition   3

</div>

The number of 1-link calls on link i is denoted by x_i, where $0 \le x_i \le C$.

The number of 2-link calls on pair (i,j) with index number $l(i,j)$ is denoted by $w_{l(i,j)}$, where $0 \leq w_{l(i,j)} \leq C$.

<div align="center">Definition   4</div>

The number of 1-link calls in the network is denoted by x, where

$$x = (x_1, x_2, ..., x_K)$$

The number of 2-link calls in the network is denoted by w, where

$$w = (w_1, .., w_\gamma)$$

<div align="center">Definition   5</div>

The state of the network at the time we observe it is denoted by (x, w), where

$$(x, w) = (x_1, .., x_K; w_1, .., w_\gamma)$$

For each i,

$$0 \leq x_i + \sum_{l \in A_i} w_l \leq C$$

where $A_i = \{l : l = l(i,j) \text{ or } l = l(j,i), \text{ some } j \neq i\}$.

We say that the link i is *full* when the number of calls of both types in i is C. 1-link calls depart from link i at rate x_i; and 2-link calls depart from pair (i,j) at rate w_l.

2.3.2 Rates of Events

1-link calls arrive on the network at rate ν_1, where

$$\nu_1 = K\lambda_1$$

2-link calls arrive on the network at rate ν_2, where

$$\nu_2 = \frac{\gamma\lambda_2}{(K-1)} = \frac{K\lambda_2}{2}$$

1-link calls depart from the network at rate ν_3, where

$$\nu_3 = \sum_{i=1}^{K} x_i$$

2-link calls depart from the network at rate ν_4, where

$$\nu_4 = \sum_{l=1}^{\gamma} w_l$$

Under the framework of our network the transitions to the next state are exponentially distributed, but with various rates. To allow transitions to occur at a uniform rate we used the technique of *uniformisation*, introduced by Lippman (1975), in which transitions occur from each state by introducing fictitious **null events**, that is transitions from a state to itself [Lippman (1975), Lippman and Stidham (1977), Ross (1985)]. This enables us to replace the continuous time problem with a discrete time problem.

The rate of **null events** in the network is denoted by ν_5, where

$$\nu_5 = KC - \nu_3 - \nu_4$$

Note that when the system is **full** i.e. there are no free links, we cannot accept arrivals of any type.

The **Total Rate of events** in the network is denoted by *Rate*, where

$$(2.5) \qquad Rate = \sum_{i=1}^{5} \nu_i = K(\lambda_1 + \frac{\lambda_2}{2} + C)$$

The rate of offered traffic per link over the network is denoted by L, where

$$(2.6) \qquad L = \lambda_1 + (K-1)\frac{\lambda_2}{2(K-1)} = \lambda_1 + \frac{\lambda_2}{2}.$$

It is also necessary to add that now that the transitions in the network occur at rate (2.5) the correction between discrete steps in the *optimality equation* after *uniformisation* and 'time' in the continuous time process for the discount factor α is

$$\overline{\alpha} = \frac{Rate}{Rate + (-\ln \alpha)}.$$

Remark: In most cases examined in this thesis $\alpha \approx 0.8$. All the results for a fixed L (e.g. 2.95) are comparable by having the same discount rate but different values of L yield slightly different actual discount value. The results presented in tables in Chapters 4, 5 and 6 have values calculated by the approximation $\overline{\alpha} = \alpha^{1/Rate}$ and are about 0.6% smaller than the results that would be derived by using the above correction.This, however, has no effect in the optimal policies and the differences between optimal $V(0)$ and the *Admission Price* optimal $V_w(0)$; see Chapter 4.

The results on this and later chapters are usually grouped by the value of L for 'historical reasons' - when running the programs this keeps the rate of events the same for a collection of different arrival rates for networks of given size.

Notation

$e_i = \underbrace{(0, .., 1, .., 0)}_{K}$, with 1 in the ith place.

50

$$e_l = \underbrace{(0, .., 1, .., 0)}_{\gamma}, \text{ with 1 in the } l^{th} \text{ place.}$$

Possible transitions: $\pm e_i$, $i = 1, 2, .., K.$, and $\pm e_l$, $l = 1, 2, .., \gamma$.

2.3.3 Transition Probabilities

The transition probabilities for the network are as follows:

$$P_\pi(x + e_i, w | x, w) = \begin{cases} 0, & \text{if } \pi(x, w) = \text{reject} \\ \frac{\lambda_1}{Rate}, & \text{otherwise} \end{cases}$$

$$P_\pi(x - e_i, w | x, w) = \frac{x_i}{Rate}, \quad \text{if } x_i \geq 0$$

$$P_\pi(x, w + e_l | x, w) = \begin{cases} 0, & \text{if } \pi(x, w) = \text{reject} \\ \frac{\lambda_2}{(K-1)} \frac{1}{Rate}, & \text{otherwise} \end{cases}$$

$$P_\pi(x, w - e_l | x, w) = \frac{w_l}{Rate}, \quad \text{if } w_l \geq 0.$$

$$P_\pi(x, w | x, w) = \frac{\nu_5}{Rate}$$

As we always accept 1-link calls when there is room to fit them in

$$P_\pi(x + e_i, w | x, w) = \begin{cases} 0, & \text{if } x_i = C \\ \frac{\lambda_1}{Rate}, & \text{otherwise} \end{cases}$$

$$\underline{Definition \quad 7}$$

The reward to be expected in the next transition out of state (x, w) when policy π

51

chooses action a is denoted by Ψ, where

$$
(2.7) \qquad \Psi = E\left[R((x,w),\pi)\right] = \sum_{a \in A} P_\pi(x,w))R(x,w)
$$

$$
= \sum_{i=1}^{K} P_\pi(x + e_i, w | x, w) R1 +
$$

$$
\sum_{l=1}^{\gamma} P_\pi(x, w + e_l | x, w) R2.
$$

Ψ can also be written as

$$
(2.8) \qquad \Psi = \frac{R1\ \lambda_1\ \sum_{i=1}^{K} I(e_i | x, w)\ + \frac{R2\ \lambda_2}{K-1} \sum_{l=1}^{\gamma} I(e_l | x, w)}{Rate},
$$

where $I(e_l | x, w) = 1$ or 0 according as a call on route l is accepted or rejected by the admission policy.

Using (2.7), (2.4) can be expressed as

$$
V_\pi(x,w) = \Psi + \overline{\alpha} \sum_{i=1}^{K} P_\pi(x \pm e_i, w | x, w) V_\pi(x \pm e_i, w) +
$$

$$
\overline{\alpha} \sum_{l=1}^{\gamma} P_\pi(x, w \pm e_l | x, w) V_\pi(x, w \pm e_l) +
$$

$$
(2.9) \qquad \overline{\alpha}[P_\pi(x, w | x, w)] V_\pi(x, w).
$$

2.4 Computational Procedures for the Optimal Policy

The problem of formulating optimal policies for Loss Networks has been a subject of research for the last 20 years. In general the optimal policy and the value functions cannot be determined for realistic size networks which comprise a number of resources and where different call-types share resources, although it is possible to

determine controls that are asymptotically optimal in a suitable defined sense. The two factors which make the problem of formulating the optimal policy so hard are the size of the state-space and the fact that, in order to compare the relative desirability of two different routes, the state of trunkgroups not in either of these routes may be relevant.

It has been shown by Key (1990) that the optimal policies can be complicated to describe and do not share properties possessed by optimal policies for the simple case of a single link. We will also show how complex the optimal policies are even in our simple networks in Chapter 3.

2.4.1 Policy Improvement

Theorem 1 suggests that, once we have determined the optimal function V, then we would know the optimal policy - it would be the stationary policy π that, when in state $z = (x, w)$, maximises

$$E[R(z, \pi)] + \overline{\alpha} \sum_{\overline{z}} P_\pi(\overline{z}|z) V_g(\overline{z})$$

Suppose now that for some stationary policy g we have computed V_g, the expected return under g; and suppose that we now define h to be the policy that, when in state z, selects the action a that maximises the above. The question is how good is h compared with g. This describes Howard's policy improvement procedure and Ross (1983) contains a proof that h is at least as good as g and if it is not strictly better than g for at least one initial state, then g and h are both optimal. In our work we apply policy improvement on a computer to obtain the optimal policy when the state space is finite.

Step 0 (Initialisation). Choose any stationary policy g and employ it in our Network as a rule for rejecting calls.

Step 1 (Value Determination). Compute V_g as the unique solution of the set of equations

$$(2.10) \qquad V_g(z) = E[R(z,g)] + \overline{\alpha} \sum_{\overline{z}} P_g(\overline{z}|z) V_g(\overline{z})$$

Step 2 (Policy Improvement). Determine a policy h as the policy that, for each state $z \in \mathcal{S}$, selects the action which maximises the right hand side of (2.10).

Step 3 (termination). If the new policy h equals the old policy g, the algorithm is stopped with policy h as the optimal. Otherwise, go to step 1 with g replaced by h.

The policy improvement algorithm converges after a finite number of iterations; a proof can be found in Tijms (1988). It is empirically found that policy improvement is a remarkably robust algorithm which converges very fast in specific problems. The number of iterations is practically independent of the number of states and of the starting policy, and varies typically between 3 and 15 (see Tijms). In our examples (see §2.5) it never exceeded 5. In Tijms (1988) it is also stated that the expected return of the policies generated by policy improvement converge at least exponentially fast to the maximum return.

Note: Policy improvement is usually more complicated than *value iteration*. However in our case it is only slightly more complicated and as we can store the optimal policy compactly we use policy improvement which allows us to get direct access to the optimal policy itself and not only the maximum expected discounted reward.

This is the right place to add that we initially found a method for storing policies for controlling the 2-link calls with the intention of treating the more general case later (i.e. possible rejection of calls of either type), but ran out of time; see §3.

2.4.2   The Value Determination Step

Step 2 of the policy improvement algorithm calculates recursively the expected returns V_π when a policy π is employed. The algorithm used to do this is as follows

Step 0. Choose $V_0(z) \geq 0$ be any arbitrary bounded function for all z. Let $n := 1$, and employ a policy π.

Step 1. Compute the unique solution to (2.10) using the following system of linear equations for all $z \in \mathcal{S}$ under the policy considered

$$(2.11) \qquad V_n(z) = [E(R(z,\pi)) + \overline{\alpha} \sum_{\overline{z}} P_\pi(\overline{z}|z) V_{n-1}(\overline{z})]$$

recursively

Step2. The algorithm is stopped when

$$\max_{z \in \mathcal{S}} |V_n(z) - V_{n-1}(z)| \leq \epsilon,$$

where ϵ is a prespecified tolerance number. Otherwise, go to step 3.

Step 3. $n := n + 1$ and go to step 1.

This algorithm is computing recursively a sequence of value functions approximating the expected discounted returns (TEDR) per unit time.

Equation (2.11) can be expressed in vector form as

$$(2.12) \qquad \mathbf{V_n} = \mathbf{q} + \mathbf{TV_{n-1}}, \quad n = 1, 2, ..$$

where $\mathbf{V_n}$ is a column vector with $|\mathcal{S}|$ components $V_n(x, w)$.

If we rearrange (2.12) the right way, the linear iteration scheme resulting can be solved by the Gauss-Seidel (G-S) iteration method for solving a linear system $\mathbf{A} \ \mathbf{V} = \mathbf{R}$, where $\mathbf{A}$ is been split into its diagonal $\mathbf{D}$, strictly lower-triangular part $\mathbf{L}$ and strictly upper-triangular part $\mathbf{U}$ so that

$$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{U}.$$

Under this assumption, the Gauss-Seidel scheme is represented by (2.12) where

$$\mathbf{T} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}.$$

$\mathbf{T}$ is called the Gauss-Seidel Iteration Matrix.

The point about using iterative techniques is that, storing the elements of matrix $\mathbf{A}$ is impracticable as the size of it will be $|\mathcal{S}|^2$; see also §1.4. That's why iterative techniques for solving the value determination step are an extremely important tool in the analysis of Markov decision processes which have a large state space as it involves solving a set of linear equations whose size is dependent on the state space, and other approaches such as the linear programming one suffer even more from complexity considerations.

The policy improvement with value determination algorithm's greatest benefit is that it can be directly implemented on a computer; see Appendix B.2 and B.3.

2.4.3 The Successive Over-Relaxation Algorithm (SOR)

The convergence speed of the value determination algorithm may be accelerated by using a relaxation factor such as in successive overrelaxation (SOR) for solving a single system of linear equations. Then at the nth iteration, a new approximation to the value function $V_n(z)$ is obtained by using both the previous values $V_{n-1}(z)$ and the residuals $V_n(z) - V_{n-1}(z)$.

The following modification of the value determination can be formulated as follows: The steps 0,1 are as before, while steps 2 and 3 become

Step 2. $\forall z \in \mathcal{S}$, change $V_n(z)$ according to

$$V_n(z) \quad = \quad (1 - \omega)V_{n-1}(z) \quad +$$

$$(2.13) \qquad \omega \left[E(R(z,\pi)) \ + \ \overline{\alpha} \sum_{\overline{z}} P_\pi(\overline{z}|z) V_{n-1}(\overline{z}) \right],$$

where $z, \overline{z} \in \mathcal{S}$.

Step 3. If the algorithm is stopped, the process is finished. Otherwise, $n := n + 1$ and go to step 1.

The SOR value determination method can also be expressed in vector form as

$$\mathbf{V_n} = \mathbf{q} + \mathbf{T}\mathbf{V_{n-1}}, \quad n = 1, 2, ..$$

but this time the matrix $\mathbf{T}$ is

$$(2.14) \qquad \mathbf{T} = (\mathbf{D} - \omega \mathbf{L})^{-1}[(\mathbf{1} - \omega)\mathbf{D} + \omega \mathbf{U}],$$

where $\mathbf{D}, \mathbf{L}$ and $\mathbf{U}$ are the matrices defined in §2.4.2 associated with matrix $\mathbf{A}$.

Note, that the iteration method with $\omega = 1$ is the Gauss-Seidel method described earlier on.

The convergence speed of the successive overrelaxation may dramatically depend on the choice of the relaxation factor ω, and even worse, the method may diverge for some choices of ω. A suitable value of ω has to be determined experimentally usually $1 \leq \omega \leq 2$.

2.5 The Efficiency of the SOR method

In our work we have tried different over-relaxation (SOR) values for ω and the optimal results in convergence speed (the total number of iterations needed) suggest that optimal ω are different for problems of different size. As we shall see later from the figures, it seems that the variation is associated more with the change in offered traffic rates than the network size; see especially Figures 2.5 and 2.6.

The next six figures show some results for *Symmetric* networks of sizes $(K, C) = (3, 3)$ and $(K, C) = (4, 3)$ for both the full model described in this Chapter as well as the reduced state-space model described in Chapter 4.

In Figures 2.5 and 2.6, $K = 3$ and $C = 3, 4, 5, 6$.

The arrival rates in Figures 2.1, 2.2, 2.3 and 2.4 are (λ_1, λ_2):

(a) (0.5;4.9) and (2;2) for the full *Symmetric* networks of both sizes.

(b) (0.5;4.9) and (2;4) for the reduced state-space *Symmetric* networks of both sizes.

In Figures 2.5 and 2.6 the arrival rates are (0.5,4.9) and (2,4) respectively.

For theses examples $R_1 = 2$ and $R_2 = 1$. In the graphs $n1 = \lambda_1$ and $n2 = \lambda_2$.
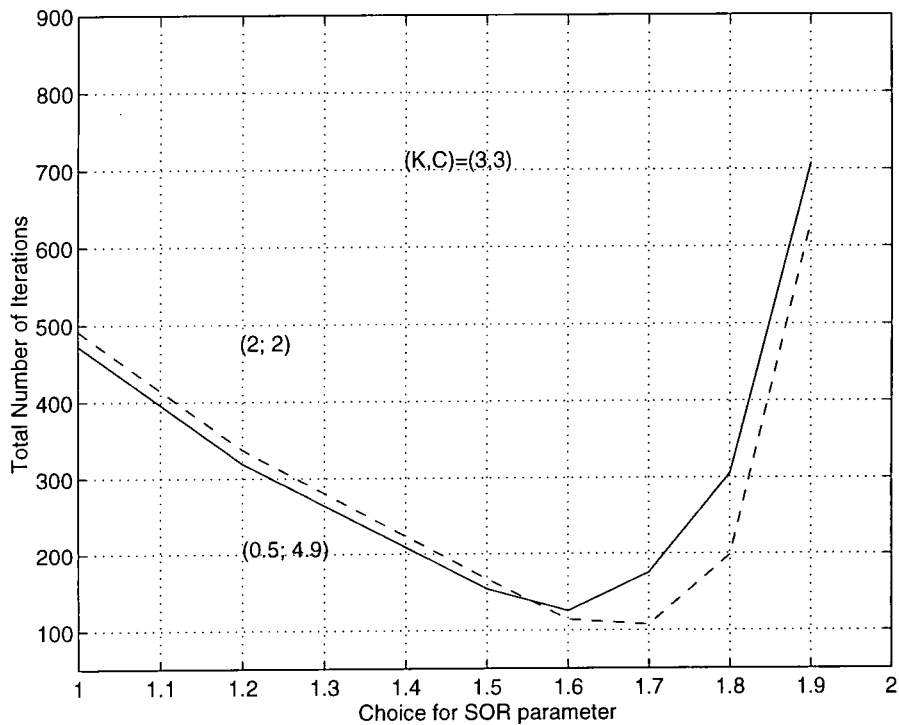
Figure 2.1: The effect of ω on the no. of iterations for the full *Symmetric* network $(K, C) = (3, 3)$.
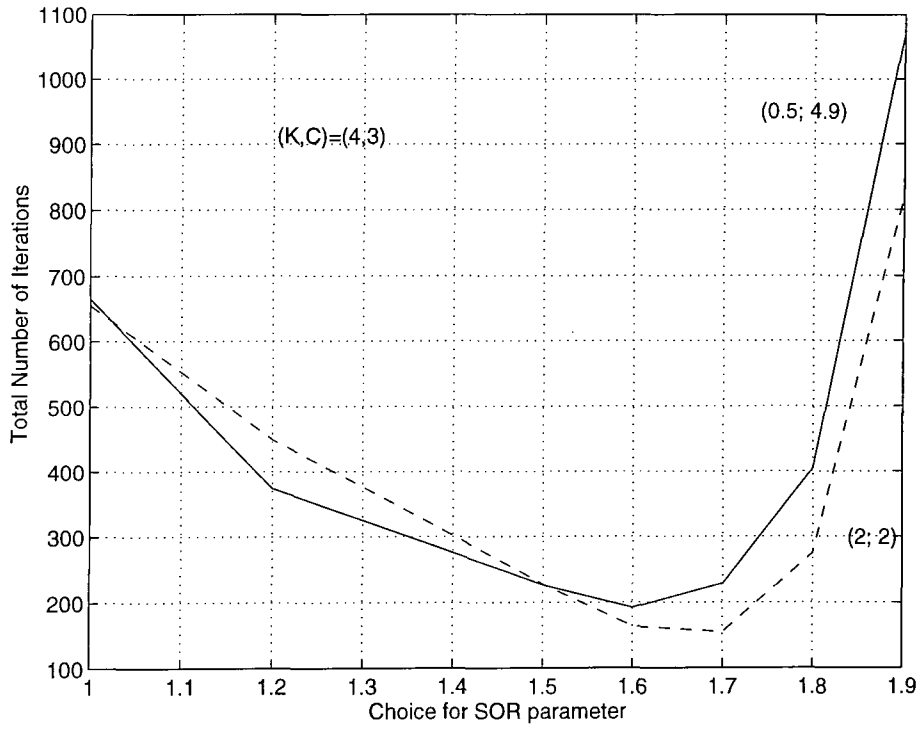
Figure 2.2: The effect of ω on the no. of iterations for the full *Symmetric* network $(K, C) = (4, 3)$.
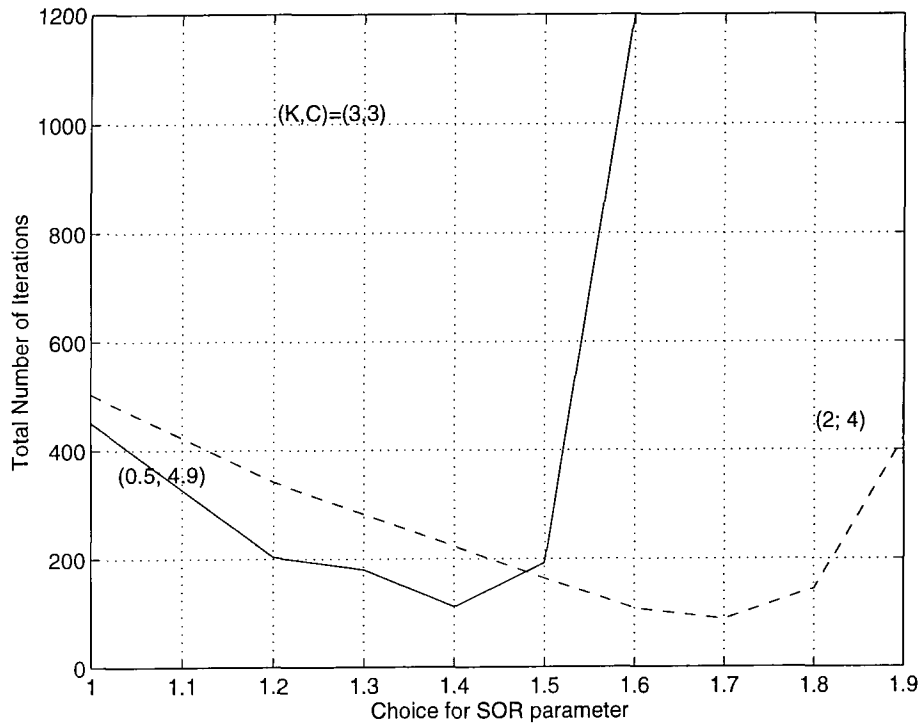
Figure 2.3: The effect of ω on the no. of iterations for the reduced state-space *Symmetric* network $(K, C) = (3, 3)$.
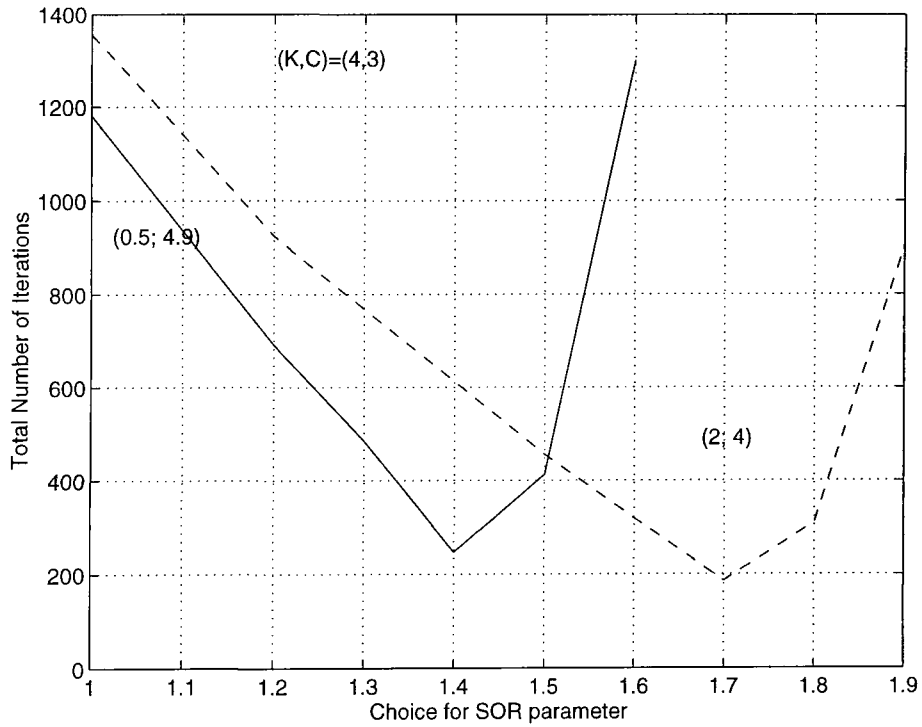
Figure 2.4: The effect of ω on the no. of iterations for the reduced state-space *Symmetric* network $(K, C) = (4, 3)$.
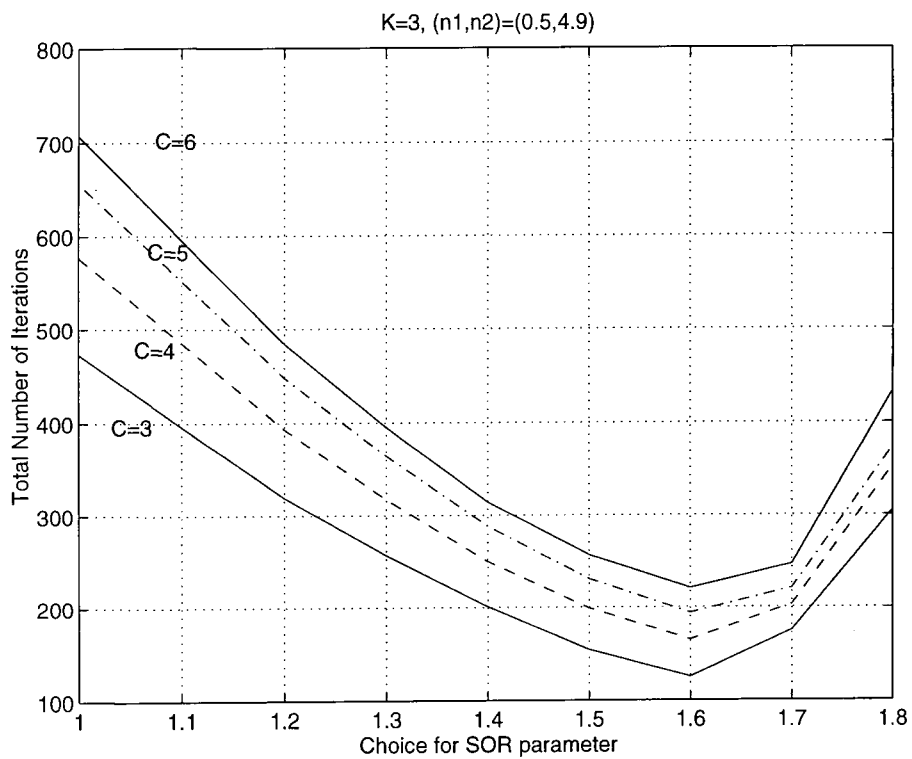
Figure 2.5: The effect of ω on the no. of iterations for the full *Symmetric* networks with $\lambda_1 = 0.5, \lambda_2 = 4.9$, $K = 3$ and $C = 3, 4, 5, 6$.
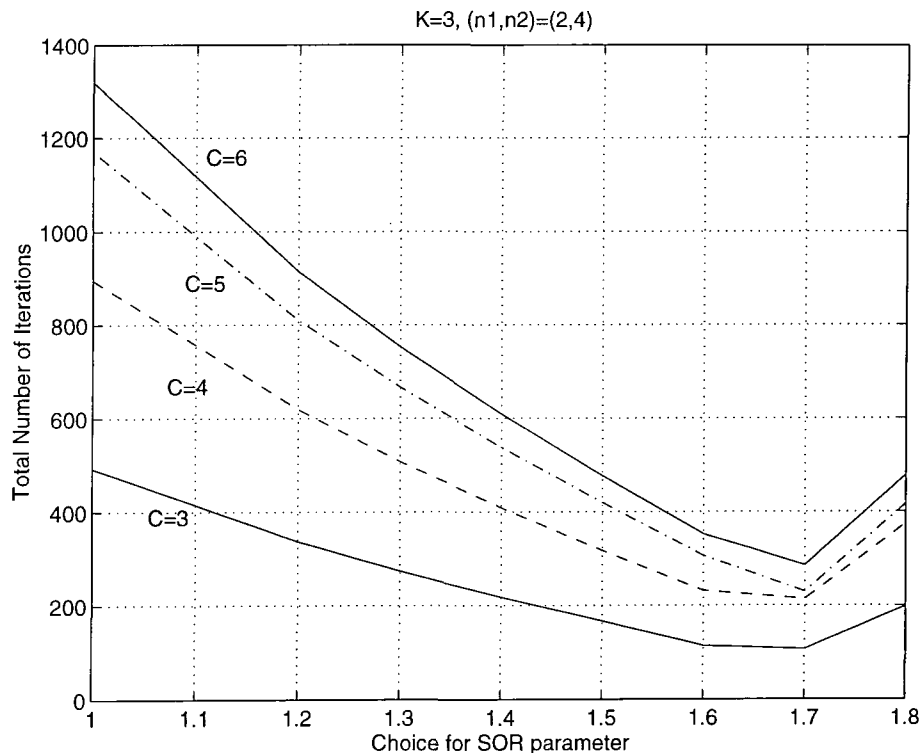
Figure 2.6: The effect of ω on the no. of iterations for the full *Symmetric* network with $\lambda_1 = 2, \lambda_2 = 4$, $K = 3$ and $C = 3, 4, 5, 6$.

The actual convergence of these schemes as well as arguments on the error analysis in relation to the numerical evaluation of the TEDR for small *Star Symmetric* networks are examined in detail in the next section.

2.6 Convergence Analysis for SOR

In previous sections we presented the Gauss-Seidel (G-S) and the SOR schemes, as the value determination algorithms adopted for the numerical calculation of the TEDR. We know that the Gauss-Seidel method converges because there is an optimal discounted reward (the problem has finite state and action space); see Ross (1983) and Tijms (1988). It is now time to show, using Numerical and Matrix Analysis results as well as data from the study-cases, that the SOR scheme converges. We will also derive results for possible rounding and truncation errors in our calculations.

Before we do so, it is necessary to include some theory and definitions from Numerical

Analysis.

The *spectral radius* $\rho(\mathbf{A})$ of a matrix $\mathbf{A}$ is defined by

$$\rho(\mathbf{A}) = \max|\lambda|,$$

where λ is an eigenvalue of $\mathbf{A}$.

The *spectral radius* of a matrix is closely related to the norm of the matrix as shown in the following theorem.

Theorem 2

If $\mathbf{A}$ is an $n \times n$ matrix, then

$\rho(\mathbf{A}) \leq ||\mathbf{A}||$ for any natural norm $|| * ||$.

Proof: Proofs can be found in Burden and Faires (1989) or Ortega (1972).

Corollary 1

If $||\mathbf{T}|| \leq 1$ for any natural matrix norm, then the sequence $\{\mathbf{V_n}\}_{n=0}^{\infty}$ in $\mathbf{V_n} = \mathbf{q} + \mathbf{TV_{n-1}}$, $n = 1, 2, ..$ converges for any $\mathbf{V_0} \in R^n$ to a vector $\mathbf{V} \in R^n$, and the following error bound holds

$$||\mathbf{V} - \mathbf{V_n}|| \leq ||\mathbf{T}||^n \, ||\mathbf{V_0} - \mathbf{V}||.$$

Therefore a necessary and sufficient condition for convergence is that $\rho(\mathbf{T}) < \mathbf{1}$.

Proof: A proof can be found in Burden and Faires (1989); see also Hageman and Young (1981).

The relationship of the rapidity of convergence to the *spectral radius* of the iteration matrix $\mathbf{T}$ can be clearly seen from Corollary 1. Since the Corollary holds for any natural norm, it follows from Theorem 2 that

$$(2.15) \qquad ||\mathbf{V_n} - \mathbf{V}|| \approx \rho(\mathbf{T})^n \, ||\mathbf{V_0} - \mathbf{V}||.$$

Thus it is desirable to select an iterative technique with minimal $\rho(\mathbf{T}) < 1$ for the solution of the associated linear system. The point about SOR with the best choice of ω is that its associated matrix $\mathbf{T}$ has minimal spectral radius.

Kahan (1958) has shown that if $a_{ii} \neq 0$ for each $i = 1, 2, .., n$, then

$$\rho(\mathbf{T_{SOR}}) \geq |\omega - 1|,$$

with equality possible only if all eigenvalues of $\rho(\mathbf{T_{SOR}})$ have modulus $|\omega - 1|$. Thus a necessary condition for the convergence of the SOR method is that $0 < \omega < 2$.

In our examples we have calculated $\rho(\mathbf{T})$ numerically on a computer for $0 < \omega < 2$ and it is indeed always < 1. That is an assurance that the SOR method converges; see §2.7.

Another way to approach convergence is the Ostrowski - Reich theorem which states that if $\mathbf{A}$ is a positive definite matrix and $0 < \omega < 2$, then the SOR method converges for any choice of initial approximate solution vector $\mathbf{V_0}$. Our matrix $\mathbf{A}$ is positive definite as it is diagonally dominant i.e. $|a_{ii}| \geq \sum_{j \neq j, j=1}^{n} |a_{ij}|$, $i = 1, 2, .., n$.

Definition   9

The *condition number* $k(\mathbf{A})$ of a matrix $\mathbf{A}$ is defined by

$$k(\mathbf{A}) = ||\mathbf{A}|| \, ||\mathbf{A}^{-1}||,$$

for any natural norm $|| * ||$.

63

The condition number of a matrix measures the sensitivity of the solution of a system of linear equations to errors in data. As we shall see in §2.7, $k(\mathbf{A})$ gives an indication of the accuracy of the results from matrix inversion and linear equation solution.

In this work the norm we are using as a measure of the magnitude of the elements of the matrix $\mathbf{A}$ is the infinity norm which is defined by $||\mathbf{A}||_\infty = \max_i \sum_j |A_{ij}|$, the largest row sum of $\mathbf{A}$.

2.7   Errors in Computational Procedures

In this section we consider an analysis on the accuracy of the results and possible errors due to (a) truncation[1] in the iterative procedures considered (SOR), and (b) rounding errors in Gauss Elimination.

To study the problem of truncating/rounding errors in calculating the TEDR values, and apart from a theoretical approach, we have compared the TEDR values starting from different initial values for the iteration and keeping the optimal policy fixed. This comparison showed that any differences are due to rounding errors.

In numerically calculating the maximal TEDR by either the G-S or the SOR method, the algorithm considered is given by

$$\mathbf{V_n} = \mathbf{q} + \mathbf{TV_{n-1}}, \quad n = 1, 2, ..$$

where $\mathbf{V_n}$ is a column vector with K components $V_n(x, w)$ and $\mathbf{T}$ is the Iteration Matrix (see §2.4). This algorithm recursively computes a sequence of value functions V approximating the maximal TEDR.

The Gauss Elimination solves $\mathbf{AV} = \mathbf{R}$.

---

[1]See §2.4.2

2.7.1 Truncation Error

Suppose that the real maximal TEDR is denoted by V^* and that a sequence satisfies $V_0, V_1, \ldots \to V^*$ and $|V_{n+1} - V^*| < \xi|V_n - V^*|$ for some $\xi < 1$. From (2.15) a suitable value of ξ is $\rho(\mathbf{T})$. The value determination algorithm approximates V^* by V_N where we choose N to be sure that $|V_N - V_{N+1}| < \epsilon$. A natural question arises now is to find out how big the distance $|V_N - V^*|$ is. Applying the triangle equality we see that

$$
\begin{aligned}
|V_N - V^*| &\leq |V_N - V_{N+1}| + |V_{N+1} - V^*| \\
&\leq \epsilon + \xi|V_N - V^*|
\end{aligned}
$$

and hence

$$
(2.16) \qquad\qquad |V_N - V^*| \leq \frac{\epsilon}{1 - \xi} \, .
$$

Knowing ϵ to be the algorithm tolerance number it remains to calculate the number ξ. From the plausible approximation

$$
|V_{N+1} - V_N| \approx \xi^N |V_1 - V_0|
$$

we get

$$
\xi \approx (\epsilon/|V_1 - V_0|)^{1/N} \, .
$$

2.7.2 Rounding Errors

We know from Numerical Analysis that in solving $\mathbf{AV} = \mathbf{R}$ a small value of the residual $\mathbf{r} = \mathbf{R} - \mathbf{AV}$ does not necessarily imply that $\|\mathbf{V} - \mathbf{V}^*\|$ will be small as well. It is also known that

$$\|\mathbf{V} - \mathbf{V}^*\| \leq \|\mathbf{r}\| \, \|\mathbf{A}^{-1}\|$$

[Theorem 8.19, Burden and Faires (1989)]. Forsythe and Moler (1967) also show that the residual vector $\mathbf{r}$ for the approximation of $\mathbf{V}^*$ has the property that

$$\|\mathbf{r}\| \approx 10^{-t} \, \|\mathbf{A}\| \, \|\mathbf{V}^*\|,$$

where the solution of $\mathbf{AV} = \mathbf{R}$ is being determined using t-digit arithmetic and Gaussian Elimination.

If we consider $\mathbf{Y}^*$ the approximate solution of $\mathbf{AY} = \mathbf{r}$ then,

$$\mathbf{Y}^* \approx \mathbf{A}^{-1}\mathbf{r} = \mathbf{A}^{-1}(\mathbf{R} - \mathbf{AV}^*) = \mathbf{A}^{-1}\mathbf{R} - \mathbf{A}^{-1}\mathbf{AV}^* = \mathbf{V} - \mathbf{V}^*.$$

so $\mathbf{Y}^*$ is an approximate of the error in approximating the solution to the original system. From Forsythe and Moler (1967) we can now deduce that

$$(2.17) \qquad \begin{aligned} \|\mathbf{Y}^*\| \approx \|\mathbf{V} - \mathbf{V}^*\| &= \|\mathbf{A}^{-1}\mathbf{r}\| \\ &\leq \|\mathbf{A}^{-1}\| \, \|\mathbf{r}\| \approx 10^{-t} k(\mathbf{A})\|\mathbf{V}^*\|. \end{aligned}$$

The above means that it is easy to deduce an approximation for the rounding error involved if we can calculate the condition number of the matrix $\mathbf{A}$ and have the solution $\mathbf{V}^*$ stored.

Gauss Elimination and the calculation of the condition number is possible for the smallest of our examples. We can use these to test the rest of the algorithms for the smallest cases. Particularly we can compare $\rho(\mathbf{T})$ with the approximation $\xi \approx (\epsilon/|V_1 - V_0|)^{1/N}$.

For larger cases we have to use SOR; see Case V in §2.7.3.

2.7.3 Numerical Examples on Errors

In the following examples of networks we present firstly the results for the approximation of errors as deduced in practice from (2.16) and secondly as they are deduced by the approximation (2.17) applying MATLAB Gauss Elimination with 13-digit arithmetic to the problems.

In particular in the following examples we calculate: (i) the condition number, (ii) the spectral radius $\rho(\mathbf{T})$, (iii) the approximation $\xi \approx (\epsilon/|V_1 - V_0|)^{1/N}$ and compare with $\rho(\mathbf{T})$. The tolerance number in the following examples is $\epsilon = 10^{-6}$.

Case I: Consider a network with $(K, C) = (2, 3)$, $(R_1, R_2) = (2, 1)$ and $(\lambda_1, \lambda_2) = (0.5, 4.9)$.

Applying policy improvement, the value determination takes $N = 45+31$ iterations to calculate the TEDR with the improved policy and therefore using the approximation (2.16) with $\epsilon = 10^{-6}$ and $\xi = 0.67828$ we get

$$\frac{\epsilon}{1-\xi} = 3.11 \times 10^{-6}.$$

$\rho(\mathbf{T}) = 0.7122 \ (\approx \xi)$.

The condition matrix for the above example calculated using MATLAB Matrix Operations is $k(\mathbf{A}) = 88.9028$ and hence (2.17) becomes

$$||\mathbf{Y}^*|| \approx 10^{-t} \, k(\mathbf{A}) \, ||\mathbf{V}^*|| = 1.391 \times 10^{-11}$$

in 13-digit arithmetic and Gaussian elimination. The above approximation suggests that $||\mathbf{V} - \mathbf{V}^*||$ is very small.

Case II: Consider a network with $(K, C) = (2, 3)$, $(R_1, R_2) = (2, 1)$ and $(\lambda_1, \lambda_2) = (1, 5)$.

Applying policy improvement, the value determination takes $N = 44+34$ iterations to calculate the TEDR with the improved policy and therefore using the approximation (2.16) with $\epsilon = 10^{-6}$ and $\xi = 0.70532$ we get

$$\frac{\epsilon}{1 - \xi} = 3.39 \times 10^{-6}.$$

$\rho(\mathbf{T}) = 0.7378 \ (\approx \xi)$.

The condition matrix for the above example calculated using MATLAB Matrix Operations is $k(\mathbf{A}) = 89.8614$ and hence (2.17) becomes

$$||\mathbf{Y}^*|| \approx 10^{-t} \ k(\mathbf{A}) \ ||\mathbf{V}^*|| = 1.8546 \times 10^{-11}$$

in 13-digit arithmetic and Gaussian elimination. The above approximation suggests that $||\mathbf{V} - \mathbf{V}^*||$ is very small.

Case III: Consider a network with $(K, C) = (3, 3)$, $(R_1, R_2) = (2, 1)$ and $(\lambda_1, \lambda_2) = (1.8, 2.3)$.

Applying policy improvement, the value determination takes $N = 61+49+39$ iterations to calculate the TEDR with the improved policy and therefore using the approximation (2.16) with $\epsilon = 10^{-6}$ and $\xi = 0.892588$ we get

$$\frac{\epsilon}{1 - \xi} = 9.31 \times 10^{-6}.$$

$\rho(\mathbf{T}) = 0.8720 \ (\approx \xi)$.

The condition matrix for the above example calculated using MATLAB Matrix Operations is $k(\mathbf{A}) = 102.5391$ and hence (2.17) becomes

$$||\mathbf{Y}^*|| \approx 10^{-t} \ k(\mathbf{A}) \ ||\mathbf{V}^*|| = 4.3621 \times 10^{-11}$$

in 13-digit arithmetic and Gaussian elimination. The above approximation suggests that $||\mathbf{V} - \mathbf{V}^*||$ is very small.

Case IV: Consider a network with $(K, C) = (3, 3)$, $(R_1, R_2) = (2, 1)$ and $(\lambda_1, \lambda_2) = (2, 3)$.

Applying policy improvement, the value determination takes $N = 67+49$ iterations to calculate the TEDR with the improved policy and therefore using the approximation (2.16) with $\epsilon = 10^{-6}$ and $\xi = 0.863274$ we get

$$\frac{\epsilon}{1-\xi} = 7.31 \times 10^{-6}.$$

$\rho(\mathbf{T}) = 0.8591 \ (\approx \xi)$.

The condition matrix for the above example calculated using MATLAB Matrix Operations is $k(\mathbf{A}) = 107.9399$ and hence (2.17) becomes

$$||\mathbf{Y}^*|| \approx 10^{-t} \ k(\mathbf{A}) \ ||\mathbf{V}^*|| = 4.9123 \times 10^{-11}$$

The above approximation suggests that $||\mathbf{V} - \mathbf{V}^*||$ is very small.

Case V: Consider a network with $(K, C) = (4, 3)$, $(R_1, R_2) = (2, 1)$ and $(\lambda_1, \lambda_2) = (2, 2)$.

Applying policy improvement, the value determination takes $N = 92+63$ iterations to calculate the TEDR with the improved policy and therefore using the approximation (2.16) with $\epsilon = 10^{-6}$ and $\xi = 0.82553$ we get

$$\frac{\epsilon}{1-\xi} = 5.56 \times 10^{-6}.$$

2.7.4   Conclusion and Accuracy Check

The Value Detemination step in Policy Improvement can be solved by either using iterative techniques (G-S, SOR) or Gauss Elimination. As mentioned in §2.7.2, the Gauss Elimination solution of $\mathbf{AV} = \mathbf{R}$ as well as the calculation of the condition number are possible for small examples of networks only. This is due to the difficulty posed by the huge state-space in calculating and storing $\mathbf{A}$ which is $|\mathcal{S}|^2$ in size; see also §2.4.2.

The above examples clearly demonstrate that $||\mathbf{Y}^*||$ is much smaller than the SOR truncation error so the Gauss Elimination solution is more accurate. In our examples with $(K = 2, C)$ and $(K = 3, C)$, where $C \leq 4$, the Gauss Elimination solution $\mathbf{V} = \mathbf{A}^{-1}\mathbf{R}$ agrees with the SOR solution to the expected precision (i.e. $\approx 10^{-6}$). That confirms the accuracy of the SOR algorithm.

2.8 The max-flow Bound for Symmetric Star Networks

In this section we demonstrate how to obtain the *max-flow* bound[2] on the performance of our routing scheme under a fixed pattern of offered traffic. The *max-flow* bound is extensively discussed in Gibbens & Kelly (1990) and Gibbens, Kelly & Key (1988); see also §1.5.1. Key (1990) obtains a bound for the optimal return under any dynamic routing scheme by solving a linear programming problem; Lemma 2.1 in Key (1990).

For the networks considered in this thesis the *max-flow* bound is obtained by the solution to the following linear programming problem **LP1**:

$$\max \left[\sum_{i=1}^{K} R_1 x_i + \sum_{l=1}^{K(K-1)/2} R_2 w_l \right],$$

where $x_i \leq \lambda_1$, $i = 1, .., K$; $(K-1)w_l \leq \lambda_2$, $l \in A_i$, $x_i + \sum_{l \in A_i} w_l \leq C$; $x_i, w_l \geq 0$.

Replacing each x_i by y_1 and each term $(K-1)w_l$ by y_2 we get the following linear programming problem **LP2**:

$\max \left[K \, R_1 \, y_1 \, + \, \frac{K}{2} \, R_2 \, y_2 \right]$, where $y_1 \leq \lambda_1$; $y_2 \leq \lambda_2$; $y_1 + y_2 \leq C$; $y_1, y_2 \geq 0$.

Clearly the maximum of **LP1** is at least as large as that of **LP2**. The question to ask is whether the maximum of **LP2** is $\geq$ than that of **LP1**. The answer is yes as demonstrated in the following Lemma.

Lemma 2

LP2 has maximum $\geq$ than that of **LP1**.

Proof: Suppose that (x, w) is an optimal solution with $x_i \neq x_j$ for some $i \neq j$, or $w_l \neq w_k$ for some $l \neq k$. By relabelling the links we can produce another non-symmetric solution (X, W) say. Consider all such solutions (x^p, w^p) produced by

---

[2]See §1.5.1 and (1.18).

permutations p of the indices $\{1, 2, .., K\}$. Consider

$$(\overline{x}, \overline{w}) = \frac{\sum_p (x^p, w^p)}{K!}.$$

$(\overline{x, w})$ is feasible for **LP1** by convexity of the feasible region and has the same objective function value. Further $\overline{x_1} = \overline{x_2} = .. = \overline{x_K}$ and $\overline{w_1} = \overline{w_2} = .. = \overline{w_\gamma}$ and so there exists an optimal solution to **LP1** which is symmetric. Hence the maximum for **LP1** is no larger than the maximum for **LP2**.

LP2 is trivial to solve. The *max-flow* bound is an upper bound for the rate of return for the network. To get a bound for the rate of return per link we just divide by the number of links K.

For the small networks we have considered[3] the *max-flow* bound is not very tight as we will see in §2.9.

2.9 The Rate of Return from the Network

2.9.1 Fixed-Point Approximation for the Blocking Probabilities

In Kelly (1986), if the capacities C_j, $j = 1, 2, .., K$ and the offered traffic ν_r are increased together a limiting regime emerges which has a very simple description. In the limit, there is a parameter $B_j \in [0, 1]$ associated with link j, and the probability that a call requesting route r is lost is given by

$$(2.18) \qquad L_r \approx 1 - \prod_{i \in r}(1 - B_i)^{A_{ir}}, \ r \in \mathcal{R}$$

where A_{ir} is the number of circuits a call on route r uses from link i; $A_{ir} \in Z_+$. In our *Symmetric Star* network, each element of the matrix $\mathbf{A} = (A_{ir}, i =$

---

[3]See §1.4.

$1, 2..., K; r \in \mathcal{R}$) is either 0 or 1, and a route r can be identified with a member of $\mathcal{R} = (\{i\}, \{i, j\}, \ i, j = 1, 2, .., K; \ i \neq j)$ in other words all the one and two link routes.

In our networks (2.18) becomes

$$(2.19) \qquad L_r \approx 1 - (1 - B_i)(1 - B_j), \text{ for } r = (i, j).$$

(2.18) is as if links block independently, link i blocking with probability B_i, where $B = (B_1, B_2, .., B_K)$ is any solution to the set

$$
\begin{aligned}
\sum_{r:i\in r} \nu_r \prod_j (1 - B_j) &= C_i, \text{ if } B_i > 0 \\
(2.20) \qquad \sum_{r:i\in r} \nu_r \prod_j (1 - B_j) &\leq C_i, \text{ if } B_i = 0 \\
B_1, B_2, ..., B_K &\in [0, 1).
\end{aligned}
$$

(2.20) simplifies in our case to

$$(2.21) \qquad \lambda_1(1 - B_i) + \frac{\lambda_2(1 - B_i)}{(K - 1)} \sum_{j \neq i, j \in r} (1 - B_j) - C = 0, \ B_i > 0$$

Kelly states that there always exist a solution to the above relation, and it is unique if the matrix $\mathbf{A} = (A_{jr})$ has rank K.

Under the symmetric assumption for our networks, the quantities B_j are the same for every link i.e. $B_1 = B_2 = ... = B_K = B$, and hence the *fixed-point* approximation relation (2.21) becomes

$$(2.22) \qquad \lambda_1(1 - B) + (K - 1)\frac{\lambda_2(1 - B)^2}{(K - 1)} - C = 0, \quad B \in [0, 1).$$

Note that (2.22) is the same for networks with various K. This is the result of normalisation of 2-link traffic to rate $\lambda_2/(K - 1)$.

The above relation (2.22) is a very useful and convenient one as it allows a straight-

forward approximation for the rate of return in the network. Denote this return as R_{FP}, which is given by

$$(2.23) \qquad R_{FP} = \sum_r R_r \nu_r (1 - L_r),$$

where L_r is the proportion of calls offered to route r which are lost and ν_r is the arrival rate on route r; since a route in our work can be identified to be single link traffic and 2-link traffic we can say $\nu_1 = \lambda_1$ and $\nu_2 = \lambda_2/(K-1)$. The relations below apply

$$(2.24) \qquad L_1 = B_1, \quad L_2 = 1 - (1 - B_1)^2,$$

for the proportion of calls offered to routes 1 and 2 respectively.

2.9.2 Calculating the Blocking Probabilities in Equilibrium

The equilibrium equations for our model can be calculated numerically on a computer and are described by the following *full balance* equations

$$(2.25) \qquad \sigma(z) \sum_{\overline{z} \in \mathcal{S}} p(z, \overline{z}) = \sum_{\overline{z} \in \mathcal{S}} \sigma(\overline{z}) p(\overline{z}, z)$$

where $\sigma(z)$ denotes the equilibrium distribution of the system being in state z, and

$$\sum_z \sigma(z) = \sum_{\overline{z}} p(\overline{z}, z) = 1, \quad z, \overline{z} \in \mathcal{S}.$$

Note that we cannot apply the *detailed balance* due to lack of reversibility.

In this work, a program has been written that solves the equations (2.25) with a procedure similar to that used in the value determination step for solving (2.9). This procedure calculates:

(a) The blocking probability of a single link traffic on link, say i, when there are C calls present on i (i.e. $F_i = 0$), as

$$B_1 = \sum_{S_1} \sigma(z) \qquad \text{where} \qquad S_1 = \{z \in \mathcal{S} : F_i = 0\},$$

where F_i is the number of free circuits on link i. The above expression gives the true blocking B_1.

(b) The actual blocking probability of a 2-link call on route l under the optimal policy, as

$$B_2 = \sum_{S_2} \sigma(z) \qquad \text{where} \qquad S_2 = \{z \in \mathcal{S} : I(e_l, z) = 0\},$$

where $I(e_l, z) = 0$ denotes rejection by the optimal policy.

Because Poisson arrivals see the equilibrium distribution[4], the rate of blocked single link traffic on link i, is given by

$$\lambda_1 \sum_{S_1} \pi(z) \qquad \text{where} \qquad S_1 = \{z \in \mathcal{S} : F_i = 0\},$$

The actual rate of return per link per unit time is also calculated numerically as follows:

$$R_E = R_1 \, \lambda_1 \, (1 - B_1) + \frac{R_2}{2} \, \frac{\lambda_2}{K - 1} \, \sum_1^{K-1} (1 - B_2)$$

which becomes

$$R_E = R_1 \, \lambda_1 \, (1 - B_1) + \frac{R_2}{2} \, \lambda_2 \, (1 - B_2)$$

Note that in the exact case it is necessary to calculate B_2 the same way B_1 is

---

[4]See Tijms (1988).

74

calculated because $(1 - B_2) \neq (1 - B_1)^2$; this is only asumptotically correct.

The following table shows some results in order to compare the Equilibrium Return (R_E), the Real Return (R_R), the *max-flow* return R_{mf} and the Fixed-Point Approximation Return (R_{FP}) as well.

Size of networks	ν_1	ν_2	R_R	R_E	R_{FP}	R_{mf}
(K, C)						
	0.5	4.9	1.53	1.28	2.05	2.25
	0.7	4.5	1.72	1.56	2.28	2.55
	1	3.9	2.02	1.92	2.63	3.00
3,3	1.3	3.3	2.34	2.32	3.01	3.45
	1.5	2.9	2.54	2.55	3.28	3.75
	1.8	2.3	2.84	2.87	3.70	4.20
	0.3	5.6	1.38	1.07	1.81	1.95
	0.6	5	1.66	1.42	2.14	2.40
	1	4.2	2.03	1.92	2.60	3.00
3,3	1.4	3.4	2.45	2.44	3.07	3.60
	1.7	2.8	2.74	2.76	3.47	4.05
	2	2.2	3.03	3.05	3.89	4.50
	0.5	4.9	1.33	0.91	2.05	2.25
	0.7	4.5	1.51	1.03	2.28	2.55
	1	3.9	1.80	1.53	2.63	3.00
4,3	1.3	3.3	2.10	2.09	3.01	3.45
	1.5	2.9	2.19	2.33	3.28	3.75
	1.8	2.3	2.56	2.69	3.70	4.20
	0.3	5.6	1.18	0.85	1.81	1.95
	0.6	5	1.42	0.92	2.14	2.40
	1	4.2	1.80	1.55	2.60	3.00
4,3	1.4	3.4	2.20	2.20	3.07	3.60
	1.7	2.8	2.47	2.54	3.47	4.05
	2	2.2	2.73	2.92	3.89	4.50

Table 212: Real, equilibrium, *max-flow* and *fixed-point* rate of return per link.

The real return R_R is not of any interest to us as we never consider what happens as $\alpha \rightarrow 1$. It only provides a good check on the accuracy of our programs for calculating the stationary distribution from (2.25). R_R shown in Table 2.2 are calculated using the following formula that connects the average and discounted reward (with $\alpha = 0.8$) $R_R \approx \frac{V(0)(1-\alpha)}{K}$. The results suggest that the *fixed-point* approximation is not very good for the small networks considered.

Chapter 3

Optimal Policies for Symmetric Networks

3.1 Complexity

For our networks presented in §1.1 and §2.1, we consider policies that always accept 1-link calls when there is room to fit them in. It is the 2-link calls we seek to restrict (in order to maximise the TEDR) and hence a policy is a set of $|\mathcal{S}| \times \frac{1}{2}K(K-1)$-tuples of Boolean variables. 336 triples when $(K = 3, C = 3)$. As the networks increase in size the optimal policy increases rapidly. In the following examples, we describe the exact optimal policy for networks with $(K = 3, C = 3)$ only to show the complexity and difficulty that both arise in trying to evaluate network's behaviour and performance by looking at the calculated optimal policy. In the following description we do not include: (a) the cases in which the optimal policy accepts the 2-link calls when there is room to fit them in; (b) the cases in which the network is full.

Let $F = (F_1, F_2, F_3)$ denote the number of free circuits on links X, Y and Z respectively. A network state is represented by a 6–tuple (X,XY,XZ,Y,YZ,Z). For example (010011) describes the state of the network with: one 2-link call on pair XY, one 2-link call on pair YZ, and a single-link call on Z with corresponding free circuits

$F = (2,1,1)$. Many states have the same number of free circuits; for example states (110010), (020001) and (200201) all have $F = (1,1,2)$.

Let 0 denote the state with all circuits on all links free i.e. $F = (C, C, C)$ for the 3-link networks. Let $V(0)$ denote the total expected discounted reward (TEDR) for the state 0. The number of possible states in the following examples is 336; see §1.4.

The quantities $\lambda_1, \lambda_2, R_1, R_2,$ and α are all defined in §2.1, §2.2 and §2.3. Note that the α values given are for the continuous time process and not the modified values used in the uniformised *optimality equation*; see §2.3.2.

In this description we follow the L.Carroll (1865) method in *Alice in Wonderland* in which we first present some evidence and then the sentence/conclusion.

In what follows we will say that a state $z \geq \bar{z}$ whenever $F_z \leq F_{\bar{z}}$. For example a state z with $F_z = (1,1,1)$ is considered $z > \bar{z}$ where $F_{\bar{z}} = (1,1,3)$ i.e. a very large state in terms of link occupancy has a very small number of free circuits.

The following examples demonstrate the complexity in describing the optimal case for networks with $(K, C) = (3,3), R_1 = 2, R_2 = 1$ and $\alpha = 0.8$ in which there are 336 possible states.

Example 3.1

$\underline{\lambda_1 = 2, \lambda_2 = 1}$

The optimal policy rejects some 2-link calls when there is room to fit them in. Specifically in this network the optimal policy rejects:

(a) All 2-link calls in 60 states with $F=$ (1,1,1), (2,2,2), (3,1,1), (2,2,1), (2,1,1), (2,1,2), (1,3,1), (1,2,1), (1,1,3), (1,1,2), (1,2,2).

(b) XY and XZ calls in states with (1,3,2), (1,2,3) and (1,3,3) free circuits.

(c) XY and YZ calls in states with (3,1,2), (3,1,3) and (2,1,3) free circuits.

(d) XZ and YZ calls in states with (3,3,1), (3,2,1) and (2,3,1) free circuits.

(e) XY calls in states with (3,1,0), (1,3,0), (2,2,0), (2,1,0), (1,2,0), (1,1,0) and (2,2,3) free circuits.

(f) XZ calls in states with (3,0,1), (1,0,3), (2,0,1), (1,0,1), (2,0,2), (2,3,2) and (1,0,2).

(g) YZ calls in states with (0,3,1), (0,1,3), (0,1,2), (0,2,1), (0,2,2), (0,1,1) and (3,2,2).

The value for the TEDR for state 0 is $V(0)=$ 60.0658.

<u>Observation 1</u>

Different states with the same number of free circuits F under the optimal policy have similar TEDR values. For example, the states (000020), (000111) and (000202) with $F = (3,1,1)$ have TEDR values at 58.1079, 58.1069 and 58.1057.

Different states with 'symmetrical' F appear to have similar Total Expected Discounted Return (TEDR) values. For example, states with $F=$ (1,2,1), (2,1,1) and (1,1,2) have TEDR values at 57.7419 (002100), 57.7429 (010011) and 57.7411 (010102). For states with $F=$ (2,2,1), (1,2,2) and (2,1,2) some TEDR values are: 58.3171 (001010), 58.3141 (001101), 58.3163 (001200) and 58.3171 (010010). For states (000001), (100000) and (000100) with $F=$ (3,3,2), (2,3,3) and (3,2,3) the TEDR is the same. This is not surprising since the 'symmetrical' denotes permutations with underlying network symmetry.

<u>Observation 2</u>

As it is clear from the above example and due to the symmetry assumption of our networks it is easy to deduce the optimal policy for all 2-link routes by just looking at its description for one of the possible 2-link routes. For example knowing that a XY call is rejected in (1,1,2) and (1,1,3) can help us deduce that rejections in XZ will be in (1,2,1) and (1,3,1) i.e. by permuting the F_2 with the F_3.

In the next examples with $(K, C) = (3, 3)$, the rate of the total traffic offered per link over the network $L = \lambda_1 + \lambda_2/2$ remains the same (=2.95) while the rate of 2-link traffic decreases.

Example 3.2

$\lambda_1 = 0.5, \lambda_2 = 4.9$

The optimal policy rejects some 2-link. Specifically in this network the optimal policy rejects:

(a) 2-link calls on XY in 12 states with $F=$ (1,1,2), (1,1,3), (1,1,1); XZ and YZ calls are accepted in these cases except (1,1,1).

(b) 2-link calls on XZ in 12 states with $F=$ (1,3,1), (1,2,1), (1,1,1); XY and YZ calls are accepted in these cases except (1,1,1).

(c) 2-link calls on YZ in 12 states with $F=$ (3,1,1), (2,1,1), (1,1,1); XY and XZ calls are accepted in these cases except (1,1,1).

Example 3.3

$\lambda_1 = 1.5, \lambda_2 = 2.9$

The optimal policy rejects a 2-link call on pair XY in states with:

(a) $F=$ (2,2,3) but not in states with $F=$ (2,2,0), (2,2,1), (2,2,2).

(b) $F=$ (1,1,0), (1,1,1), (1,1,2) and (1,1,3).

(c) $F=$ (2,1,0), (2,1,1), (2,1,2) and (2,1,3).

(d) $F=$ (1,2,0), (1,2,1), (1,2,2) and (1,2,3).

(e) $F=$ (1,3,0), (1,3,1), (1,3,2) and (1,3,3).

(f) $F=$ (3,1,0), (3,1,1), (3,1,2) and (3,1,3).

Example 3.4

$\lambda_1 = 1.8, \lambda_2 = 2.3$

The optimal policy rejects:

(a) All 2-link calls in 45 states with $F=$ (1,3,1), (3,1,1), (1,1,3), (2,2,2), (1,1,1), (2,1,1), (1,1,2) and (1,2,1).

(b) XY and XZ calls in states with $F=$ (1,2,2), (1,3,2), (1,2,3) and (1,3,3).

(c) XZ and YZ calls in states with $F=$ (2,2,1), (2,3,1), (3,2,1) and (3,3,1).

(d) XY and YZ calls in states with $F=$ (2,1,2), (3,1,2), (2,1,3) and (3,1,3).

(e) XY calls in states with $F=$ (2,2,3), (3,1,0), (1,3,0), (2,1,0), (1,2,0) and (1,1,0).

79

(f) YZ calls in states with $F = $ (3,2,2), (0,3,1), (0,1,3), (0,2,1), (0,1,2) and (0,1,1).

(g) XZ calls in states with $F = $ (2,3,2), (1,0,3), (3,0,1), (2,0,1), (1,0,2) and (1,0,1).

Note: As demonstrated in the above examples, in various states all, two or one of the 2-link call types are rejected. We focus our analysis on rejection of calls on a particular 2-link route, XY. If we consider the previous case for example, by rejecting a call on XY we include states from (a), (b), (d) *and* (e).

3.2   Optimal Policies: Properties when $R_2 < 2R_1$

Before we proceed with giving evidence that suggest certain properties of the optimal policies, we shall briefly refer to the important results of Key (1990)[1] on the properties of the optimal policy for networks where links are held independently. In our work we assume that 2-link calls hold both links for the same amount of time (dependency[2]) and therefore it is very interesting to investigate whether 'independent occupancy' results as considered by Key (1990) apply to our networks.

We mentioned earlier on that we assume $R_2 < 2R_1$. We have assumed this motivated from Theorems 4.1 and 4.2 in Key (1990) which are presented in §1.8. Key's results only imply acceptance of all 1-link calls for $R_2 < R_1$ which is often the case in most of the examples in this Chapter. Assumptions A1 and A2 in Key (1990) hold for our networks. Theorem 4.3 in Key (1990) does not apply to our networks: think for example of a *Star* network with $K \geq 3$.

The properties we seek to find evidenece for or against are the following:

Property A: Dependency on the State-Space

Property B: Property P2 in Key (1990) suggests that if we reject type j calls in $z + e_i$ then we reject them in z for call types i and j which are disjoint. Property P2 means that for calls which are disjoint, and thus could be widely separated in a network, in general, the more type i calls in progress, the less likely we are to reject

---

[1]See §1.8

[2]See §1.1

type j calls, and vice-versa; disjoint calls are for example XY and Z calls.

Property C: Monotonicity. If an arrival for a 2-link call on route k is rejected in state $\bar{z}$, then it will also be rejected in states z, where $z \geq \bar{z}$.

Property D: Property P1 in Key (1990) suggests that if we reject a type i call in state z, then we reject it in state $z + e_k$ for calls i and k which are distinct and not disjoint; not disjoint calls are for example XY and YZ calls.

Property E: Weak Monotonicity Assumption C1 in Key (1990) in which type k calls are monotonic with respect to themselves, that is we assume that for all call types k, if we reject a type k call in state z, then we reject a type call k in state $z + e_k$.

By looking at examples of networks for the cases with

(a) $K = 2$, $C \leq 5$,

(b) $K = 3$, $C \leq 5$,

(c) $K = 4$, $C \leq 5$,

(d) $K = 5$, $C = 3$.

and for 24 various offered traffic per case, we have evidence that suggest the properties A, B and E apply in our *Symmetric* networks:

Property A: Dependency on the State-Space The optimal policy depends upon the full state space $\mathcal{S}$ and not just the free circuit configurations. This is demonstrated in the following examples.

| Example 3.5 |

With parameters $(K, C) = (3, 3)$, $\lambda_1 = 0.5, \lambda_2 = 4.9, R_1 = 2, R_2 = 1$.

The optimal policy rejects a 2-link call on pair (Y,Z) in the following states with $F = (1,1,1)$:

$$(200020) \text{ and } (200111)$$

With the same free circuits, $F = (1,1,1)$, the optimal policy accepts a 2-link call on

pair (Y,Z) in states:

$$(200202), (110011) \text{ and } (020002)$$

[The values of these states are 20.8016, 20.8411 and 20.8452 respectively].

These differences are not due to truncation error effects. For instance at state (200020) by comparing the 'accept' and 'reject' decisions we see that[3]

$$R_2 + \overline{\alpha} V_\alpha(200030) = 20.5672$$

while

$$\overline{\alpha} V_\alpha(200020) = 20.5862$$

and the difference is four orders of magnitude bigger than the truncation error (which is calculated to be 3.05×10^{-6}; see §2.4.7.

Property B: Non-Locality and Disjointness

The following examples give evidence to support Property P2 for disjoint types of calls. In what follows we examine the disjoint types of calls XY and Z.

In examples 3.5.1 and 3.5.2 $(K,C) = (3,3), R_2 = 1, R_2 = 1$ and $\alpha = 0.8$ and the number of states is 336.

$\boxed{\textbf{Example 3.5.1}}$

$\underline{\lambda_1 = 0.5, \lambda_2 = 4.9}$

The optimal policy rejects a 2-link call on pair XY in states with $F = (1,1,1), (1,1,2), (1,1,3)$ but not in states with $F = (1,1,0)$.

For example consider the following states in which, if XY is rejected in $z + e_3$, then it will be rejected in z.

(1) Rejection of XY in (110102) coincides with rejection in both (110101) and (110100); with free circuits (1,1,1), (1,1,2) and (1,1,3) respectively.

---

[3]See §2.4.1.

(2) Rejection of XY in (020002) coincides with rejection in both (020001) and (020000); with free circuits (1,1,1), (1,1,2) and (1,1,3) respectively.

(3) Rejection of XY in (200201) coincides with rejection in (200200); with free circuits (1,1,2) and (1,1,3) respectively.

Example 3.5.2

$\lambda_1 = 1, \lambda_2 = 4$

The optimal policy rejects a 2-link call on pair XY in states with:

(a) F= (2,1,1), (2,1,2), (2,1,3) but not in states with F= (2,1,0).

(b) F= (1,2,1), (1,2,2), (1,2,3) but not in states with F= (1,2,0).

(c) F= (1,1,0), (1,1,1), (1,1,2) and (1,1,3).

In examples 3.5.3 and 3.5.4 $(K, C) = (3, 4)$ and the number of states is 1023.

Example 3.5.3

$\lambda_1 = 2, \lambda_2 = 2$

The optimal policy rejects a 2-link call on pair XY in states with F= (2,2,3), (2,2,4) but not in states with F= (2,2,0), (2,2,1), (2,2,2).

For example consider the following states in which, if XY is rejected in $z + e_3$, then it will be rejected in z.

(a) Rejection of XY in (200201) coincides with rejection in (200200); with free circuits (2,2,3) and (2,2,4) respectively.

(b) Rejection of XY in (110101) coincides with rejection in (110100); with free circuits (2,2,3) and (2,2,4) respectively.

The optimal policy also rejects a XY call in all states with free circuits (1,3,n), (3,1,n), (1,2,n), (2,1,n), (1,1,n) (1,4,n) (4,1,n) where n=0,1,2,3,4.

Example 3.5.4

$\lambda_1 = 0.5, \lambda_2 = 4.9$

The optimal policy rejects a 2-link call on pair XY in states with $F = (1,1,2)$, $(1,1,3)$, $(1,1,4)$ but not in states with $F = (1,1,0)$, $(1,1,1)$,

For example consider the following states in which, if XY is rejected in $z + e_3$, then it will be rejected in z.

(1) Rejection of XY in (030002) coincides with rejection in both (030001) and (030000); with free circuits $(1,1,2)$ and $(1,1,3)$ and $(1,1,4)$ respectively.

(2) Rejection of XY in (111201) coincides with rejection in (111200); with free circuits $(1,1,2)$ and $(1,1,3)$ respectively.

Discussion on Property B

The above examples demonstrate clearly that the optimal policy is *non-local* in that the status of link Z effects the acceptance of 2-link calls on XY. It seems that the optimal policy acts like a very intelligent mechanism which in Example 3.5.3 for example rejects a XY call in states (010000) and (100100) and not in other states with $F = (2,2,0)$ waiting probably for more profitable calls to arrive on routes XZ and YZ.

The above examples also suggest that Property P2 in Key (1990) seems to apply in our networks. Property P2 suggests that for calls that are 'disjoint', and thus could be separated in a network, in general, the more i calls in progress, the less likely we are to reject type j calls, and vice-versa. This is true in all of our examples.

Property C: Monotonicity If an arrival for a 2-link call on route k is rejected in state $\bar{z}$, then it will also be rejected in states z, where $z \geq \bar{z}$.

Our results tell us that Property C does not hold for our networks in general; see Examples 3.2, 3.5.2 and 3.5.3 and 3.5.4.

In Example 3.2: If we describe states with free circuits $(1,1,0)$, $(1,1,1)$, $(1,1,2)$, and $(1,1,3)$ as z_0, z_1, z_2 and z_3 respectively, then it is obvious that $z_0 > z_1 > z_2 > z_3$ in terms of occupancy. Rejection in z_3 coincides with rejection in z_2 and z_1 but not in

z_0. Rejection in z_2 coincides with rejection in z_1 but not in z_0. Rejection in z_1 does not coincide with rejection in z_0.

In Example 3.5.3: If we describe states with free circuits (2,2,0), (2,2,1), (2,2,2), (2,2,3), and (2,2,4) as z_0, z_1, z_2, z_3 and z_4 respectively, then it is obvious that $z_0 > z_1 > z_2 > z_3 > z_4$ in terms of occupancy. Rejection in z_4 coincides with rejection in z_3 but not in z_0, z_1 and z_2.

Property D: Not Disjointness

For not disjoint types of calls i and k if we reject type i in z, then we will reject it in $z + k$. Not disjoint types of calls are for example XY and YZ calls.

In 13 examples Property D did hold.

Example 3.5.5

$\lambda_1 = 1.8, \lambda_2 = 2.3$

A XY call is rejected in state (010000) with (2,2,3) free circuits as well as in (010010) with (2,1,2) free circuits; see Example 3.3.

Property E: Weak Monotonicity If we reject a type k 2-link call in state z, then we reject a type call k in state $z + e_k$. In 13 examples of various size networks Property E was true.

Example 3.5.6

$\lambda_1 = 2, \lambda_2 = 2.2$

A XY call is rejected in state (010000) with (2,2,3) free circuits and also in (020000) with (1,1,3) free circuits.

Example 3.5.7

$\lambda_1 = 1.6, \lambda_2 = 4.8$

A XY call is rejected in state (000110) with (3,1,2) free circuits and also in (010110) with (2,0,2) free circuits.

3.3 The Optimal Policy as $\alpha \to 1$.

In this section we will briefly present observations on the optimal policy for networks with $K, C, \lambda_1, \lambda_2, R_1, R_2$ fixed in which $\alpha \to 1$. The results suggest that as $\alpha \to 1$ the optimal policy rejects more 2-link calls. That means that the optimal policy takes into account the fact that the reward earned in the present becomes less important than potential earnings in the future. Key (1990) demonstrates this fact for the case of single link offered a number of traffic. It seems to be true in the star-shaped networks we study which confirms their simplicity in some respect.

In what follows we will describe the optimal policy on the rejection of 2-link calls on route XY as well as rejection of all 2-link calls (XY, XZ and YZ); results for the other 2-link routes can be deduced by applying the 'symmetry' assumption; see Observation 2 in §3.1.

| Example 3.6 |

$\underline{\lambda_1 = 0.5, \lambda_2 = 4.9}$

I: $\alpha = 0.8$. The optimal policy rejects XY calls in 12 states with $F = (1,1,1)$, $(1,1,2)$ and $(1,1,3)$ free circuits. $V(0) = 22.9963$.

II: $\alpha = 0.9$. (a) The optimal policy rejects XY calls in 9 states with $F = (1,1,2)$ and $(1,1,3)$ free circuits. (b) The optimal policy rejects XY, XZ and YZ calls in 10 states with $F = (1,1,1)$. $V(0) = 45.9458$.

III: $\alpha = 0.95$. (a) The optimal policy rejects XY calls in 23 states with $F = (1,1,0)$, $(1,1,2)$, $(1,1,3)$, $(2,1,3)$ and $(1,2,3)$ free circuits. (b) The optimal policy rejects XY, XZ and YZ calls in 11 states with $F = (1,1,1)$. $V(0) = 91.7682$.

IV: $\alpha = 0.99$. (a) The optimal policy rejects XY calls in 24 states with $F = (1,1,0)$, $(1,1,2)$, $(1,1,3)$, $(2,1,3)$ and $(1,2,3)$ free circuits. (b) The optimal policy rejects XY, XZ and YZ calls in 11 states with $F = (1,1,1)$. $V(0) = 458.4749$.

V: $\alpha = 0.999$. (a) The optimal policy rejects XY calls in 27 states with $F = (1,1,0)$, $(1,1,2)$, $(1,1,3)$, $(2,1,3)$ and $(1,2,3)$ free circuits. (b) The optimal policy rejects XY, XZ and YZ calls in 11 states with $F = (1,1,1)$. $V(0) = 4583.5120$.

VI: $\alpha = 0.9992$. (a) The optimal policy rejects XY calls in 27 states with $F = (1,1,0)$, $(1,1,2)$, $(1,1,3)$, $(2,1,3)$ and $(1,2,3)$ free circuits. (b) The optimal policy rejects XY, XZ and YZ calls in 11 states with $F = (1,1,1)$. $V(0) = 5729.3537$.

To find approximately the Expected Average Reward (AER) for the above cases we just multiply the above $V(0)$ quantities with their respective $(1-\alpha)$. The results for the above cases are then as follows: I) 0.2856, (II) 0.2703, (III) 0.2633, (IV) 0.2580 and (V) 0.2569 and (VI) 0.25684; see (1.5) in §1.3.3 for the relation between the Average and the Discounted Expected Reward.

As we see the optimal policy remains the same for $\alpha > 0.999$ and so this policy is the Expected Average Reward (EAR) optimal; see Ross (1983).

3.4 Properties of the Optimal Policy when $R_2 \approx 2 \times R_1$

In this section we will only describe the behaviour of the optimal policy under the assumption $R_2 \approx 2 \times R_1$. In all the examples $(K, C) = (3,3)$ and $R_1 = 1$, and $\alpha = 0.8$.

Example 3.7.

$\underline{\lambda_1 = 0.5, \lambda_2 = 4.9}$

(a) $R_2 = 1$: The optimal policy rejects a 2-link call on pair XY in states (020000), (020001) and (200200) with $F = (1,1,3)$ but not in states with $F = (1,1,2)$, $(1,1,0)$, $(1,1,1)$.

(b) $R_2 = 1.5$: The policy rejects a 2-link call on pair XY in state (020000) with $F = (1,1,3)$ but not in states with $F = (1,1,2)$, $(1,1,0)$, $(1,1,1)$.

(c) $R_2 = 1.8$: The policy never rejects a 2-link call on pair XY when there is room to fit it in.

(d) $R_2 = 1.9$: The policy never rejects a 2-link call on pair XY when there is room to fit it in.

(e) $R_2 = 2$: The policy never rejects a 2-link call on pair XY when there is room to fit it in.

(f) $R_2 = 2.3$: The policy never rejects a 2-link call on pair XY when there is room to fit it in.

In the following example we keep the rewards fixed and see what happens as the offered traffic varies.

Example 3.8

$R_1 = 1, R_2 = 1.9$

The policy

(a) never rejects a 2-link call on pair XY when there is room to fit it in for the following examples with arrival rates $(\lambda_1, \lambda_2)= (1.4, 3.4)$, $(2, 4)$ and $(2, 2)$.

(b) rejects one 2-link call on pair XY in states (020000), (020001) and (200200) with $F = (1,1,3)$ but not in states with $F = (1,1,2)$, $(1,1,0)$, $(1,1,1)$ with arrival rates $(\lambda_1, \lambda_2)= (0.7, 6.6)$.

In the following example we keep the traffic fixed and try different rewards for the single link traffic.

Example 3.9

$\lambda_1 = 0.7, \lambda_2 = 6.6, R_1 = 1, \alpha = 0.8$

(a) $R_2 = 1$: The optimal policy rejects a 2-link call on pair XY in 10 states with $F = (1,1,3)$, $(1,1,2)$ but not in states with $F = (1,1,1)$, $(1,1,0)$.

(b) $R_2 = 1.5$: The optimal policy rejects a 2-link call on pair XY in 7 states with $F = (1,1,3)$, $(1,1,2)$ but not in states with $F = (1,1,1)$, $(1,1,0)$.

(c) $R_2 = 1.8$: The optimal policy rejects a 2-link call on pair XY in 4 states with $F = (1,1,3)$, $(1,1,2)$ but not in states with $F = (1,1,1)$, $(1,1,0)$.

(d) $R_2 = 1.9$: The policy rejects a 2-link call on pair XY in 4 states with $F = (1,1,3)$, $(1,1,2)$ but not in states with $F = (1,1,1)$, $(11,0)$.

(e) $R_2 = 2$: The policy rejects a 2-link call on pair XY in 3 states with $F = (1,1,3)$ but not in states with $F = (1,1,2)$, $(1,1,1)$, $(1,1,0)$.

(f) $R_2 = 2.3$: The policy rejects a 2-link call on pair XY in 3 states with $F = (1,1,3)$ but not in states with $F = (1,1,2)$, $(1,1,1)$, $(1,1,0)$.

3.4.1 Future Work

It remains to be investigated in the future whether the policy that is found in cases (d), (e) and (f) is indeed the optimal one, and also to see if it is at all optimal to restrict the single link traffic in order to maximise TEDR for the latter cases.

At the time this thesis was completed there was a program in progress to investigate possible rejections of 1-link calls in cases where $R_2 \approx 2 \times R_1$. That would enable us to know if the policy derived in the above examples is indeed the optimal one. Unfortunatelly time limitations did not allow us to continue. Part of this program can be found in Appendix B.5.

Chapter 4

Admission Price Policies

4.1 Definition and Background

As mentioned in Chapter 1 and demonstrated in §3, the optimal policies are not easy to interpret; the results of our programs give us the optimal policy in an explicit way which looks extremely complex.

An idea considered in our work has been to approximate the optimal policy by considering policies that depend only upon the occupancy status of the links. Our aim is to calculate important features of the network behaviour under these approximating policies. A class of such policies investigated in this chapter are the *Admission Price* policies; we shall call them Ω policies. These policies are not optimal but they are believed to be [Hunt, Laws, MacPhee and Ziedins] asymptotically optimal for large networks; where the number of links grows to infinity. In this work we investigated if such policies are any good for small networks. The results suggest that they are.

The Ω policies are easy to describe: Let $\mathbf{W} = (W_0, W_1, .., W_C)$ be a set of constants where $0 \le W_i \le R_1$; Let a 2-link route call arrive requesting route (i, j) and let F_i and F_j be the number of free circuits on links i and j respectively. The 2-link call is accepted when

$$W_{F_i} + W_{F_j} \leq R_2.$$

Comments:

(a) The Ω policies are well defined for any collection of W_i.

(b) It is of interest to find good W_i but numerical optimisation methods struggle because of the nature of V; see §4.1.2.

(c) Because we know (numerically) the optimal value function, we can make excellent guesses for the W_i.

4.1.1 Calculating the W_i.

Numbers W_{F_i} can be calculated the following way:

For the optimal V under the optimal (but difficult to describe) policy, and for all the states $z = (x, w)$ that could possibly accept an increase in their number of 1-link calls, we calculate the differences

$$V(z + e_i) - V(z).$$

Then we look at those differences for a collection of states $z_a, z_b, ..z_m$ with the same number of free links F_i. For 'good' W_i the following relation must be satisfied

$$V(z + e_i) - V(z) \approx W_{F_i}.$$

The choice for such W_{F_i} is taken from within the range of the above differences; usually their mean value.

The restriction that $W_i \leq R_1$ is necessary since we assumed in this work that $R_2 \leq 2R_1$ and because of the interpretation that differences $V(z + e_i) - V(z)$ have as the extra amount to be paid for starting in state $z + e_i$ rather than in z.

91

The motivation in considering policies which only depend on the state of the links of a network was the work of Ott & Krishnan (1985, 1986) and Key (1990), who both investigate value functions which can describe the optimal policy and a routing scheme. These authors argue from the single link in isolation towards the more complex network.

Ott & Krishnan based their *Separable Routing* scheme on the calculation of (1.20)[1] by considering the differences $V(k+1) - V(k)$ as a representation of the expected number of additional calls blocked (in the long run) when we start with $k+1$ busy circuits rather than k; see also Howard (1960) and Tijms (1988).

Key (1990) uses Howard's results on relative values and considers the difference $V_j - V_i$ as the difference in total expected reward over an indefinitely long period caused by starting in state j rather than i under a certain policy employed. He also states that the relative values can be used in calculating the reward functions until accuracy is achieved.

In this work we check how good these Ω policies are numerically on a computer; see Appendix B.4 for details of the programs used. As the results in §4.2 suggest, these policies are indeed very close to the optimal.

4.1.2 Restrictions and Optimisation

The problem is how to choose good values for the W_i. We only think that W's found from looking at the true optimal V will be good and indeed they are, but there are lots of examples where we don't know V so how can we choose the W_i in such cases? The answer is not easy. An idea would be to use optimisation methods to try to get better W_i's so that we can check our guesses about the W_i.

In optimisation our first task is to find an expression for

$$V(0; \mathbf{W}) = h(\Omega),$$

---

[1]See §1.7.1

92

where Ω is the *Admission Price* policies described in section §4.1. The problem could then be expressed as follows:

$$\max_W h(\Omega)$$
$$\text{where}$$
$$0 \leq W_i \leq R_1.$$

The function and the constraints are continuous and the optimisation problem is a non-linear constrained problem. We know by examining the policy that $h(\Omega)$ will be locally constant in $W-$space with discontinuities at various places (many W_i's give the same policy and the policy determines V) so all methods that assume that V is differentiable will be unreliable.

We have tried optimisation methods for the W_i's on a computer:

(a) Firstly, by considering the problem as a Sequential Quadratic Programming (SQP) problem in which a quadratic subproblem is solved at each iteration. In 15 examples of different networks, this did not perform any optimisation for the reasons stated above.

(b) Secondly, by considering the problem with no constraints using the simplex algorithm of Nelder and Mead (1965) in which the simplex algorithm automatically rescales itself according to the local geometry of the function $h(\Omega)$. The method was easily and quickly programmed and was very modest in storage demand but did not perform any optimisation when we started with the best Ω policy; when started with a choice not as good as the best Ω it performed a slight improvement but the W_i's rarely reached the optimal level.

For small networks we should not worry about this for the reason that the W from the very simple and 'easy' - relatively speaking - networks are very good if applied for larger networks. For example, in cases like $(K, C) = (4, 3)$, $(4, 4)$, $(4, 5)$, the results are excellent if we use W from the $(3, 3)$, $(3, 4)$, $(3, 5)$ networks respectively. As we have lots of initial W's from the 2, 3 and 4 links networks we have in our hands an estimation of the W_i which is - at least as far as our examples suggest - excellent.

4.2 Comparing the Optimal Policy with the Admission Price Policy

In this section we present tables with results and figures that show:

(1) How good the approximation $V_W(0)$ is for $V(0)$;

(2) The behaviour of the W_i themselves. In the examples presented in the tables the size of the numerical error ranges:

(a) from 1.15×10^{-6} to 4.67×10^{-6} for networks $(K, C) = (2, 3)$.

(b) from 2.83×10^{-6} to 9.82×10^{-6} for networks $(K, C) = (3, 3)$.

(c) from 5.22×10^{-6} to 8.12×10^{-6} for networks $(K, C) = (4, 3)$.

We present the results in groups of three categories:

(a) The size of the network i.e. (K, C);

(b) For a specific size of a network we keep L fixed for convenience; this keeps the rate of events the same for a collection of different arrival rates. We have focused on examples for networks where L is 'critical' for the link i.e. nearly full, full, overloaded etc; see in §2.3.2.

(c) For a specific L we consider various arrival rates. In the tables $\nu_1 = \lambda_1$ and $\nu_2 = \lambda_2$. In the figures $n1 = \lambda_1$ and $n2 = \lambda_2$. All the other quantities have been defined in §2. Unless otherwise specified $R_1 = 2$ and $R_2 = 1$ and $\mu_1 = 1, \mu_2 = 1$.

$V(0)$ is calculated using the *uniformised optimality equation* and by employing the policy improvement algorithm; see §2.4.

$V_W(0)$ is calculated by value determination when Ω policy is employed for the rejection of 2-link calls.

For an explanation of why α is not exactly 0.8 see the remark on page 50.

$$\boxed{K = 2, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V(0)$	$V_W(0)$
0.5	4.9	2.95	15.6528	14.7918
0.7	4.5	2.95	17.5433	17.0851
1	3.9	2.95	20.4357	20.3214
1.3	3.3	2.95	23.5023	23.5020
1.5	2.9	2.95	25.5357	25.5353
1.8	2.3	2.95	28.4599	28.4599
0.5	5	3	15.6812	14.8235
0.7	4.6	3	17.5733	17.1149
1	4	3	20.4578	20.3481
1.3	3.4	3	23.5230	23.5228
1.5	3	3	25.5536	25.5535
2	2	3	30.3991	30.3991
0.3	5.6	3.1	14.3999	12.5278
0.6	5	3.1	16.6175	16.6174
1	4.2	3.1	20.4994	20.3979
1.4	3.4	3.1	24.5818	24.5812
1.7	2.8	3.1	27.5340	27.5339
2	2.2	3.1	30.4095	30.4094
0.5	6	3.5	15.9102	15.7782
0.7	5.6	3.5	17.8234	17.3670
1	5	3.5	20.6379	20.5681
1.3	4.4	3.5	23.6930	23.2417
1.6	3.8	3.5	26.6509	26.6493
2	3	3.5	30.4430	30.4431
0.5	7	4	16.0666	15.7300
0.7	6.6	4	18.0078	17.5801
1	6	4	20.7770	20.7313
1.3	5.4	4	23.8157	23.8151
1.6	4.8	4	26.7247	26.7239
2	4	4	30.4714	30.4714

Table 4.1: Networks with $(K, C) = (2, 3)$.

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V(0)$	$V_W(0)$
0.5	4.9	2.95	22.9963	22.8486
0.7	4.5	2.95	25.8420	25.8320
1	3.9	2.95	30.3238	30.2914
1.3	3.3	2.95	35.0391	35.0389
1.5	2.9	2.95	38.1230	38.1224
1.8	2.3	2.95	42.5410	42.5230
0.5	5	3	23.0410	22.8902
0.7	4.6	3	25.8825	25.8703
1	4	3	30.3579	30.3276
1.3	3.4	3	35.0693	35.0693
1.5	3	3	38.1477	38.1465
2	2	3	45.4389	45.4389
0.3	5.6	3.1	20.6261	20.5218
0.6	5	3.1	24.4902	24.4882
1	4.2	3.1	30.4228	30.3966
1.4	3.4	3.1	36.6757	36.6753
1.7	2.8	3.1	41.1450	41.1325
2	2.2	3.1	45.4556	45.4556
0.5	6	3.5	23.4143	23.3994
0.7	5.6	3.5	26.2203	26.1996
1	5	3.5	30.6507	30.6334
1.3	4.4	3.5	35.3305	35.3233
1.6	3.8	3.5	39.8202	39.8042
2	3	3.5	45.5097	45.5097
0.5	7	4	23.6898	23.6188
0.7	6.6	4	26.4749	26.4418
1	6	4	30.8740	30.8642
1.3	5.4	4	35.5319	35.5112
1.6	4.8	4	39.9446	39.9201
2	4	4	45.5568	45.5568

$$K = 3, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$$

Table 4.2: Networks with $(K, C) = (3, 3)$.

$\boxed{K = 4, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$				

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V(0)$	$V_W(0)$
0.5	4.9	2.95	26.5908	26.4586
0.7	4.5	2.95	30.2549	30.2494
1	3.9	2.95	35.9764	35.9574
1.3	3.3	2.95	41.9580	41.9576
1.5	2.9	2.95	45.7919	45.7918
1.8	2.3	2.95	51.2012	51.1965
0.5	5	3	26.6335	26.5042
0.7	4.6	3	30.2916	30.2854
1	4	3	36.0065	35.9989
1.3	3.4	3	41.9841	41.9875
1.5	3	3	45.8108	45.8107
2	2	3	54.6797	54.6797
0.3	5.6	3.1	23.5052	23.3229
0.6	5	3.1	28.4877	28.4869
1	4.2	3.1	36.0646	36.0518
1.4	3.4	3.1	43.9647	43.9647
1.7	2.8	3.1	49.4720	49.4711
2	2.2	3.1	54.6874	54.6874
0.5	6	3.5	26.9887	26.8898
0.7	5.6	3.5	30.5968	30.5858
1	5	3.5	36.2743	36.2663
1.3	4.4	3.5	42.2056	42.2056
1.6	3.8	3.5	47.7782	47.7749
2	3	3.5	54.7126	54.7126
0.5	7	4	27.2517	27.0193
0.7	6.6	4	30.8310	30.8067
1	6	4	36.4886	36.4785
1.3	5.4	4	42.3745	42.3744
1.6	4.8	4	47.8736	47.8720
2	4	4	54.7351	54.7351

Table 4.3: Networks with $(K, C) = (4, 3)$.

Observations on W_i's

(I) Robustness: An *Admission Price* Policy can only be robust if the W_i do not change much as the offered traffics change. This is not true for our Ω policies.

(II) Our observed W_i satisfy the following restriction

$$W_i \geq W_{i+1}.$$

The fact that $W_i - W_{i+1} \geq 0$ relates to the obvious fact that spare capacity has potential worth as we can use it to carry future calls. In fact if we define $\delta(W_i) = W_i - W_{i+1}$, then we observe

$$\delta(W_i) \geq \delta(W_{i+1})$$

which means that units of spare capacity become more valuable as the system fills up.

(III) As stated in §2.6, states with the same F have similar TEDR values. That explains why *Admission Price* policies with appropriate W_i's policies are good.

(IV) The W_i's get bigger as $\lambda_1 = n1$ increases. This increase translates to more acceptance of single link traffic which returns a bigger reward than the 2-link one.

(V) As it is demonstrated in the following graphs the W_i's do not change much as K increases. This is of practical importance as it suggests that the W_i's from small networks could well be used as a good approximate for the optimal policy in larger networks. We found that by applying the Ω policy derived from networks $(K, C) = (3, 3)$ to $(K, C) = (4, 3)$ and $(K, C) = (5, 3)$ we have excellent results.

In the next figures we present results on the performance of the Ω policy which include:

(a) How W_i change as $\lambda_1 = n1$ increases; for fixed K, C and L.

(b) How W_i change as K increases; for fixed C, L.
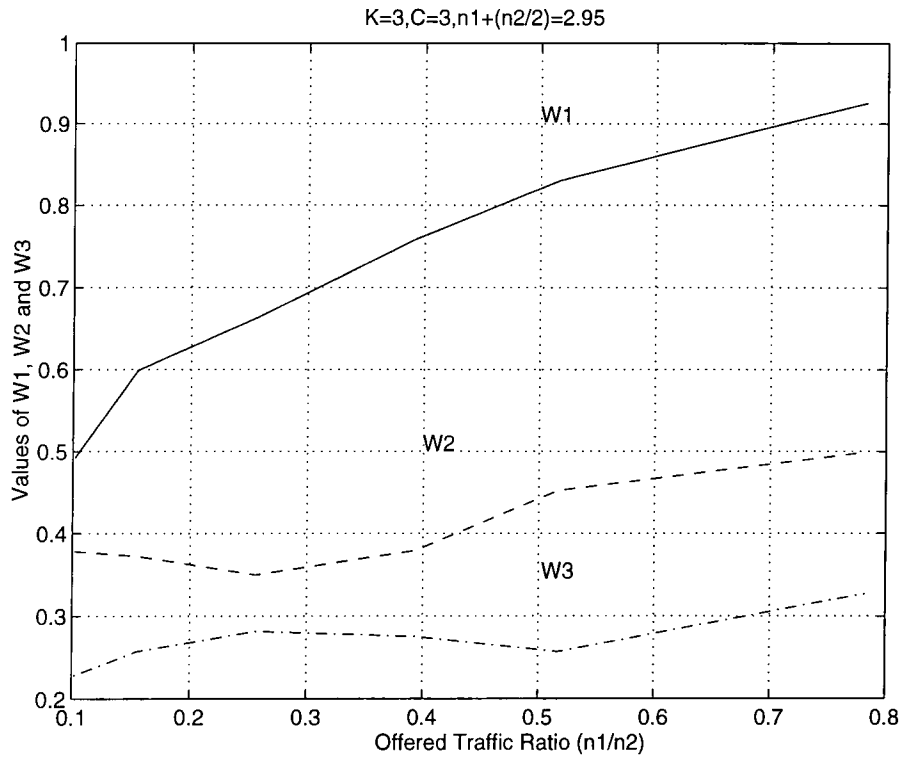
Figure 4.1: Change in w_i as n1 increases; for n1+(n2/2)=2.95.
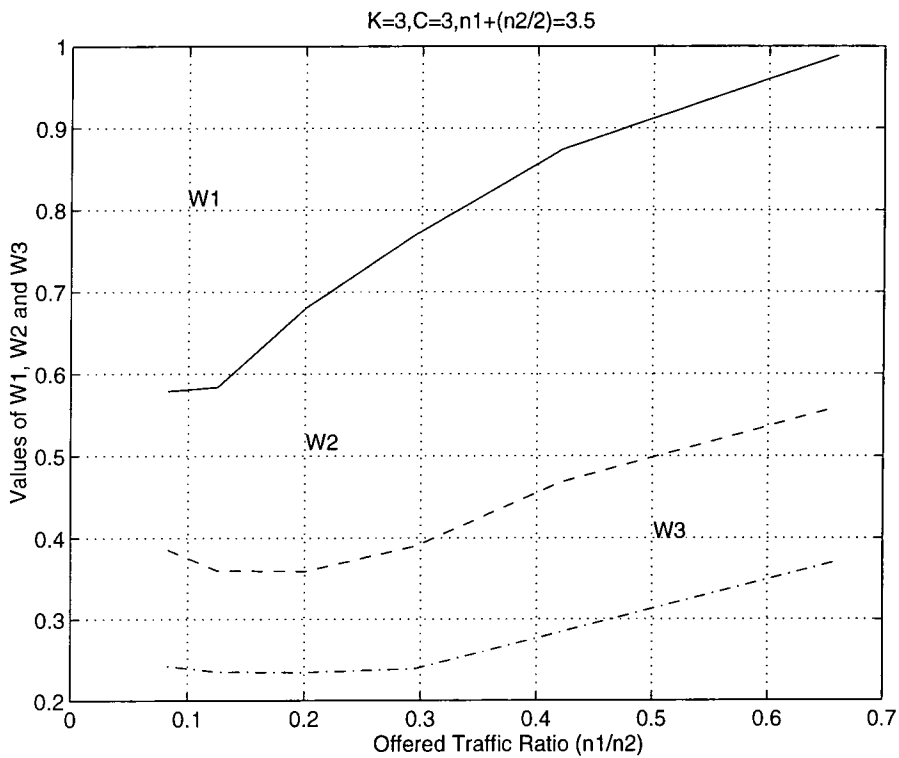
Figure 4.2: Change in w_i as n1 increases; for n1+(n2/2)=3.5.
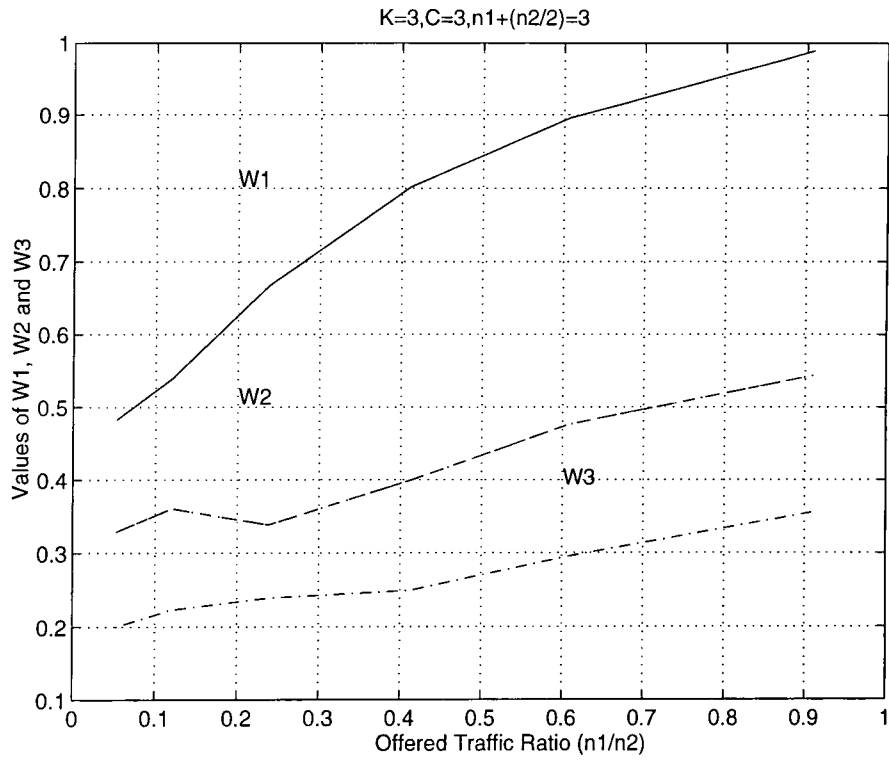
Figure 4.3: Change in w_i as n1 increases; for n1+(n2/2)=3.
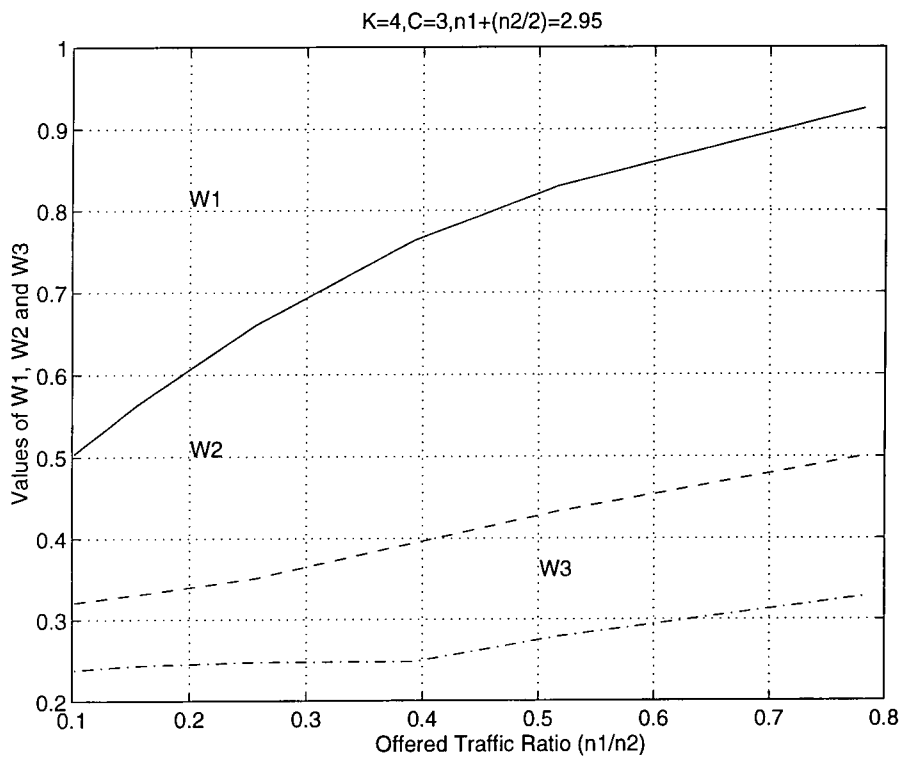
Figure 4.4: Change in w_i as n1 increases; for n1+(n2/2)=2.95.
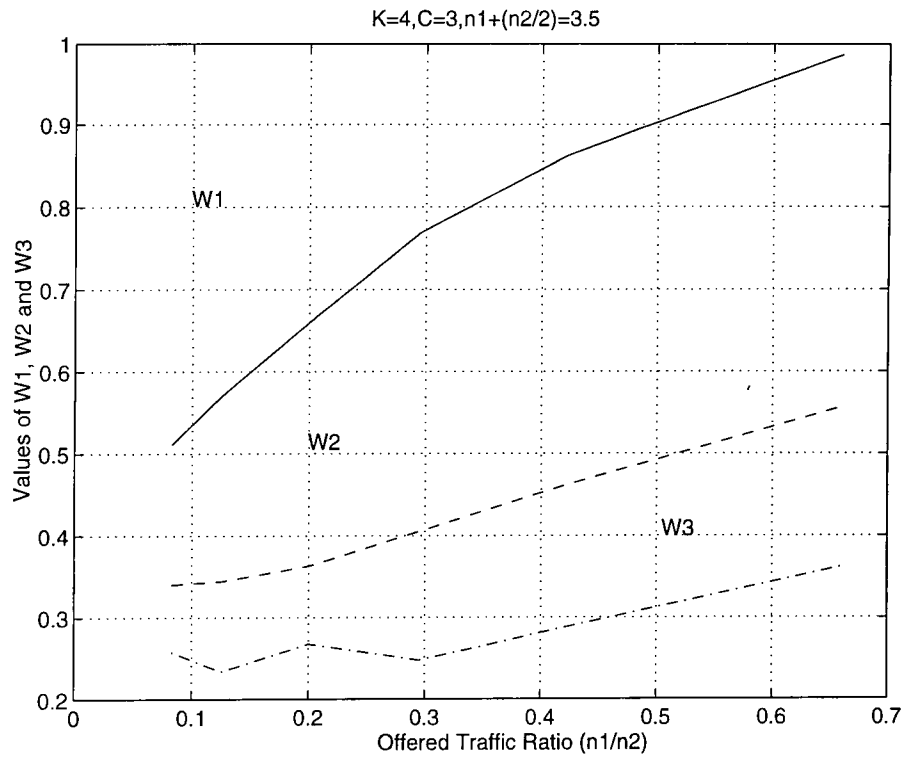
Figure 4.5: Change in w_i as n1 increases; for n1+(n2/2)=3.5
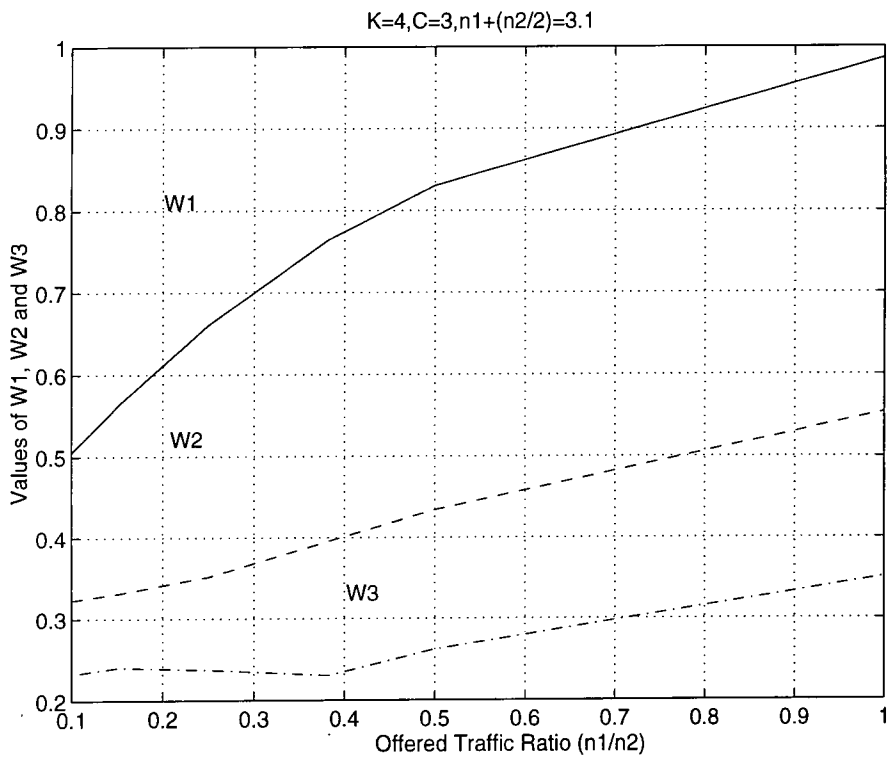
Figure 4.6: Change in w_i as n1 increases; for n1+(n2/2)=3.1.
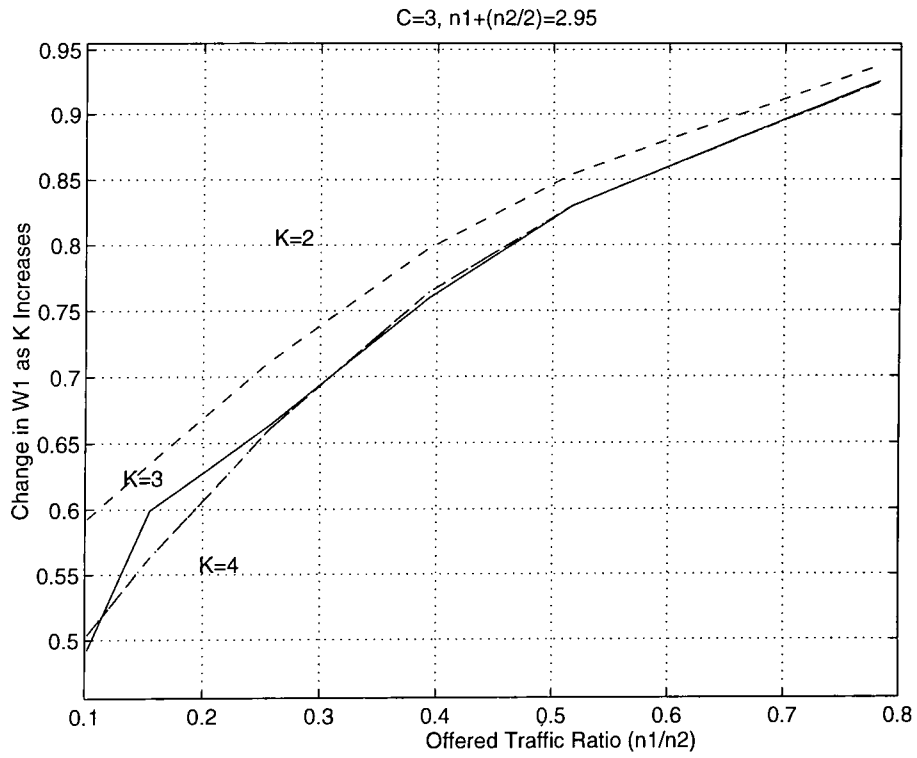
Figure 4.7: Change in w_1 as K increases for n1+(n2/2)=2.95.
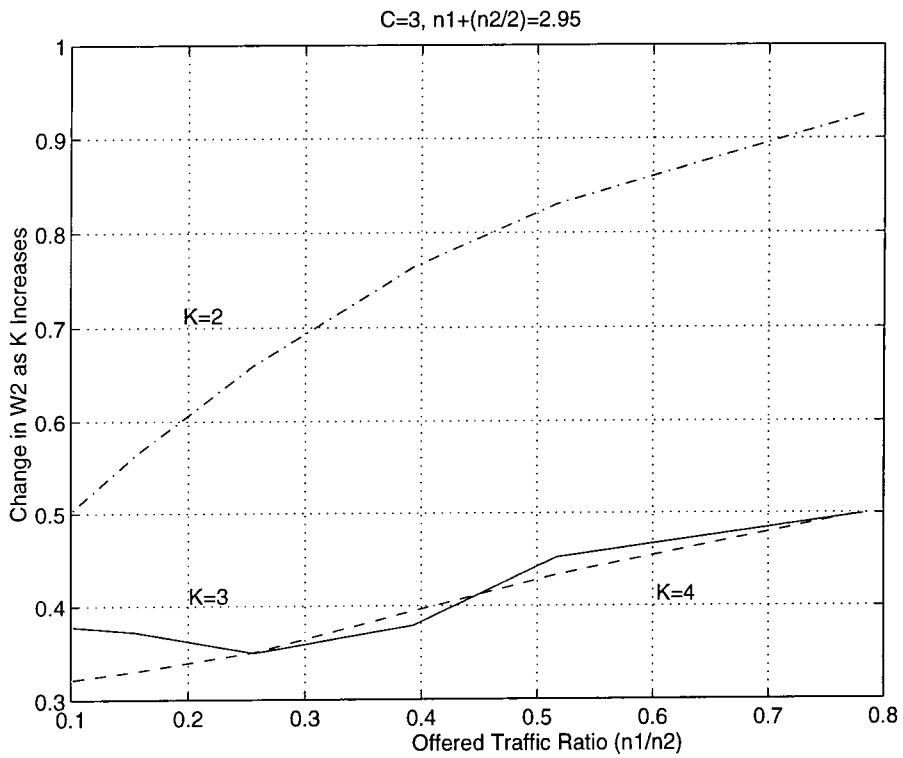
Figure 4.8: Change in w_2 as K increases for n1+(n2/2)=2.95.
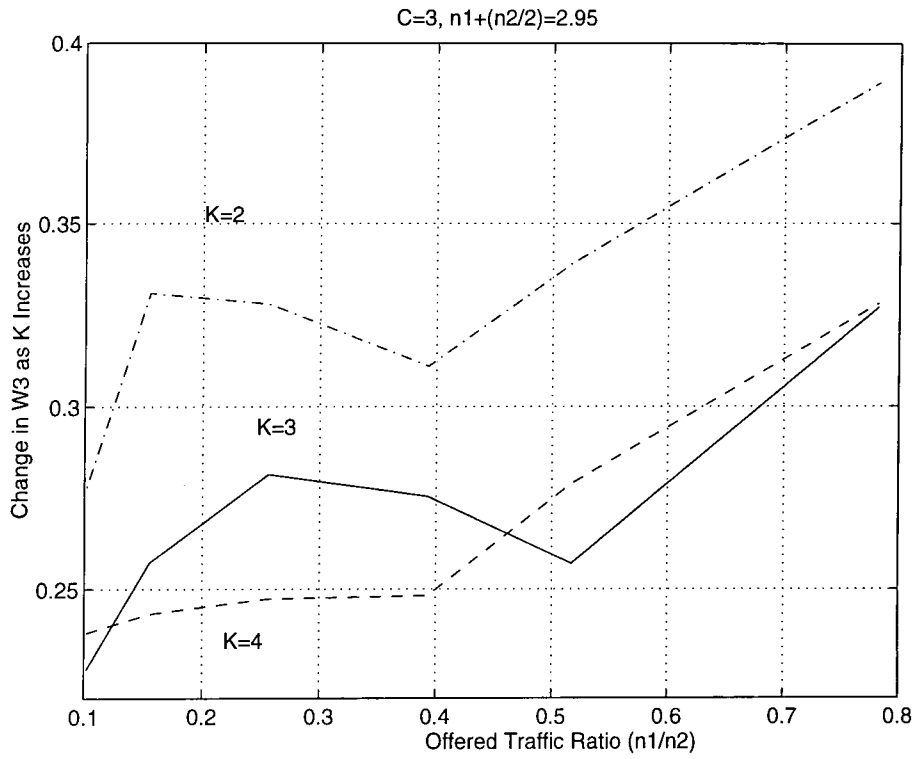
Figure 4.9: Change in w_3 as K increases for n1+(n2/2)=2.95.
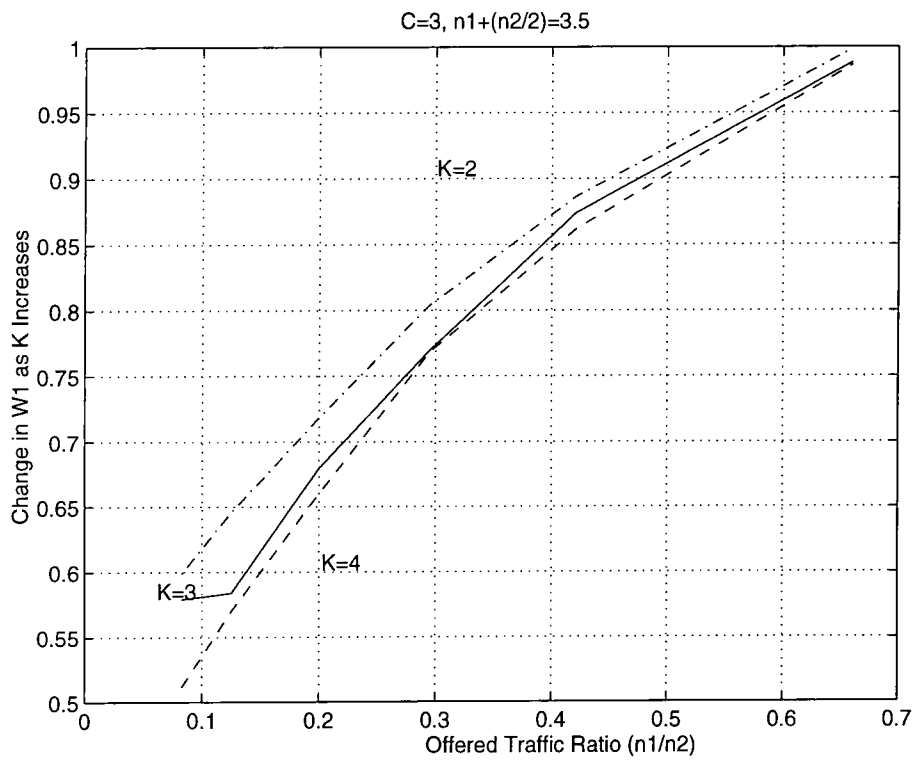
Figure 4.10: Change in w_1 as K increases for n1+(n2/2)=3.5.
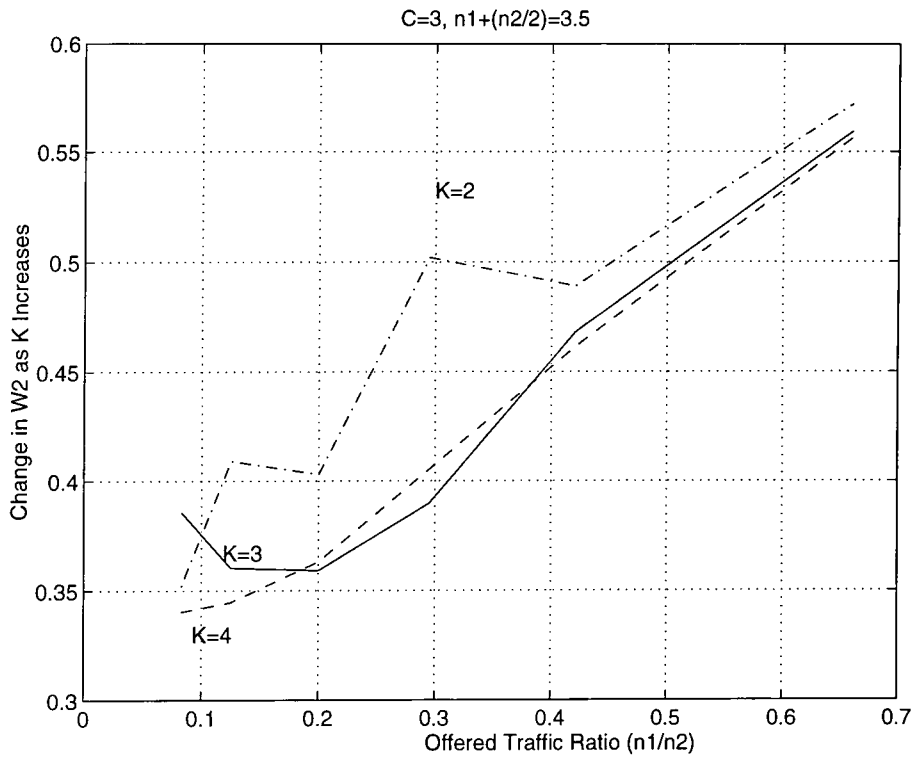
Figure 4.11: Change in w_2 as K increases for n1+(n2/2)=3.5.
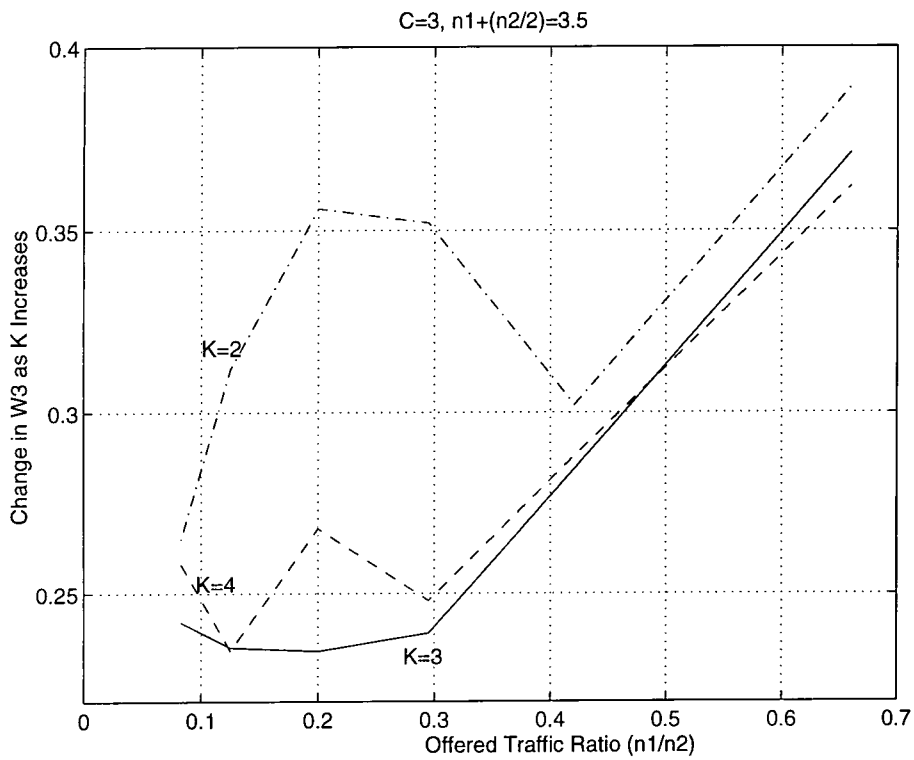
Figure 4.12: Change in w_3 as K increases for n1+(n2/2)=3.5.

Chapter 5

The Reduced State-Space Model

5.1 The Reduced State-Space Model

In this model of a *Symmetric Star* network, Poisson streams of calls arrive at the network requesting routes. There are two types of route: 1-link routes and 2-link routes involving any pair of the single links. 1-link calls arrive at rate λ_1 on each link and 2-link calls arrive at rate $\lambda_2/(K-1)$ on each 2-link pair. A 2-link call on pair (i,j), after its acceptance, is split into 2 independent single link calls on links i and j. Both types of calls have an exponential holding time with mean 1, (Exp(1)). For every 1-link call accepted we earn reward R_1 and for every 2-link call accepted we earn reward R_2. Rewards are bounded and earned immediately. We assume that $R_2 < 2R_1$; see §2.

Definition 11

The state of the network at the time we observe it is the collection of link occupancies in the network, and it is denoted by x, where $x = (x_1, x_2, ..., x_K)$.

For each link i, $0 \leq x_i \leq C$. We say that the link i is *full* when the number of calls in i is C. Calls depart from the network at rate $\sum_i x_i$

105

V_r is the optimal value function (TEDR) for the reduced state-space model. V_{rw} is the optimal value function of the reduced model in which Ω policy is employed. V is the optimal value function (TEDR) for the full model.

The network is modelled as a Markov decision process and the analysis followed is the one described in Chapter 2 for the full model. The *optimality* approach as well as the policy improvement study are the same as for the full model. To calculate the Total Expected Discounted Reward for the reduced model V_r we use the SOR method described in §2.4.3.

The reason we studied the reduced state-space model was that it is a simplification for the full model because the state-space is reduced[1] to $(C + 1)^K$ states which is a great advantage. The reduction of the size of the state-space is a very attractive feature and in this chapter we investigate the performance of the reduced state-space model to see how much it behaves like the full model. We do this by comparing the optimal value function for the reduced state-space model V_r to the optimal value function for the full model V. If the reduced state-space model behaves like the full model we could then work with the simplified version to deduce results on the nature of the optimal policy as well as the general behaviour of the network.

Another reason was to look at the optimal policy and its properties on this simplified version. A natural question to ask was whether the optimal policy is of *Admission Price* form. The results show that it is not, though once again there are very good Ω policies.

In the reduced state-space model the policy is a $\frac{K(K+1)}{2}$-tuple of Boolean variables for each of the $(C + 1)^K$ states, with the first K places for single link calls. In the reduced state-space networks considered in this chapter, the policy is to accept the single link calls if there is room to fit them in, and accept (or reject) 2-link calls according to (a) the optimal policy; and (b) the relevant Ω policy derived by the specific networks.

---

[1] See Table 1.1 in §1.

5.2 Computing and Results

This section describes the way we proceeded with analysing the reduced state-space network. In this analysis three questions were of importance to us:

Question A: Can the reduced state-space model approximate the full model? If yes, how well? To answer this question one has but to compare the optimal value function for the reduced state-space model V_r with the optimal value function for the full model V. Tables 5.1 and 5.2 show some results for networks with $(K, C) = (3,3)$ and also $(4,3)$ which suggest that: (a) The reduced state-space model is a good approximation for the full model in the case with $(K, C) = (3,3)$ and (b) the reduced state-space model is not such a good approximation for the case with $(K, C) = (4,3)$.

Question B: Is the reduced state-space Ω policy a good approximation of the optimal policy in such networks? The results suggest that the *Admission Price* policies (Ω) are an excellent approximation for the optimal policy. In Tables 5.1 and 5.2 we compare the optimal value functions $V_{rw}(0)$ (when Ω is employed) with $V_r(0)$.

Question C: Is the Ω policy from the reduced state-space model a good approximation for the optimal policy of the full model and vice-versa? To answer this question one has but to employ the Ω policy of one model to the other and look at the results. The results suggest that the Ω policy of the reduced state-space model is a very good approximation for the optimal policy of the full model (see Table 5.5) and vice-versa (see Tables 5.3 and 5.4). In both cases very little improvement is left to be performed by applying them; in some cases none.

The results in the following Tables are presented in groups of three categories (a) The size of the network i.e. (K, C), (b) for a specific size of a network we keep L fixed; we have focused on examples for networks where L is fixed and 'critical' for the link i.e. nearly full, full, overloaded etc; see §2.3.1. (c) For a specific L we consider various arrival rates. Note that in the tables and figures following $\nu_1 = \lambda_1$ and $\nu_2 = \lambda_2$. All the other quantities have been defined in §2.

For an explanation of why α is not exactly 0.8 see the remark on page 50.

$$\boxed{K = 3, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_r(0)$	$V_{rw}(0)$	$V(0)$
0.5	4.9	2.95	22.6982	22.6379	22.9963
0.7	4.5	2.95	25.6198	25.6107	25.8420
1	3.9	2.95	30.1459	30.1344	30.3238
1.3	3.3	2.95	34.9453	34.9453	35.0391
1.5	2.9	2.95	38.0624	38.0624	38.1230
1.8	2.3	2.95	42.5160	42.5139	42.5410
0.5	5	3	22.7407	22.6832	23.0410
0.7	4.6	3	25.6583	25.6483	25.8825
1	4	3	30.1782	30.1678	30.3579
1.3	3.4	3	34.9740	34.9740	35.0693
1.5	3	3	38.0850	38.0850	38.1477
2	2	3	45.4275	45.4275	45.4389
0.3	5.6	3.1	20.2407	20.1827	20.6261
0.6	5	3.1	24.2523	24.2523	24.4902
1	4.2	3.1	30.2395	30.2310	30.4228
1.4	3.4	3.1	36.5948	36.5948	36.6757
1.7	2.8	3.1	41.1052	41.0954	41.1450
2	2.2	3.1	45.4431	45.4431	45.4556
0.5	6	3.5	23.1132	23.0673	23.4143
0.7	5.6	3.5	25.9784	25.9615	26.2203
1	5	3.5	30.4535	30.4491	30.6507
1.3	4.4	3.5	35.2131	35.2130	35.3305
1.6	3.8	3.5	39.7572	39.7465	39.8202
2	3	3.5	45.4927	45.4927	45.5097
0.5	7	4	23.3897	23.3582	23.6898
0.7	6.6	4	26.2212	26.1921	26.4749
1	6	4	30.6668	30.5850	30.8740
1.3	5.4	4	35.3976	35.3896	35.5319
1.6	4.8	4	39.8709	39.8556	39.9446
2	4	4	45.5354	45.5354	45.5568

Table 5.1: Reduced state-space networks with $(K, C) = (3, 3)$.

$$\boxed{K = 4, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_r(0)$	$V_{rw}(0)$	$V(0)$
0.5	4.9	2.95	30.2229	30.1657	26.5908
0.7	4.5	2.95	34.1066	34.0994	30.2459
1	3.9	2.95	40.1520	40.1466	35.9764
1.3	3.3	2.95	46.5446	46.5446	41.9580
1.5	2.9	2.95	50.6831	50.6831	45.7919
1.8	2.3	2.95	56.6035	56.6006	51.2012
0.5	5	3	30.2886	30.2275	26.6335
0.7	4.6	3	34.1584	34.1505	30.2916
1	4	3	40.1977	40.1928	36.0066
1.3	3.4	3	46.5847	46.5847	41.9841
1.5	3	3	50.7148	50.7145	45.8108
2	2	3	60.4732	60.4732	54.6797
0.3	5.6	3.1	26.9180	26.8861	23.5052
0.6	5	3.1	32.3074	32.3062	28.4877
1	4.2	3.1	40.2863	40.2810	36.0646
1.4	3.4	3.1	48.7399	48.7399	43.9647
1.7	2.8	3.1	54.7262	54.7154	49.4720
2	2.2	3.1	60.4958	60.4958	54.6874
0.5	6	3.5	30.7907	30.7529	26.9887
0.7	5.6	3.5	34.5873	34.5786	30.5968
1	5	3.5	40.5903	40.5842	36.0646
1.3	4.4	3.5	46.9219	46.9221	43.9647
1.6	3.8	3.5	52.9413	52.9354	49.4720
2	3	3.5	60.5691	60.5691	54.6874
0.5	7	4	31.1750	31.1527	27.2517
0.7	6.6	4	34.9137	34.8959	30.8310
1	6	4	40.8958	40.8810	36.4886
1.3	5.4	4	47.1761	47.1752	42.3745
1.6	4.8	4	53.1085	53.0931	47.8736
2	4	4	60.6341	60.6341	54.7351

Table 5.2: Reduced state-space networks with $(K, C) = (4, 3)$.

		$K = 3, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$		
ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_1(0)$	$V_r(0)$
0.7	4.5	2.95	25.6107	25.6198
1	3.9	2.95	30.1344	30.1459
1.3	3.3	2.95	34.9453	same
1.5	2.9	2.95	38.0624	same
1.8	2.3	2.95	42.4990	42.5160
0.5	5	3	22.6288	22.7407
0.7	4.6	3	25.6483	25.6583
1	4	3	30.1678	30.1782
1.5	3	3	38.0850	same
2	2	3	45.4275	same
0.3	5.6	3.1	20.1827	20.2407
0.6	5	3.1	24.2523	same
1	4.2	3.1	30.2314	30.2395
1.7	2.8	3.1	41.0954	41.1052
2	2.2	3.1	45.4431	same

Table 5.3: Employing the Ω of §4 in a reduced state-space network with $(K, C) = (3, 3)$.

		$K = 4, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$		
ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_1(0)$	$V_r(0)$
0.3	5.6	3.1	26.8861	26.9067
0.6	5	3.1	32.3062	32.3070
1.4	3.4	3.1	48.7399	same
1.7	2.8	3.1	54.7154	54.7224
2	2.2	3.1	60.4958	same
0.5	6	3.5	30.7529	30.7757
1	5	3.5	40.5842	40.5880
1.3	4.4	3.5	46.9219	same
1.6	3.8	3.5	52.9354	52.9392
2	3	3.5	60.5691	same
0.5	7	4	30.9412	31.0757
0.7	6.6	4	34.8959	34.9137
1	6	4	40.8810	40.8898
1.3	5.4	4	47.1752	47.1758
2	4	4	60.6341	same

Table 5.4: Employing the Ω of §4 in a reduced state-space network with $(K, C) = (4, 3)$.

$$K = 3, C = 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_2(0)$	$V(0)$
0.7	4.5	2.95	22.8750	22.9963
1	3.9	2.95	30.2914	30.3238
1.3	3.3	2.95	35.0388	35.0391
1.5	2.9	2.95	38.1223	38.1230
1.8	2.3	2.95	42.5411	same
0.5	5	3	22.9215	23.0411
0.7	4.6	3	25.8702	25.8825
1	4	3	30.3245	30.3580
1.5	3	3	38.1464	same
2	2	3	45.4389	same
0.3	5.6	3.1	20.5218	20.6262
0.6	5	3.1	24.8820	24.4490
1	4.2	3.1	30.3965	30.4229
1.7	2.8	3.1	41.1325	41.1450
2	2.2	3.1	45.4557	same

Table 5.5: Employing the Ω of §5 in a full network with $(K, C) = (3, 3)$.

In the following figures we can see that the behaviour of the W_i for the reduced model is similar to those of the full model:

(I) Robustness: The *Admission Price* policy for the reduced state-space model is not robust as w_i do change as the offered traffics change.

(I) Our observed W_i satisfy the following restriction $W_i \geq W_{i+1}$; see also §4.2.

(III) As stated in §2.6, states with the same F have similar TEDR values. That explains why our policies are good.

(IV) The W_i's get bigger as ν_1 increases. This increase translates to more acceptance of single link traffic which returns a bigger reward than the 2-link one.

(IV) The W_i's do not change significantly as K increases. The practical importance of this is discussed in §4.2. By applying the Ω policy derived from networks $(K, C) = (3, 3)$ to $(K, C) = (4, 3)$ and $(K, C) = (5, 3)$ we have excellent results.

Figures 5.9 and 5.10 demonstrate clearly the similarity in the Ω policies for the full and reduced model; see also Question C. In the figures $n1 = \lambda_1, n_2 = \lambda_2$.
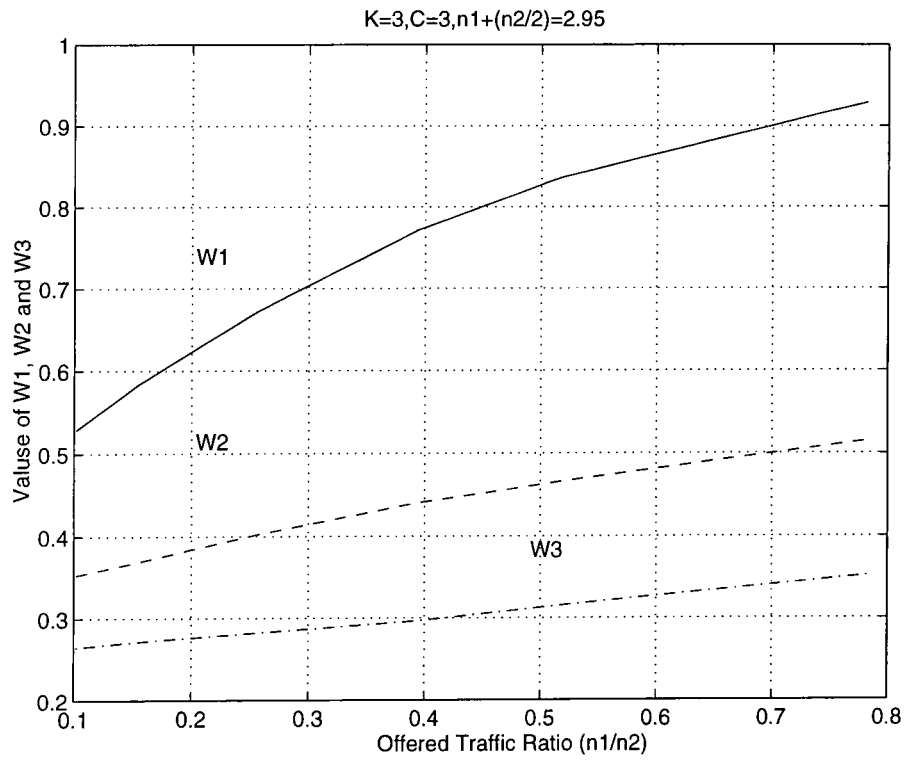
Figure 5.1: Change in w_i as n1 increases; for n1+(n2/2)=2.95.
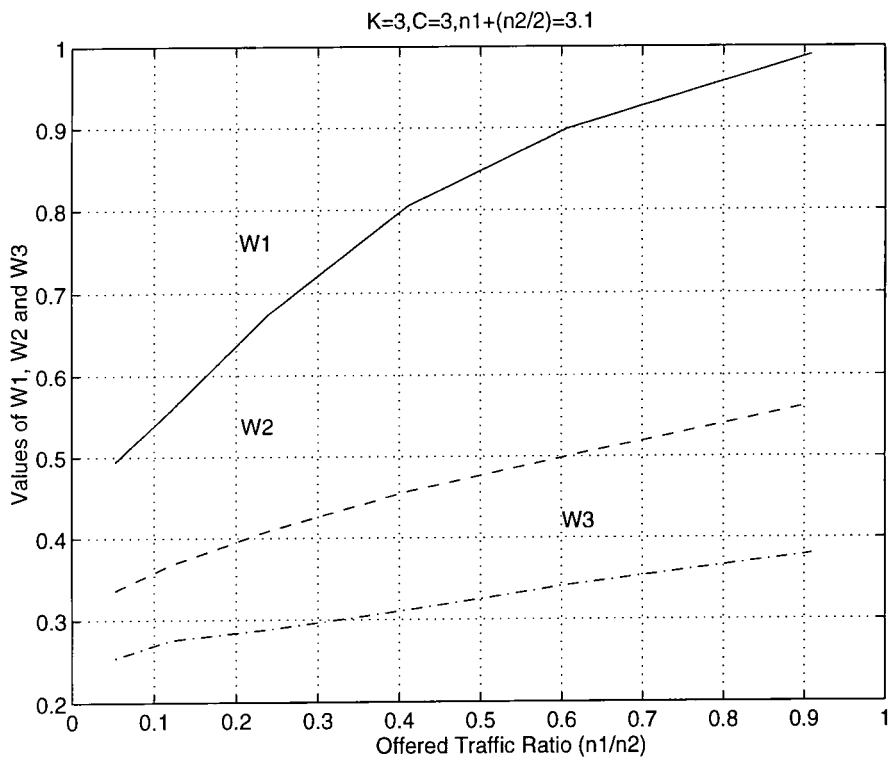
Figure 5.2: Change in w_i as n1 increases; for n1+(n2/2)=3.1.

112
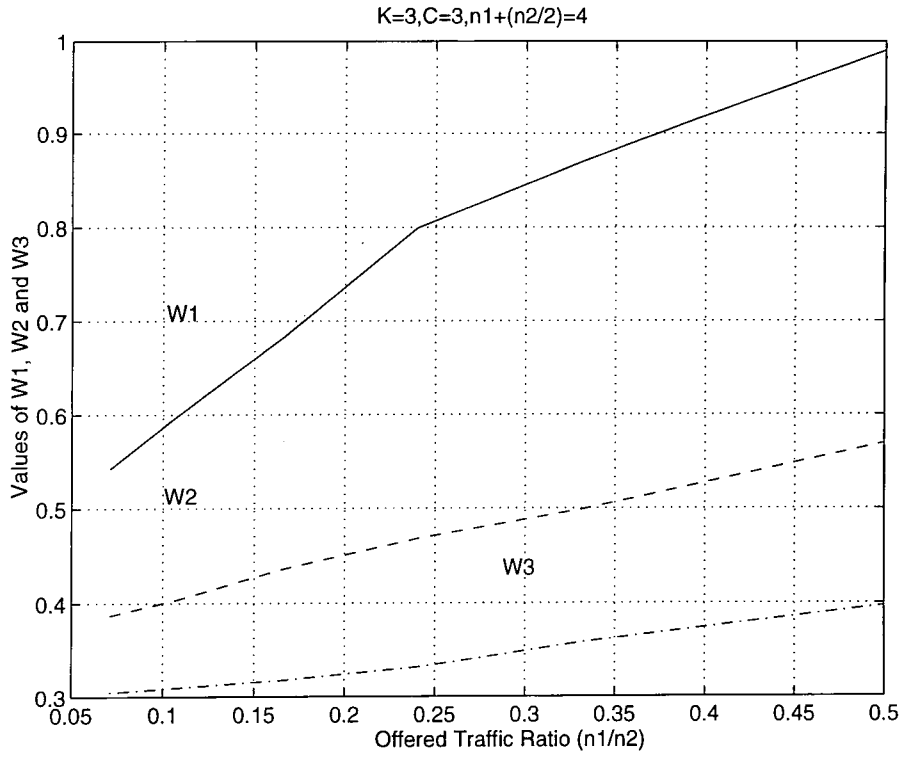
Figure 5.3: Change in w_i as n1 increases; for n1+(n2/2)=4.
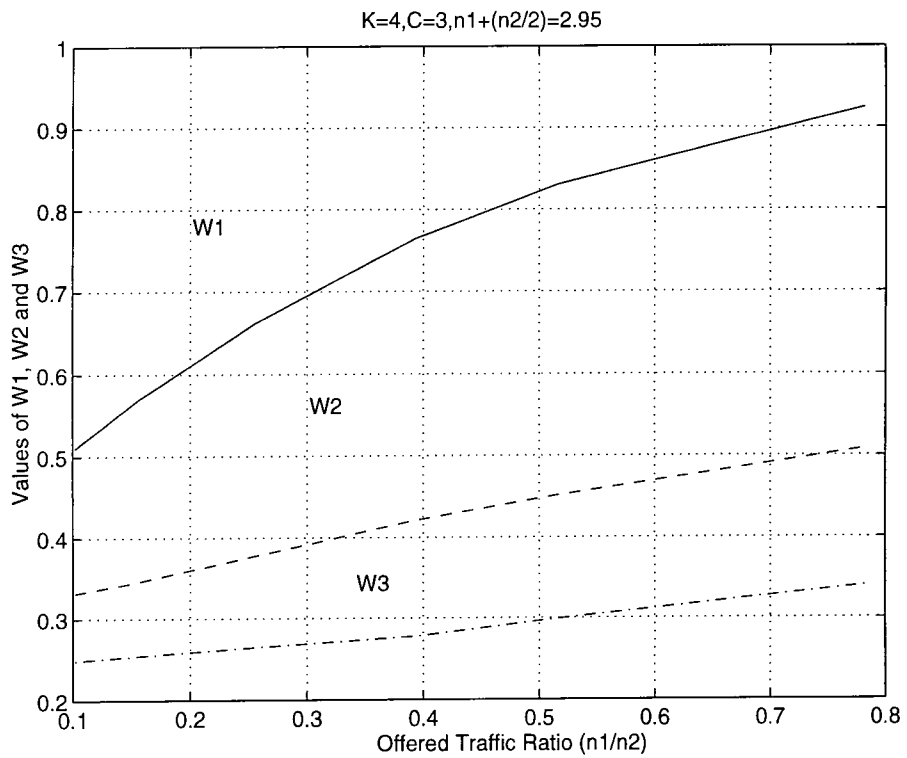
Figure 5.4: Change in w_i as n1 increases; for n1+(n2/2)=2.95.
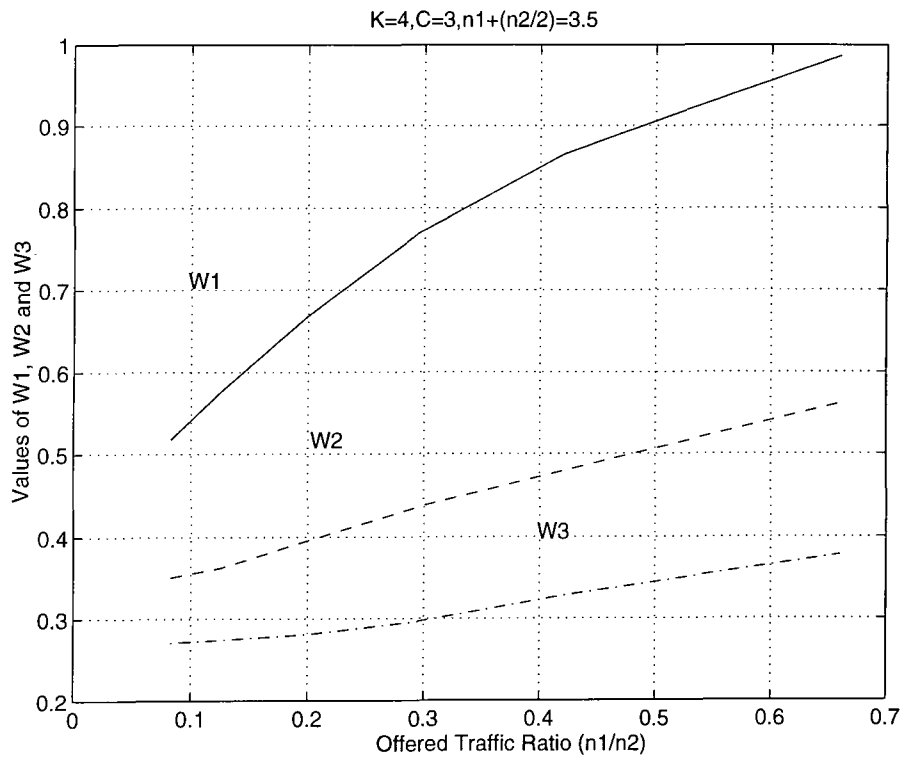
Figure 5.5: Change in w_i as n1 increases; for n1+(n2/2)=3.5.
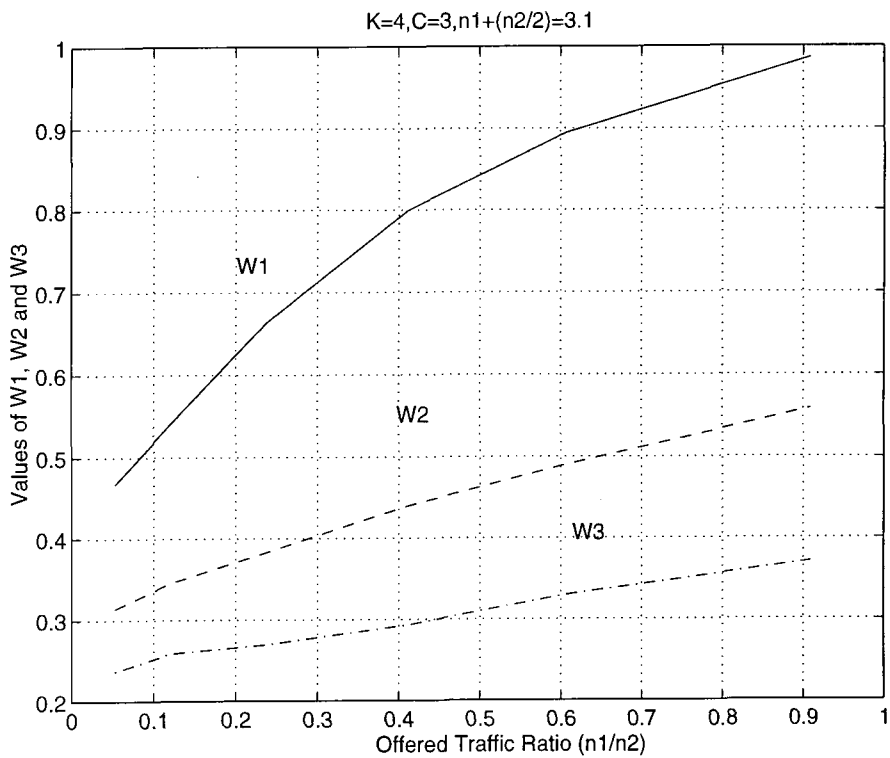
Figure 5.6: Change in w_i as n1 increases; for n1+(n2/2)=3.1.

114.
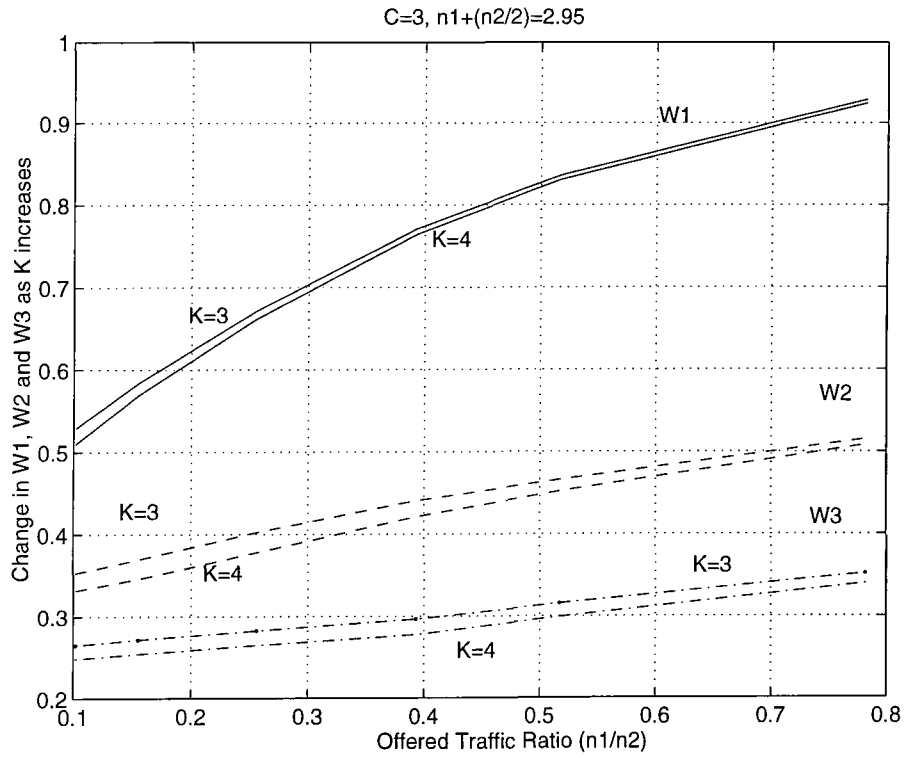
Figure 5.7: Change in w_i as K increases; for n1+(n2/2)=2.95.
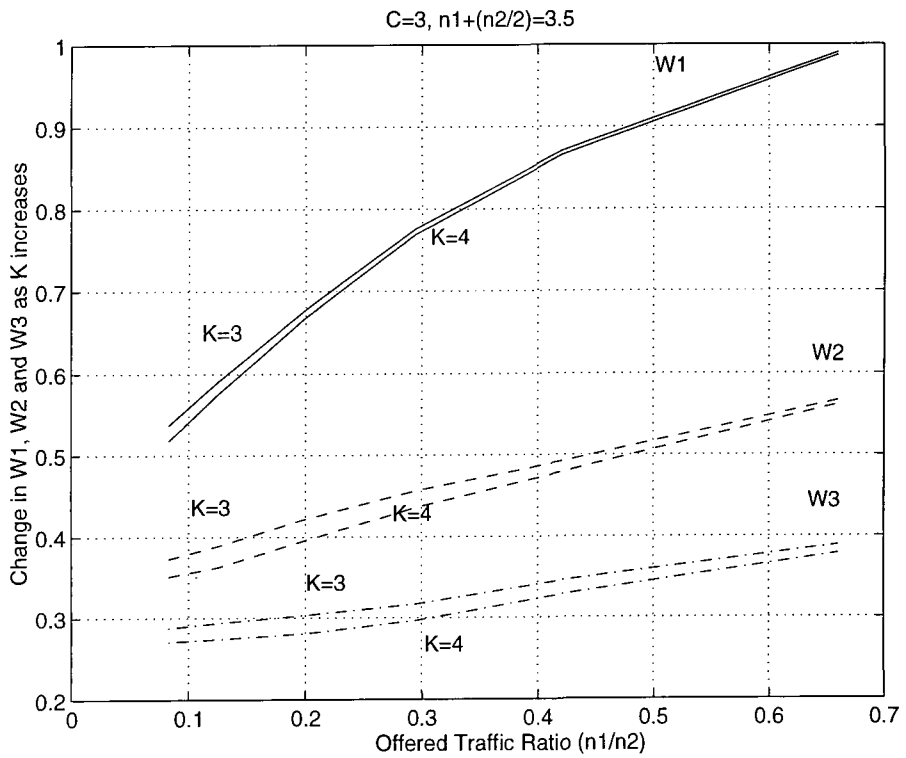
Figure 5.8: Change in w_i as K increases; for n1+(n2/2)=3.5.
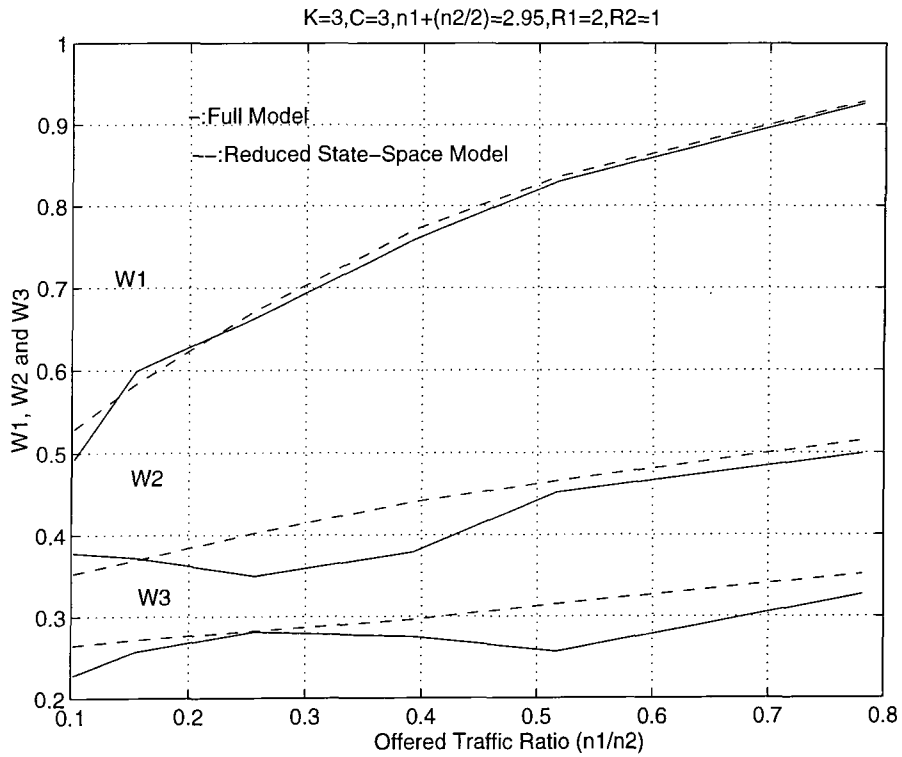
Figure 5.9: Comparing the Ω policies of the full and reduced state-space networks with $(K, C) = (3, 3)$.
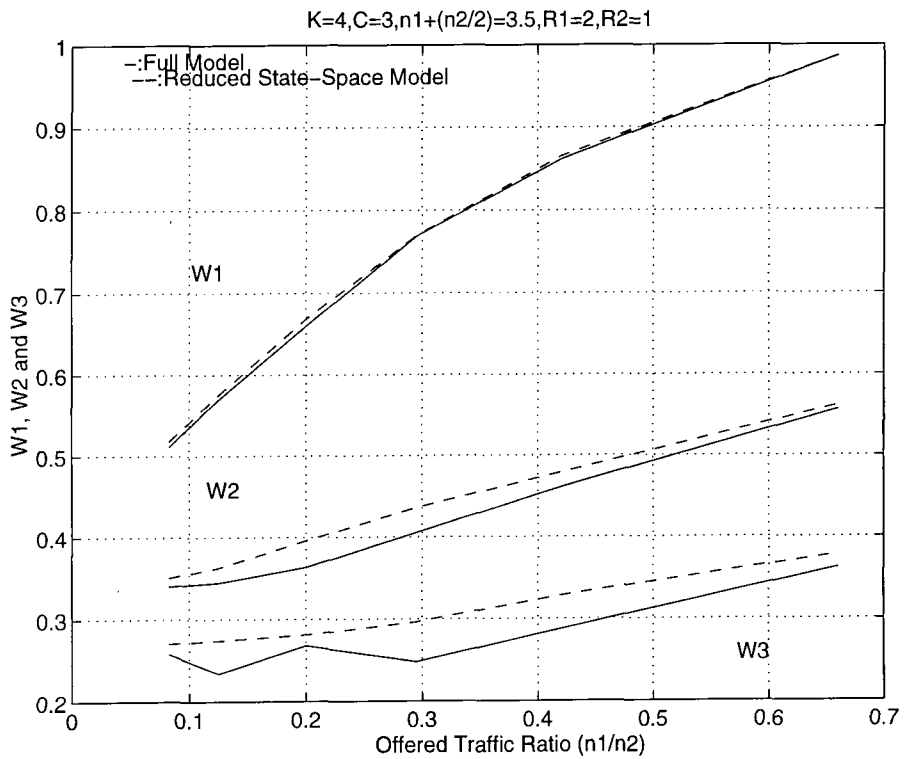
Figure 5.10: Comparing the Ω policies of the full and reduced state-space networks with $(K, C) = (4, 3)$.

5.3 Optimal Policies and Properties

For the reduced state-space networks considered in this chapter, we study, as before, policies that always accept 1-link calls when there is room to fit them in.

In the following examples, we describe the exact optimal policy for networks with $(K = 3, C = 3)$. In the following description we do not include: (a) the cases in which the optimal policy accepts the 2-link calls when there is room to fit them in; (b) the cases in which the network is full.

Let $F = (F_1, F_2, F_3)$ denote the number of free circuits on links X, Y and Z respectively. A network state is represented by a triple (X,Y,Z). For example (012) describes the state of the network with one call on link Y and 2 calls on link Z.

The quantities $\lambda_1, \lambda_2, R_1, R_2$, and α are all defined in §2.1, §2.2 and §2.3. Note that the α values given are for the continuous time process and not the modified values used in the uniformised *optimality equation*; see §2.3.2.

In what follows we will a consider a state $z \geq \bar{z}$ whenever $F_z \leq F_{\bar{z}}$. For example a state z with $F_z = (1,1,1)$ is considered $z > \bar{z}$ where $F_{\bar{z}} = (1,1,3)$ i.e. a very large state in terms of link occupancy has a very small number of free circuits.

In the following examples the size of the state-space is 64 states; see Table 1.1.

The properties we look at in the reduced state-space networks are the following:

Property A: For the reduced state-space networks the policy by definition depends upon the state or equivalently, the free circuits. The crucial point is that the optimal policy isn't an Ω policy because the state of the link Z matters in deciding acceptance of a 2-link route in XY and therefore the optimal policy cannot be of the form: accept if $W_{F_X} + W_{F_Y} \leq R_2$; see Property B.

Property B: [Property P2 in Key (1990)] For calls which are disjoint, i and j say, and thus could be widely separated in a network, in general, the more type i calls in progress, the less likely we are to reject type j calls, and vice-versa; disjoint calls are for example XY and Z calls. This property does hold for the reduced state-space networks considered; see Examples 5.1 and 5.4.

Property C: Monotonicity. If an arrival for a 2-link call on route k is rejected in state $\bar{z}$, where $z \geq \bar{z}$, then it will also be rejected in states z. Our results suggest that this property does not hold in general; see Examples 5.1, 5.2 and 5.4.

Property D: [Property P1 in Key (1990)] If we reject a type i call in state z, then we reject it in state $z + k$ for calls i and k which are distinct and not disjoint; not disjoint calls are for example XY and YZ calls. This property does hold in general in the reduced state-space networks considered; see Example 5.2.

Property E: Weak Monotonicity [Assumption C1 in Key (1990)] If we reject a type k call in state z, then we reject a type call k in state $z + e_k$. This property does hold for the reduced state-space networks considered; see Example 5.2.

Example 5.1

$\lambda_1 = 0.5, \lambda_2 = 4.9$

The optimal policy rejects a 2-link call on pair XY in states with $F=$ (1,1,3), (1,1,2) but not in states with $F=$(1,1,0), (1,1,1);

If we denote states (220), (221), (222) and (223) with (1,1,3), (1,1,2), (1,1,1), (1,1,0) free circuits as z_0, z_1, z_2, z_3, where $z_0 < z_1 < z_2 < z_3$, we see that rejection in z_0, z_1 does not coincide with rejection in z_2, z_3 and therefore Property C does not hold.

We also note that for the distinct calls XY and Z, a rejection of XY in state (221) with (1,1,2) free circuits coincides with rejection of XY in state (220) with (1,1,3) free circuits. Continuing the analysis for all states confirms that Property B holds for this example.

Example 5.2

$\lambda_1 = 1.8, \lambda_2 = 2.3$

The optimal policy rejects a 2-link call on pair XY in states with $F=$ (1,1,3), (1,1,2), (1,1,1), (1,1,0); (3,1,3), (3,1,2), (3,1,1), (3,1,0); (1,2,3), (1,2,2), (1,2,1), (1,2,0); (2,1,3), (2,1,2), (2,1,1), (2,1,0), but not in (2,2,1), (2,2,0).

Rejection of XY in state (110) with (2,2,3) free circuits coincides with rejection in state (220) with (1,1,3) free circuits. That shows that Property E does hold.

If we consider XY and YZ calls which are not disjoint, rejection of XY in state (211) with (1,2,2) free circuits coincides with rejection of XY in state (222) with (1,1,1) free circuits and that suggests that Property D does hold.

Example 5.3

$\lambda_1 = 0.7, \lambda_2 = 5.6$

The optimal policy rejects a 2-link call on pair XY in states with F= (1,1,3), (1,1,2), (1,1,1), (1,1,0); (2,1,3); (1,2,3).

Example 5.4

$\lambda_1 = 1, \lambda_2 = 6$

The optimal policy rejects a 2-link call on pair XY in states with F= (1,1,3), (1,1,2), (1,1,1), (1,1,0); (3,1,3), (3,1,2); (1,2,3), (1,2,2), (1,2,1), (1,2,0); (2,1,3), (2,1,2), (2,1,1), (2,1,0); (1,3,3), (1,3,2) but not in (3,1,1), (3,1,0); and (1,3,1), (1,3,0).

Consider disjoint calls XY and Z. Rejection of XY in states with (3,1,2) free circuits coincides with rejection in states with (3,1,3) free circuits and therefore Property B does hold.

Observation 1

Due to the symmetry assumption of our networks it is easy to deduce the optimal policy for all 2-link routes by just looking at its description for one of the possible 2-link routes. For example knowing that a XY call is rejected in (1,1,2) and (1,1,3) can help us deduce that rejections in XZ will be in (1,2,1) and (1,3,1) i.e. by permuting the F_2 with the F_3.

Note: In various states all, two or one of the 2-link call types are rejected. We focus our analysis on 2-link call rejection on a particular pair in a states with particular F without restricting possible rejections of the other pairs in those states.

5.4 Using the Ω policy from the full networks

In this example we present the results of numerical optimisation in which an initial estimate for the W is taken from the full model case, and the Nelder-Mead (1965) simplex algorithm from the MATLAB OPTIMISATION TOOLBOX was applied in an attempt to find better W_i; see also §3.1.1.

The results from a variety of examples were not encouraging as in no case did the Nelder-Mead algorithm find W_i as good as those derived from knowledge of the optimal value function for the reduced model.

$\boxed{\text{Example 5.5}}$

For the case $(K = 3, C = 3)$, $(\lambda_1, \lambda_2) = (0.5, 4.9)$ of the reduced state-space model with $V_r = 22.6982$, we first take as an initial estimate a $\mathbf{W}$ from the full model. $V(0)$ is 22.5881 in the first iteration. Then it improves (when it does) to:

(a) 22.638 for $\mathbf{W} = (0.5416, 0.3788, 0.288)$ and initial estimate $(0.49, 0.378.\ 0.288)$ from the full model.

(b) 22.638 for $\mathbf{W} = (0.55, 0.45, 0.35)$ and initial estimate $(0.5, 0.4, 0.3)$.

(c) 22.5881 for $\mathbf{W} = (0.36, 0.27, 0.180)$ and initial estimate $(0.4, 0.3, 0.2)$.

This example as well as 14 more demonstrate that the task of directly finding good W_i with standard numerical optimisation routines is not giving any results.

5.5 The Ott and Krishnan 'costs'

In §1.7.1, the *Separable Routing* Scheme of Ott and Krishnan (1985,1986) was presented. The scheme was carried out by considering the value cost of adding a call to a link. As the links were assumed independent, the cost of adding a multilink call to links $k_1, .., k_m$, in the respective states $j_1, .., j_m$ is given by

$$\sum_{i=1}^{m} \Delta(k_i, j_i), \text{ where } \Delta(k, j) = \frac{E(\lambda, C)}{E(\lambda, j)}, \quad 0 \le j \le C,$$

and each link k has C circuits and is offered a Poisson load of λ Erlangs.

For the reduced state-space model of §5.1, the cost of adding a 2-link call to links i and j will then be given by $W = \Delta(i, x_i) + \Delta(j, x_j)$, where x_k is the occupancy of link k. Krishnan and Ott's (1985,1986) routing scheme (admission policy) for a 2-link call is now specified as follows: when a 2-link call arrives requesting route on (i, j), the cost in the current network state, of each admissible pair that has at least one free circuit on each link in the pair is calculated by the above expression. If W exceeds R_2 (the cost of a lost 2-link call), then the call is rejected; otherwise it is accepted.

Remember that our *Admission Price* Routing Scheme accepts a 2-link call on a pair (i, j) whenever $W_{F_i} + W_{F_j} \leq R_2$, where F_k denotes the free circuits on link k.

It is obvious that there is a likeness in the ideas behind the two schemes, and it would be, therefore, interesting to compare the values of Ott and Krishnan's costs with our Ω policy values.

For the *Symmetric* reduced state-space network with K links of capacity C, and with arrival rates for single link calls and 2-link calls to be λ_1, λ_2 the Poisson demand for circuits on every link is $\lambda = L$, where L is given by (2.6).

Example 5.6

For a *Symmetric* network with $K = 3, C = 3, \lambda_1 = 0.5$ and $\lambda_2 = 4.9$, the Ott and Krishnan $\Delta(k, x_k)$ are shown in Table 5.5. In the same table the *Admission Price* policies W's for both the full and the reduced state-space network are given.

Calls present (x_k)	$\Delta(k, x_k)$	W_{full}	$W_{reduced}$
3	1	2	2
2	0.65	0.4924	0.5292
1	0.45	0.3781	0.3529
0	0.34	0.2881	0.2650

Table 5.5: Separable Routing and Admission Price Policies

Chapter 6

Asymmetric Models

6.1   Introduction

In this chapter we consider star-shaped, circuit-switched Loss networks which consist
of K links of capacity C_i, $i = 1, 2, .., K$, linked through a common node. This is
known as an *Asymmetric Star* network. We also relax the assumptions about the
offered traffic rates being the same for all 1-link routes and for 2-link routes and also
the assumption that all types of traffic have the same mean holding time.

The reason we looked at *Asymmetric* networks is that they are very complicated
and there are no good theoretical results existing even for the single link case when
distinct call types have different mean holding times. We present in this chapter
some results of research on the behaviour of small *Asymmetric* networks as well as
properties of the optimal policies in some of them. We do not in this thesis tackle
the subject theoretically.

6.2   The Model

Calls requesting routes arrive at the network randomly and according to a Poisson
distribution. There are two types of route: 1-link routes and 2-link routes involving

any pair of the single links. Requests for 1-link routes arrive on link i at rate λ_{1i}, and requests for 2-link routes arrive at rate $\lambda_{2(i,j)}/(K-1)$ on each 2-link pair (i,j).

1-link routes have an exponential holding time with mean $\mu_1 = 1$, and 2-link routes have an exponential holding time with mean μ_2.

The number of all possible pairs (i,j), where $i \leq j$, is $\beta = \frac{K(K+1)}{2}$. The number of all possible 2-link pairs (i,j), where $i < j$ is $\gamma = \frac{K(K-1)}{2}$. All the 2-link pairs (i,j) are indexed with the formula $l(i,j)$ given in Lemma 1 in §2. The number of 1-link calls on link i is denoted by x_i. The number of 2-link calls on pair (i,j) with index number $l(i,j)$ is denoted by w_l. The number of 1-link calls in the network is denoted by x and the number of 2-link calls in the network is denoted by w. The state of the network at the time we observe it is denoted by (x,w). The above quantities are all defined in §2. For every 1-link call carried we earn reward R_1 and for every 2-link call carried we earn reward R_2. Rewards are bounded and earned immediately. The unit of reward per unit time for carrying a 2-link call is now R_2/μ_2 (it was R_2).

1-link calls depart from link i at rate x_i; and 2-link calls depart from pair (i,j) at rate w_l/μ_2.

The operation of such networks is considered as a Markov decision process in which we investigate the performance of different policies using an *optimality equation* in order to maximise the **Total Expected Discounted Reward (TEDR)**[1]. The *optimality equation* for calculating the optimal policies and the TEDR as well as the iterative methods used to do so, are also presented and discussed in §2. In fact, the networks considered in this section only differ from those of §2 in that the rate of events and therefore the transition probabilities are slightly different. The state-space is also different. For example for networks with $K = 3$ and capacities $C = (3,3,4)$, $C = (3,3,5)$ and $C = (3,3,6)$ the sizes are respectively 477, 622 and 768 possible states.

---

[1]Defined in §2.

6.3 Rates of Events and Transition Probabilities for Asymmetric Networks

1-link calls arrive on the network at rate ν_1, where $\nu_1 = \sum_{i=1}^{K} \lambda_{1i}$.

2-link calls arrive on the network at rate ν_2, where $\nu_2 = \sum_{l=1}^{\gamma} \frac{\lambda_{2l}}{(K-1)}$.

1-link calls depart from the network at rate $\nu_3 = \sum_{i=1}^{K} x_i$ and 2-link calls depart from the network at rate $\nu_4 = \sum_{l=1}^{\gamma} (w_l/\mu_2)$.

The rate of **null events** in the network is denoted by ν_5, where

$$(6.1) \qquad \nu_5 = \sum_{i=1}^{K} \frac{C_i}{\overline{\mu}} - \nu_3 - \nu_4,$$

where $\overline{\mu} = \min\{\mu_1, \mu_2\}$.

The **Total Rate of events** in the network is denoted by *Rate*, where

$$(6.2) \qquad Rate = \sum_{i=1}^{5} \nu_i = \sum_{i=1}^{K} \lambda_{1i} + \sum_{l=1}^{\gamma} \frac{\lambda_{2l}}{(K-1)} + \sum_{i=1}^{K} \frac{C_i}{\overline{\mu}}.$$

The transitions in the network occur at rate (6.2) and the correction between steps in the *optimality equation* and 'time' in the continuous time process for the discount factor α is

$$\overline{\alpha} = \frac{Rate}{Rate + (-\ln \alpha)}.$$

In the examples considered we used the approximation $\overline{\alpha} = \alpha^{1/Rate}$. The correction as well as the effect that our approximation has in calculating TEDR values and optimal policies is discussed in §2.3.2; see remark on page 50.

The notation is that of Chapter 2, and the transition probabilities for the network are as follows:

$$P_\pi(x_i + e_i, w | x, w) = \begin{cases} 0, & \text{if } \pi(x, w) = \text{reject} \\ \frac{\lambda_{1i}}{Rate}, & \text{otherwise} \end{cases}$$

$$P_\pi(x_i - e_i, w | x, w) = \frac{x_i / \mu_1}{Rate}, \quad if \quad x_i \geq 0$$

$$P_\pi(x, w_l + e_l | x, w) = \begin{cases} 0, & \text{if } \pi(x, w) \text{ reject} \\ \frac{\lambda_{2l}}{Rate(K-1)}, & \text{otherwise} \end{cases}$$

$$P_\pi(x, w_l - e_l | x, w) = \frac{w_l / \mu_2}{Rate}, \quad if \quad w_l \geq 0.$$

$$P_\pi(x, w | x, w) = \frac{\nu_5}{Rate}$$

As in earlier chapters, one link calls are accepted whenever there is room for them. Hence

$$P_\pi(x_i + e_i, w | x, w) = \begin{cases} 0, & \text{if } x_i >= C \\ \frac{\lambda_{1i}}{Rate}, & \text{otherwise} \end{cases}$$

The average reward for the *Asymmetric* network we consider is denoted by Ψ, where

(6.3) $$\Psi = \frac{1}{Rate}[R1 \sum_{i=1}^{K} \lambda_{1i} I(e_i | x, w) + \frac{R2}{K-1} \sum_{l=1}^{\gamma} \lambda_{2l} I(e_l | x, w)]$$

where $I(e_l | x, w) = 1$ or 0 according as a call on route l is accepted or rejected by the admission policy.

6.4 Computing and Results

We present the results in groups of three categories:

(a) The size of the network i.e. (K, C);

(b) For a specific size of a network we keep $L = \lambda_1 + \lambda_2/2$ fixed[2]; we have focused on examples for networks where L is 'critical' for the link i.e. nearly full, full, overloaded etc;

(c) For a specific L we consider various arrival rates.

The tables suggest that the *Admission Price* policies are a good approximation for the optimal policy under various considerations for the *Asymmetric* networks.

In the following Tables and figures $n1 = \nu_1 = \lambda_1$ and $n2 = \nu_2 = \lambda_2$. $R_1 = 2, R_2 = 1$. All the other quantities have been defined in §2. μ_2 will be 1 unless specified differently.

Table 6.1: presents some results for *Asymmetric* networks with $(K, C) = (3, 3)$ in which the arrival rate of single link traffic on link Z, λ_{13} is different than that of links X and Y λ_{11} and λ_{12}. The results suggest that the *Admission Price* policy is a good approximation for the optimal policy.

Tables 6.2, 6.3 and 6.4: present results for *Asymmetric* networks with $K = 3$ and capacities $C = (3, 3, 4), (3, 3, 5)$ and $(3, 3, 6)$. The results suggest that the *Admission Price* policies are a good approximation of the optimal policy. The properties of the optimal policies in the latter networks are analysed in §6.5.

The *Admission Price* policy described by W_i is calculated by our usual method from the optimal value function under the optimal (but difficult to describe) policy, and for all the states $z = (x, w)$ that could possibly accept an increase in their number of 1-link calls. For such states we calculate the differences $V(z + e_i) - V(z)$. Then we look at those differences for a collection of states $z_a, z_b, .. z_m$ with the same number of free links F_i. For 'good' W_i $V(z + e_i) - V(z) \approx W_{F_i}$ must be satisfied. The choice for such W_{F_i} is taken from within the range of the above differences; usually

---

[2]See §2.3.2.

their mean value; see also Chapter 4.

Table 6.5: presents the case of a $(K, C) = (3, 3)$ network and $\mu_2 = 0.8$. Compare Table 6.5 with Table 4.2.

Table 6.6: demonstrates the performance of the *Admission Price* policies for a network with $(k, C) = (3, 3)$ and μ_2 increasing. See also discussion of the optimal policy in this case in §6.6.

The behaviour of the Ω policy is the same as described in §4.2.

Note that by comparing Figures 6.3 with 6.5, and 6.4 with 6.5, it is clear that the W_i do not change much as the capacity varies for the 3rd link.

For an explanation of why α is not exactly 0.8 see the remark on page 50.

$$\boxed{K =° 3, \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$$

Single link	2-link call	Capacities	Optimal	w-Optimal
$\lambda_{11,2,3}$	$\lambda_{21,2,3}$	$C_{1,2,3}$	$V(0)$	$V_W(0)$
0.5 0.5 0.5	4.9	3,3,3	22.9963	22.8750
0.5 0.5 1.16	4.9	3,3,3	26.2778	25.9658
0.5 0.5 1.55	4.9	3,3,4	27.1022	26.7767
0.5 0.5 1.93	4.9	3,3,5	32.0108	31.7014
1.3 1.3 1.3	3.3	3,3,3	35.0391	35.0389
1.3 1.3 1.76	3.3	3,3,3	37.3559	37.3401
1.3 1.3 2.35	3.3	3,3,4	38.6687	38.6534
1.3 1.3 2.93	3.3	3,3,5	44.9085	44.8831
1.8 1.8 1.8	2.3	3,3,3	42.5410	42.5230
1.8 1.8 2.13	2.3	3,3,3	44.1330	44.1281
1.8 1.8 2.85	2.3	3,3,4	45.6073	45.6013
1.8 1.8 3.56	2.3	3,3,5	52.6654	52.5794

Table 6.1: Networks with λ_{13}/C_3 constant

$$K = 3, C = (3,3,4), \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_\alpha(0)$	$V_W(0)$
0.5	4.9	2.95	21.7478	21.5569
0.7	4.5	2.95	24.4493	24.4200
1	3.9	2.95	28.7907	28.6471
1.3	3.3	2.95	33.2670	33.2153
1.5	2.9	2.95	36.2549	36.2425
1.8	2.3	2.95	40.5929	40.5510
0.5	5	3	21.7947	21.6026
0.7	4.6	3	24.4905	24.4585
1	4	3	28.8245	28.6814
1.3	3.4	3	33.2969	33.2416
1.5	3	3	36.2803	36.2668
2	2	3	43.3637	43.3324
0.3	5.6	3.1	19.3951	18.7791
0.6	5	3.1	23.1699	23.1183
1	4.2	3.1	28.8884	28.7468
1.4	3.4	3.1	34.8465	34.8284
1.7	2.8	3.1	39.2294	39.1944
2	2.2	3.1	43.3835	43.3508
0.5	6	3.5	22.1846	21.9895
0.7	5.6	3.5	24.8346	24.7803
1	5	3.5	29.1064	28.9714
1.3	4.4	3.5	33.5472	33.4598
1.6	3.8	3.5	37.9402	37.9026
2	3	3.5	43.4504	43.4114
0.5	7	4	22.4683	22.2815
0.7	6.6	4	25.0941	25.0173
1	6	4	29.3137	29.1905
1.3	5.4	4	33.7369	33.6980
1.6	4.8	4	38.0797	38.0293
2	4	4	43.5148	43.4670

Table 6.2: Networks with $C=$ (3,3,4)

$$K = 3, C = (3, 3, 5), \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_\alpha(0)$	$V_W(0)$
0.5	4.9	2.95	23.4263	23.1659
0.7	4.5	2.95	26.1843	26.0638
1	3.9	2.95	30.6177	30.3458
1.3	3.3	2.95	35.0779	34.9928
1.5	2.9	2.95	38.1238	38.0030
1.8	2.3	2.95	42.6614	42.5814
0.5	5	3	23.4937	23.2281
0.7	4.6	3	26.2421	26.1185
1	4	3	30.6675	30.3938
1.3	3.4	3	35.1199	35.0321
1.5	3	3	38.1620	38.0351
2	2	3	45.6225	45.5476
0.3	5.6	3.1	20.9844	20.4359
0.6	5	3.1	24.9770	24.7892
1	4.2	3.1	30.7624	30.4853
1.4	3.4	3.1	36.7212	36.6101
1.7	2.8	3.1	41.2599	41.1888
2	2.2	3.1	45.6697	45.5868
0.5	6	3.5	24.0721	23.7614
0.7	5.6	3.5	26.7439	26.5818
1	5	3.5	31.0946	30.8011
1.3	4.4	3.5	35.4835	35.3614
1.6	3.8	3.5	39.9966	39.9149
2	3	3.5	45.8273	45.7388
0.5	7	4	24.5118	24.1714
0.7	6.6	4	27.1359	26.9317
1	6	4	31.4202	31.1111
1.3	5.4	4	35.7698	35.6060
1.6	4.8	4	40.2426	40.1235
2	4	4	45.9737	45.8746

Table 6.3: Networks with $C =$(3,3,5)

$$K = 3, C = (3, 3, 6), \mu_2 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_\alpha(0)$	$V_W(0)$
0.5	4.9	2.95	26.6353	26.3354
0.7	4.5	2.95	29.5568	29.2722
1	3.9	2.95	34.0770	33.8368
1.3	3.3	2.95	38.6492	38.4930
1.5	2.9	2.95	41.7328	41.5717
1.8	2.3	2.95	46.4414	46.3152
0.5	5	3	26.7454	26.4310
0.7	4.6	3	29.6548	29.3596
1	4	3	34.1656	33.9162
1.3	3.4	3	38.7238	38.5622
1.5	3	3	41.8033	41.6327
2	2	3	49.6603	49.4969
0.3	5.6	3.1	24.1087	23.6947
0.6	5	3.1	28.3947	28.0711
1	4.2	3.1	34.3353	34.0680
1.4	3.4	3.1	40.3935	40.2238
1.7	2.8	3.1	45.0549	44.9286
2	2.2	3.1	49.7727	49.5881
0.5	6	3.5	27.7135	27.2647
0.7	5.6	3.5	30.5147	30.1165
1	5	3.5	34.9227	34.5953
1.3	4.4	3.5	39.3668	39.1458
1.6	3.8	3.5	43.9445	43.7734
2	3	3.5	50.1623	49.9739
0.5	7	4	28.4955	27.9252
0.7	6.6	4	31.1840	30.5453
1	6	4	35.5100	35.1160
1.3	5.4	4	39.8659	39.5831
1.6	4.8	4	44.4079	44.1543
2	4	4	50.5519	50.2915

Table 6.4: Networks with $C = (\mathbf{3,3,6})$.

$$K = 3, C = 3, \mu_2 = 0.8, R_1 = 2, R_2 = 1, \alpha = 0.8$$

ν_1	ν_2	$\nu_1 + \nu_2/2$	$V_\alpha(0)$	$V_W(0)$
0.5	4.9	2.95	25.2701	24.8526
0.7	4.5	2.95	27.7956	26.6446
1	3.9	2.95	31.9165	31.4875
1.3	3.3	2.95	36.0950	35.9199
1.5	2.9	2.95	38.9657	38.9222
1.8	2.3	2.95	43.1925	42.9558
0.5	5	3	25.3446	24.9222
0.7	4.6	3	27.8563	26.7031
1	4	3	31.9696	31.5410
1.3	3.4	3	36.1393	35.9675
1.5	3	3	39.0066	38.9646
2	2	3	45.9144	45.8077
0.3	5.6	3.1	23.0098	22.1278
0.6	5	3.1	26.7275	26.4895
1	4.2	3.1	32.0701	31.6431
1.4	3.4	3.1	37.6502	37.5679
1.7	2.8	3.1	41.9053	41.5665
2	2.2	3.1	45.9637	45.8519
0.5	6	3.5	25.9787	25.5174
0.7	5.6	3.5	28.3669	27.2069
1	5	3.5	32.4101	31.6901
1.3	4.4	3.5	36.5127	36.3723
1.6	3.8	3.5	40.7551	40.2608
2	3	3.5	46.1310	46.0013
0.5	7	4	26.4589	24.2533
0.7	6.6	4	28.7510	27.5994
1	6	4	32.7484	32.0485
1.3	5.4	4	36.8005	35.7422
1.6	4.8	4	40.9983	40.4603
2	4	4	46.2899	46.1428

Table 6.5: Networks with $(K, C) = (3, 3)$ and $\mu_2 = 0.8$.

$$\boxed{K = 3, C = 3, \mu_1 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8}$$

μ_2	$V_\alpha(0)$	$V_W(0)$
0.5	30.4205	25.5860
0.7	26.6804	24.1823
0.8	25.2701	24.8526
0.9	24.0541	23.7876
1.1	22.0883	22.0773
1.3	20.7773	20.3181
1.5	19.8160	19.0632
1.7	19.1032	18.0421
1.9	18.5503	17.1950
2	18.3136	16.8235

Table 6.6: Networks with various μ_2. for $\nu_1 = 0.5; \nu_2 = 4.9$.

6.5 Optimal Policy and Properties

For *Asymmetric* networks presented in §6.1, we consider policies that always accept 1-link calls when there is room to fit them in. It is the 2-link calls we seek to restrict (in order to maximise the TEDR) and hence a policy is a set of $|\mathcal{S}| \times \frac{1}{2}K(K-1)$–tuples of Boolean variables. 477 triples when $K = 3, C = (3,3,4)$. As the networks increase in size the optimal policy increases rapidly. In the following examples, we describe the exact optimal policy for some networks with:

(a) $K = 3$ and various $C = (3,3,C_3)$;

(b) $(K,C) = (3,3)$, $(\lambda_1 = 0.5, \lambda_2 = 4.9)$ and various μ_2;

(c) $(K,C) = (3,3)$, $\mu_2 = 0.8$ (it was 1) and various arrival rates;

only to show the complexity and difficulty that both arise in trying to conclude about a network's behaviour and performance by looking at the calculated optimal policy. In the following description we do not include the cases in which the optimal policy accepts the 2-link calls when there is room to fit them in, nor the cases in which the network is full.

132

Let $F = (F_1, F_2, F_3)$ denote the number of free circuits on links X, Y and Z respectively. A network state is represented by a 6–tuple (X,XY,XZ,Y,YZ,Z). For example (010011) describes the state of the network with: one 2-link call on pair XY, one 2-link call on pair YZ, and a single-link call on Z with coresponding free circuits $F = (2,1,1)$. Many states have the same number of free circuits.

The quantities $\lambda_1, \lambda_2, R_1, R_2,$ and α are all defined in §2.1, §2.2 and §2.3. If not specified $\mu_1 = 1, \mu_2 = 1$.

For the description of the optimal policy we proceed the same way as in the analysis of §2.

6.5.1 What Happens as C_3 increases

Example 6.1

$K = 3, \lambda_1 = 0.5, \lambda_2 = 4.9, R_1 = 2, R_2 = 1, \alpha = 0.8$

I: Capacities C=(3,3,4). States=477

The optimal policy rejects 2-link calls when there is room to fit them in. Specifically in this network the optimal policy rejects:

(a) 2-link calls on XY in 49 states with F= (1,1,4), (1,1,3), (1,1,2), (1,1,1), (1,1,0), (2,1,3), (2,1,4), (1,2,3) and (1,2,4); XZ and YZ calls are accepted in the previous cases.

(b) 2-link calls on XZ for F= (1,3,1), (1,2,1); XY and YZ are accepted in the previous cases.

(c) 2-link calls on YZ for F= (3,1,1), (2,1,1); XY and XZ calls are accepted in the previous cases.

Compare with Example 3.2. Note that changes in the capacity of link Z have an effect on the acccentance/rejection of XY calls i.e. the optimal policy is not of an admission price form as it depends upon the state-space and not just the two link involved.

133

II: Capacities C=(3,3,5). States=622

The optimal policy rejects 2-link calls when there is room to fit them in. Specifically in this network the optimal policy rejects:

(a) 2-link calls on XY in 94 states with $F=$ (1,1,5), (1,1,4), (1,1.3), (1,1,2), (1,1,1), (1,1,0), (2,1,5), (2,1,4), (2,1,3), (2,1,2), (1,2,5), (1,2,4), (1,2,3), (1,2,2); XZ and YZ calls are accepted in the previous cases.

(b) 2-link calls on XZ for $F=$ (1,3,1), (1,2,1); XY and YZ are accepted in the previous cases.

(c) 2-link calls on YZ for $F=$ (3,1,1), (2,1,1); XY and XZ calls are accepted in the previous cases.

Example 6.2

$K = 3, \lambda_1 = 1.8, \lambda_2 = 2.3, R_1 = 2, R_2 = 1, \alpha = 0.8$

I: Capacities C=(3,3,4). States=477

The optimal policy rejects 2-link calls when there is room to fit them in. Specifically in this network the optimal policy rejects:

(a) All 2-link calls in states with $F=$ (1,3,1), (3,1,1), (1,1,3), (2,2,1), (1,1,4), (1,1,1), (2,1,1), (1,1,2), (1,2,1) and (2,2,4).

(b) XY and XZ calls in states with $F=$ (1,3.4), (1,3,3), (1,3,2) and (1,2,4), (1,2,3) and (1,2,2).

(c) XZ and YZ calls in states with $F=$ (2,3,1), (3,2,1) and (3,3,1).

(d) XY and YZ calls in states with $F=$ (2,1,4), (2,1,3), (2,1,2) and (3,1,4), (3,1,3) and (3,1,2).

(e) XY calls in states with $F=$ (2,2,4), (2,2,3), (2,2,2), (2,2,0), (2,1,0), (3,1,0), (1,3,0), (1,1,0) and (1,2,0).

(f) YZ calls in states with $F=$ (0,3,1), (0,2,1), (0,1,4), (0,1,3), (0,1,2) and (0,1,1).

(g) XZ calls in states with $F=$ (3,0,1), (2,0,1), (1,0,1), (1,0,2) (1,0,3) and (1,0,4).

134

II: Capacities C=(3,3,5). States=622

The optimal policy rejects 2-link calls when there is room to fit them in. Specifically in this network the optimal policy rejects:

(a) All 2-link calls in states with $F=$ (3,1,1), (1,3,1), (2,2,1), (1,2,1), (2,1,1), (1,1,5), (1,1,4), (1,1,3), (1,1,2) and (1,1,1).

(b) XY and XZ calls in states with $F=$ (1,3,5), (1,3.4), (1,3,3), (1,3,2), (1,2,5), (1,2,4), (1,2,3) and (1,2,2).

(c) XZ and YZ calls in states with $F=$ (2,3,1), (3,2,1) and (3,3,1).

(d) XY and YZ calls in states with $F=$ (2,1,5), (2,1,4), (2,1,3), (2,1,2) and (3,1,5), (3,1,4), (3,1,3) and (3,1,2).

(e) XY calls in states with $F=$ (2,2,5), (2,2,4), (2,2,3), (2,2,2), (2,2,0), (2,1,0), (1,2,0), (2,3,5), (2,3,4), (3,1,0), (1,3,0), (3,2,4) (3,2,5) and (1,1,0).

(f) YZ calls in states with $F=$ (0,2,1), (0,3,1), (0,1,5), (0,1,4), (0,1,3), (0,1,2) and (0,1,1).

(g) XZ calls in states with $F=$ (3,0,1), (2,0,1), (1,0,1), (1,0,2) (1,0,3), (1,0,4) and (1,0,5).

6.5.2 What Happens as μ_2 increases

In the following examples we investigate what happens as μ_2 increases. As μ_2 increases the optimal value function (TEDR) V decreases as less valuable calls are using up more network resources. In the examples we describe the optimal policy for such cases ans see that the optimal policy rejects more and more 2-link traffic as μ_2 increases.

In the following cases $R_1 > (R_2/\mu_2)$.

| Example   6.3 |

$K = 3, C = 3, \lambda_1 = 0.5, \lambda_2 = 4.9, \mu_1 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$

The optimal policy rejects:

I: $\mu_2 = 0.5$. 2-link calls on XY in 1 state with $F=$ (1,1,3) but not in (1,1,2), (1,1,1), (1,1,0); XZ and YZ calls are accepted in (1,1,3). $V(0) = 30.42962$.

II: $\mu_2 = 0.8$. 2-link calls on XY in 10 states with $F=$ (1,1,3) and (1,1,2) but not in (1,1,1), (1,1,0); XZ and YZ calls are accepted in (1,1,3) and (1,1,2) states. $V(0) = 25.2701$.

III: $\mu_2 = 0.9$. 2-link calls on XY in 10 states with $F=$ (1,1,3) and (1,1,2) but not in (1,1,1), (1,1,0); XZ and YZ calls are accepted in (1,1,3) and (1,1,2) states. $V(0) = 24.05442$.

IV: $\mu_2 = 1$. See Example 3.2 in Chapter 3. $V(0) = 22.9963$.

V: $\mu_2 = 1.1$. (a) XY, XZ and YZ calls in 11 states with (1,1,1) free circuits. (b) 2-link calls on XY in 11 states with $F=$ (1,1,3) and (1,1,2) but not in (1,1,0). $V(0) = 22.08844$.

VI: $\mu_2 = 1.7$. (a) XY, XZ and YZ calls in 11 states with (1,1,1) free circuits. (b) XY and XZ calls in 9 states with (1,2,2), (1,1,2) and (1,2,1) free circuits. (c) 2-link calls on XY in 21 states with $F=$ (1,1,3), (1,1,2), (1,1,0), (2,1,3) and (1,2,3). $V(0) = 19.0572$.

VII: $\mu_2 = 2$. (a) XY, XZ and YZ calls in 12 states with (1,1,1), (1,2,1), (1,1,2) and (2,1,1) free circuits. (b) XY and XZ calls in 9 states with (1,2,2), (1,1,2) and (1,2,1) free circuits. (c) 2-link calls on XY in 27 states with $F=$ (1,1,3), (1,1,2), (1,1,0), (2,1,3), (1,2,3), (2,2,3), (2,1,0) and (1,2,0). $V(0) = 18.3173$.

Example 6.4

$K = 3, C = 3, \lambda_1 = 1.8, \lambda_2 = 2.3, \mu_1 = 1, R_1 = 2, R_2 = 1, \alpha = 0.8$

The optimal policy rejects:

I: $\mu_2 = 0.5$. (a) XY, XZ and YZ calls in 11 states with (1,1,1) free circuits. (b) 2-link calls on XY in 11 states with $F=$ (1,1,3), (1,1,2) and (1,1,0). $V(0) = 45.1248$.

II: $\mu_2 = 0.8$. (a) XY, XZ and YZ calls in 41 states with (3,1,1), (1,3,1), (1,1,3), (1,1,1), (1,2,1), (1,1,2) and (2,1,1) free circuits. (b) XY and XZ calls in 8 states with

(1,2,2), (1,2,3) and (1,3,2) free circuits. (c) 2-link calls on XY in 34 states with $F=$ (3,1,0), (1,1,0), (2,1,0), (1,3,0), and (1,2,0). $V(0) = 43.1926$.

III: $\mu_2 = 0.9$. (a) XY, XZ and YZ calls in 41 states with (3,1,1), (1,3,1), (1,1,3), (1,1,1), (1,2,1), (1,1,2) and (2,1,1) free circuits. (b) XY and XZ calls in 8 states with (1,2,2), (1,2,3) and (1,3,2) free circuits. (c) 2-link calls on XY in 37 states with $F=$ (3,1,0), (1,1,0), (2,1,0), (1,3,0), (2,2,3) and (1,2,0). $V(0) = 42.8465$.

6.5.3 Properties

In 6 *Asymmetric* networks in which the number of link $K = 3$ and capacity varies as well as networks in which K, C is fixed but μ_2 increases the results suggest that the properties of the optimal policy are those of the *Symmetric* networks. In particular:

Property A: Dependency on the State-Space.

Property B: For calls which are disjoint, i and j say, and thus could be widely separated in a network, in general, the more type i calls in progress, the less likely we are to reject type j calls, and vice-versa; disjoint calls are for example XY and Z calls. This property does hold for the *Asymmetric* networks. See Example 6.1 in which XY rejection in states with (2,1,3) free circuits coincides with rejection in states with (2,1,4) free circuits.

Property C: Monotonicity. If an arrival for a 2-link call on route k is rejected in state $\bar{z}$, where $z \geq \bar{z}$, then it will also be rejected in states z. Our results suggest that this property does not hold in general; see Examples 6.1 in which rejection in states with (2,1,4) and (2,1,3) free circuits does notcoincide with rejection in states (2,1,2), (2,1,1) and (2,1,0).

Property D: If we reject a type i call in state z, then we reject it in state $z + k$ for calls i and k which are distinct and not disjoint; not disjoint calls are for example XY and YZ calls. This property does hold for the *Asymmetric* networks.

Property E: Weak Monotonicity If we reject a type k call in state z, then we reject a type call k in state $z + e_k$. This property does hold.
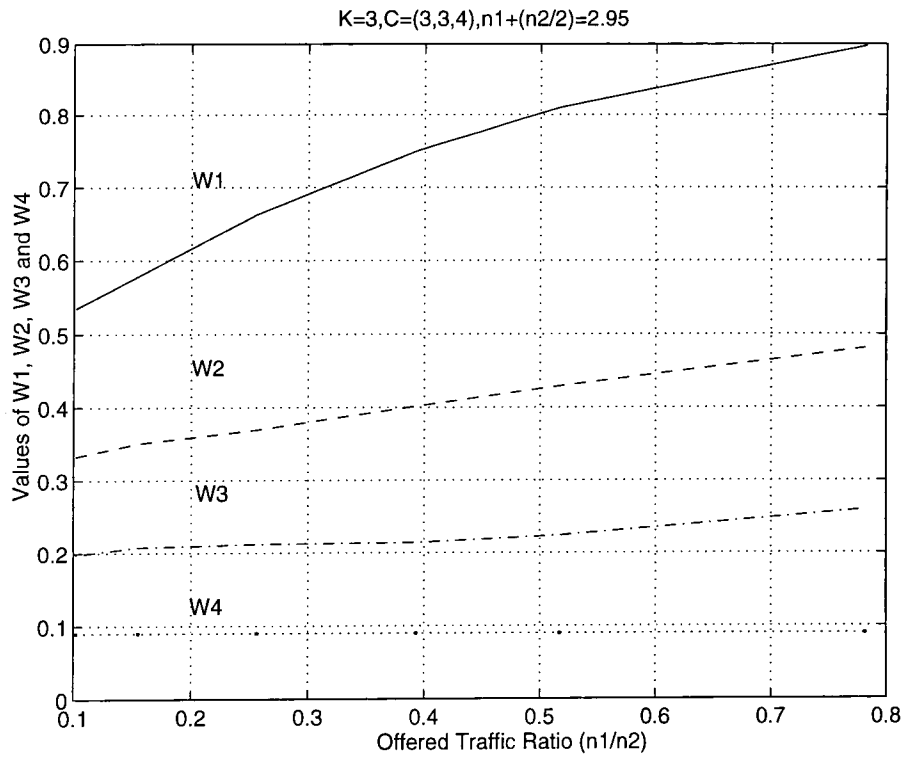
Figure 6.1: Change in w_i for $(K = 3, C = 3, 3, 4)$ as n1 increases; n1+(n2/2)=2.95.
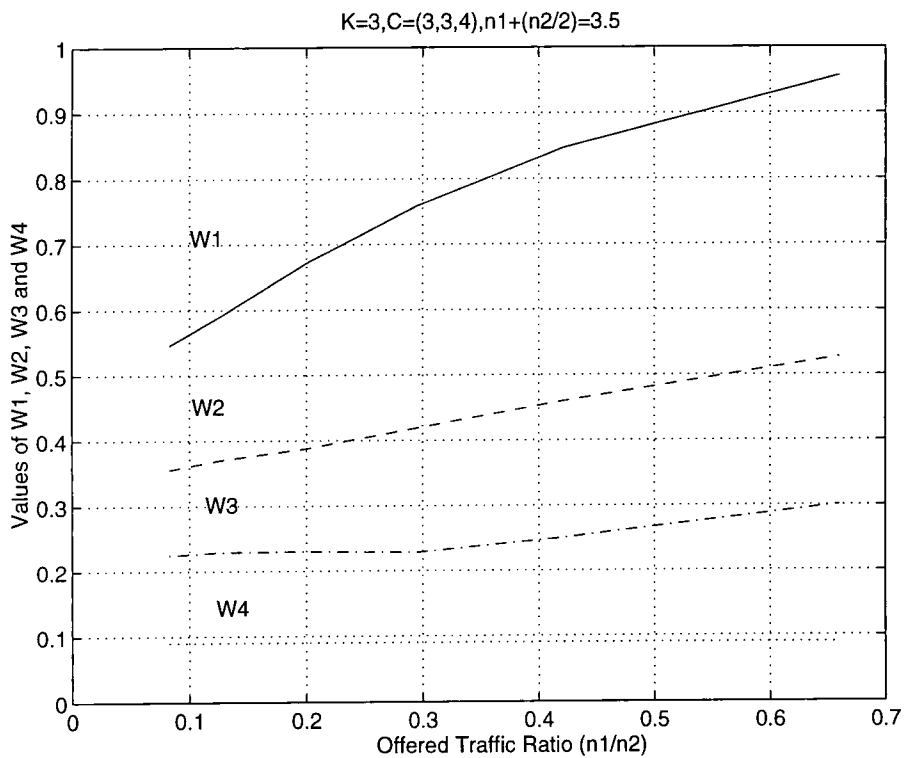
Figure 6.2: Change in w_i for $(K = 3, C = 3, 3, 4)$ as n1 increases; n1+(n2/2)=3.5.

138
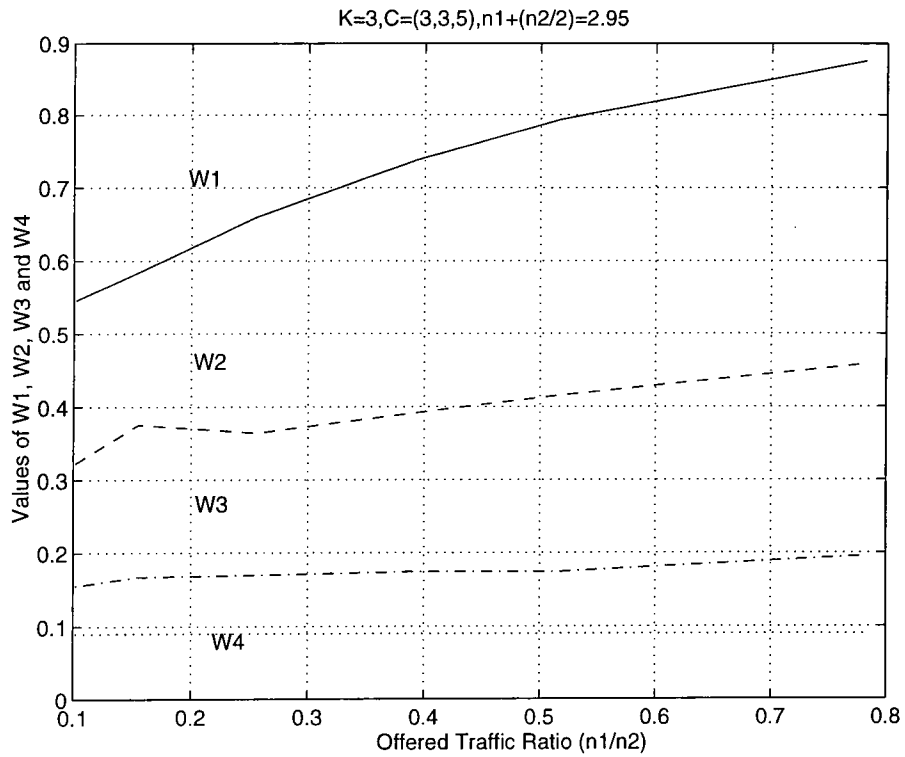
Figure 6.3: Change in w_i for $(K = 3, C = 3, 3, 5)$ as n1 increases; n1+(n2/2)=2.95.
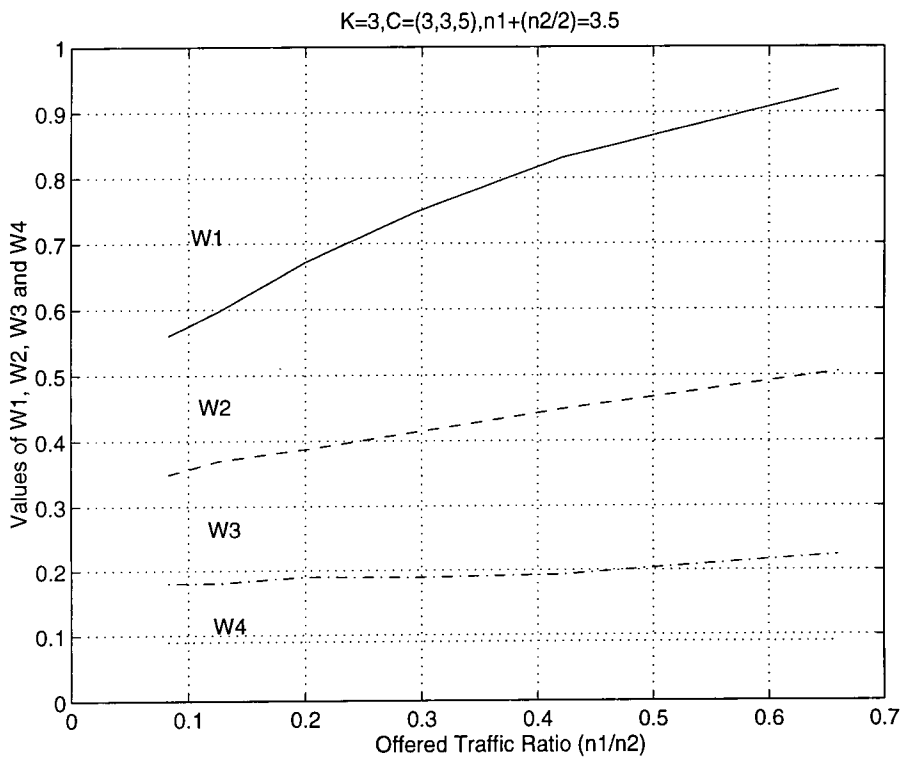
Figure 6.4: Change in w_i for $(K = 3, C = 3, 3, 5)$ as n1 increases; n1+(n2/2)=3.5.
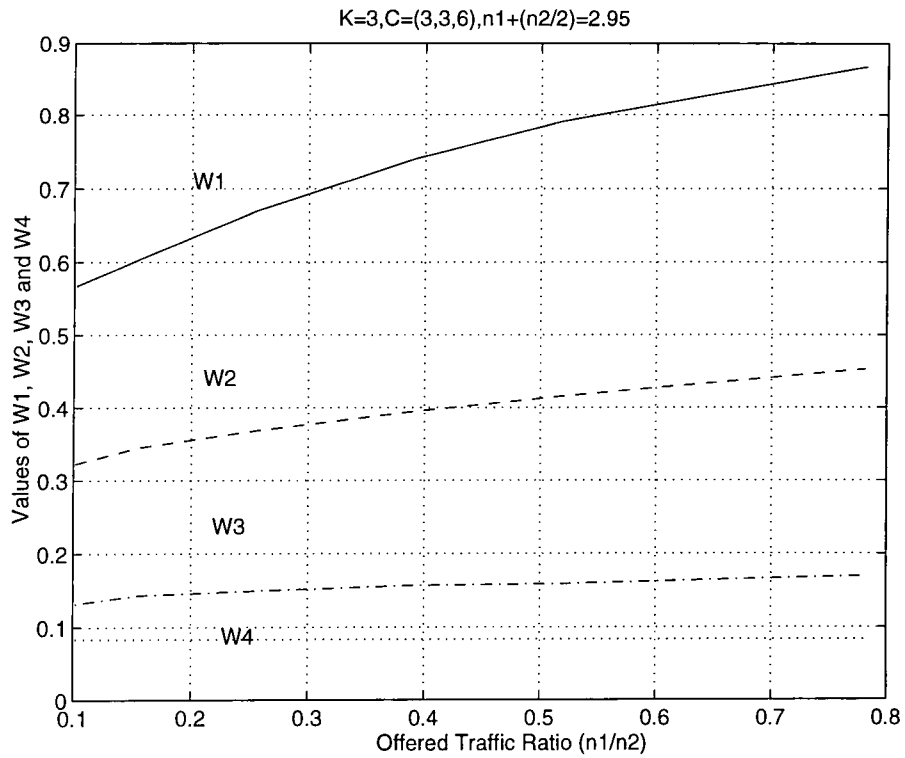
Figure 6.5: Change in w_i for $(K = 3, C = 3, 3, 6)$ as n1 increases; n1+(n2/2)=2.95.
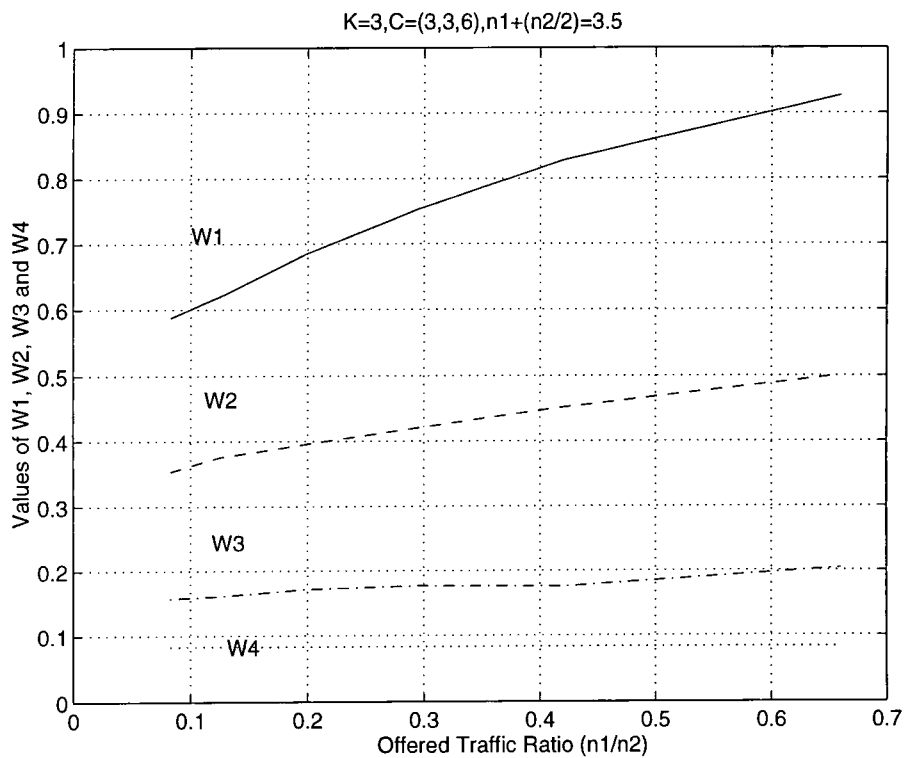
Figure 6.6: Change in w_i for $(K = 3, C = 3, 3, 6)$ as n1 increases; n1+(n2/2)=3.5.
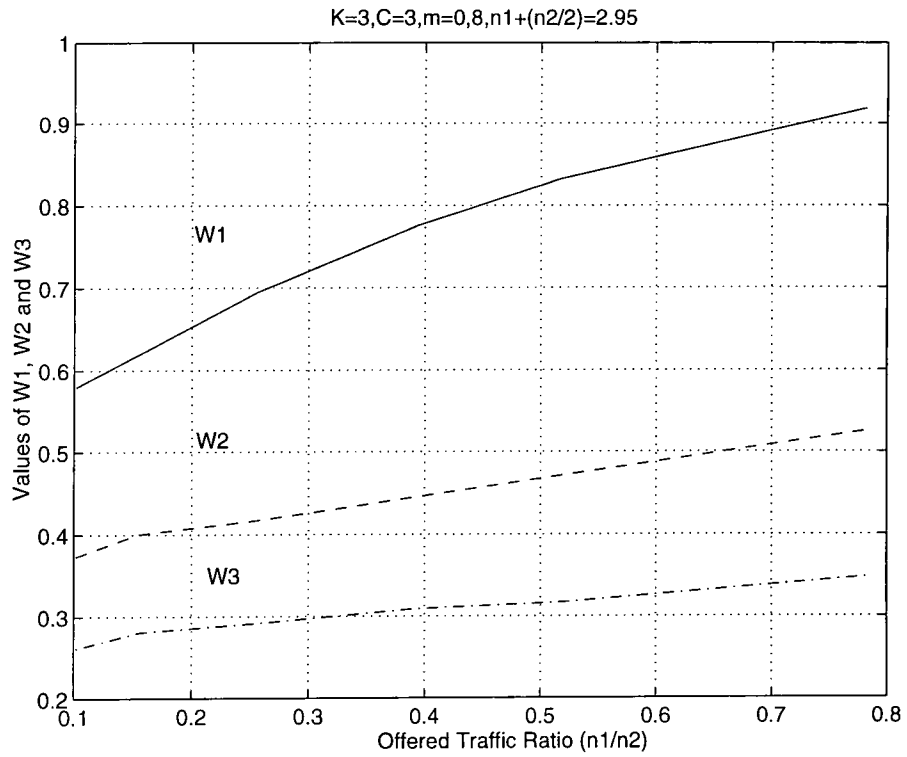
Figure 6.7: Change in w_i for $(K = 3, C = 3)$ and $\mu_2 = 0.8$ as n1 increases; n1+(n2/2)=2.95.
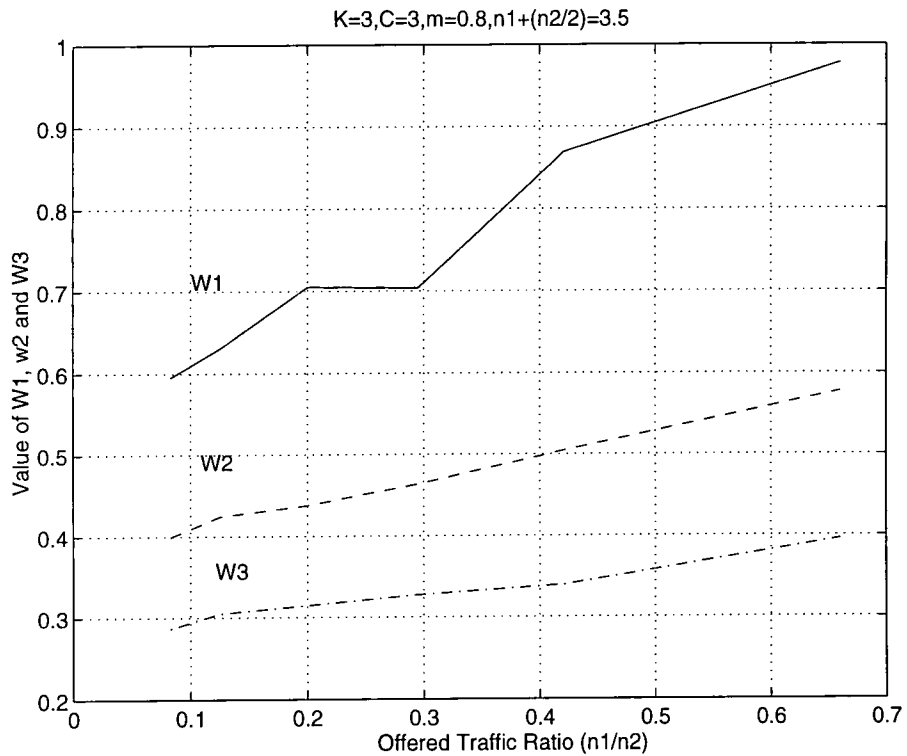
Figure 6.8: Change in w_i for $(K = 3, C = 3,)$ and $\mu_2 = 0.8$ as n1 increases; n1+(n2/2)=3.5.

6.6 The max-flow Bound for Asymmetric Networks

In chapter 2 we presented the *max-flow* bound for the *Symmetric* networks. Here we will just give an expression of the linear programming problem whose solution gives an upper bound on the performance of *Asymmetric* networks, and present a few results for networks with $K = 3$ and $C = (C_1, C_1, C_2)$.

For the general formula we will consider that the arrival rates for single link and 2-link traffic are $(\lambda_1, \lambda_1, \lambda_1)$ and $(\lambda_2, \lambda_2, \lambda_2)$ respectively.

The relevant linear programming problem[3] is

LP1:

$$\max \left[\sum_1^K R_1 x_i + \sum_1^l R_2 w_l \right]$$

$$x_i \leq \lambda_1, \ i = 1, .., K; \qquad (K-1)w_l \leq \lambda_2, \ l \in A_i$$

$$x_1 + w_1 + w_2 \leq C_1, \ x_2 + w_1 + w_3 \leq C_1 \text{ and } x_3 + w_1 + w_3 \leq C_2, x_i, w_l \geq 0.$$

---

[3]See also §2.8

Chapter 7

Conclusions

In this thesis *Symmetric Star* Loss circuit-switched networks which consist of K links of the same capacity C linked through a common node were considered. There are 1-link routes and 2-link routes involving any pair of the single links on which calls can request admission. Arrivals form independent Poisson streams on each route and the routing is fixed. Dependency arises through occupancy of pairs of circuits. Both types of calls have the same exponential holding time. For different types of calls we earn different rewards with the single link calls generating bigger rewards. The rewards are discounted at a fixed rate.

The operation of the networks is viewed as a Markov Decision Process. In the networks we investigated *stationary* policies which accept or reject traffic requests in order to maximise the Total Expected Discounted Reward (TEDR). We solved the *optimality equation* numerically for a range of small examples by using the Policy Improvement iterative algorithm of Dynamic Programming.

Reduced state-space networks were also considered in which a call on a 2-link route, once accepted, is split into two independent calls on the links involved. This greatly reduces the size of the state-space. Finally we looked at *Asymmetric Star* networks with different number of circuits per link and different exponential holding times.

7.1 Optimal Policies and their Properties

Generally, optimal policies are complex to describe and 'nobody' has exact solutions except for problems on a single link. Our networks are in some respect the simplest networks which aren't a single link or two in series but even so only very small examples can be treated numerically. For such examples we showed how complex the optimal policies are and found evidence that suggest that the following properties - except *Monotonicity* - hold for all the models of networks considered i.e. the full state-space, reduced state-space and *Asymmetric* networks:

(a) *Dependency on the State-Space* The optimal policy depends upon the full state space $\mathcal{S}$ and not just the free circuit configurations.

(b) *Non-Locality and Disjointness* If we reject type j calls in $z + e_i$ then we reject them in z for call types i and j which are disjoint. Property B means that for calls which are disjoint, and thus could be widely separated in a network, in general, the more type i calls in progress, the less likely we are to reject type j calls, and vice-versa.

(c) *Not Disjointness* If we reject a type i call in state z, then we reject it in state $z + e_k$ for calls i and k which are distinct and not disjoint.

(d) *Weak Monotonicity* In which type k calls are monotonic with respect to themselves, that is we assume that for all call types k, if we reject a type k call in state z, then we reject a type call k in state $z + e_k$.

The property of *Monotonicity* in which if an arrival for a 2-link call on route k is rejected in state $\overline{z}$, then it will also be rejected in states z, where $z \geq \overline{z}$, does not hold as our examples suggest.

7.2 Admission Price Policies Ω

These policies are not optimal but they are believed to be asymptotically optimal for large networks. In this thesis we investigated if such policies are any good for small networks. Our results suggest that they are very good for both the full and

reduced state-space networks; this is also true for some examples of *Asymmetric* networks.

Our Ω policies are not robust in that they change much as the offered traffics change. The costs they assume get bigger as λ_1 increases; this increase translates to more acceptance of single link traffic which returns a bigger reward than the 2-link one. The policies reflect clearly the fact that spare capacity has potential worth as we can use it to carry future calls and that units of spare capacity become more valuable as the system fills up.

Although not robust, our *Admission Price* policies have the following very interesting properties:

(a) they do not change much as K increases. This is of practical importance as it suggests that the Ω calculated from small networks could be used as a good approximate for the optimal policy in larger networks. For example, we found that by applying the Ω policy derived from networks $(K, C) = (3, 3)$ to $(K, C) = (4, 3)$ and $(K, C) = (5, 3)$ the results are excellent; and

(b) Ω policies of the reduced state-space model is a very good approximation for the optimal policy of the full model and vice-versa; in both cases very little improvement is left to be performed by applying them and in some cases none.

Our *Admission Price* policies were also compared to similar ones suggested in routing schemes proposed by Ott & Krishnan (1985, 1986) and Key (1990) for an example with $(K = 3, C = 3)$.

7.3 Other Results

In this work, as mentioned earlier, we looked for optimal policies in time where the state of the network can be described by the state of all current routes. Thus, the optimal state-dependent routing in our case is a problem of optimal control of a Markov Decision Process in a huge state-space. Because of the size of the state-space, numerical calculations are very difficult and any policy which is given in the form of routing decisions is unimplementable. We comment on the size of the state-

space and we discuss the difficulties arising in trying to calculate the state-space for the *Symmetric* networks considered in this thesis.

The Value Detemination step in Policy Improvement can be solved by either using iterative techniques (G-S, SOR) or Gauss Elimination. The actual convergence of these schemes as well as arguments on the error analysis in relation to the numerical evaluation of the TEDR for small *Star Symmetric* networks are examined which suggest that the the Gauss Elimination solution is more accurate.

Unfortunately, at the time of writing this thesis, the solution of $\mathbf{AV} = \mathbf{R}$ as well as the calculation of the condition number were possible for small examples of networks only. This is due to the difficulty posed by the huge state-space in calculating and storing $\mathbf{A}$ which is $|\mathcal{S}|^2$ in size. Nevertheless, in our examples with $(K = 2, C)$ and $(K = 3, C)$, where $C \leq 4$, the Gauss Elimination solution $\mathbf{V} = \mathbf{A}^{-1}\mathbf{R}$ agrees with the SOR solution to the expected precision (i.e. $\approx 10^{-6}$). That confirms the accuracy of the SOR algorithm.

In our work we have also tried different over-relaxation (SOR) parameters and the optimal results in convergence speed (the total number of iterations needed) suggest that the optimal parameters are different for problems of different size. As we show, it seems that the variation is associated more with the change in offered traffic rates than the network size.

Appendix A

The Size of the State-space

The exact size of the state space for our *Star* network with K links of capacity C and both 1 and 2-link calls is not easy to find. For any given values of K and C it can be determined with a recursive counting routine carried out on a computer[1] but a general asymptotic of a useful nature seems too difficult to find.

To see why this is consider the following counting scheme: for each possible number of 2-link calls in the network and for each possible arrangement of these calls count the number of different arrangements of 1-link calls using the free circuits remaining and then sum. Let $N(K, C)$ denote the size of the state space.

For $K=2$ we find

$$N(2,C) = \sum_{j=0}^{C} 1 \ (C+1-j)^2 = \sum_{j=1}^{C+1} j^2 = \frac{(C+1)(C+2)(2C+3)}{6}$$

since there is only one way to allocate j 2-link calls on this network.

For $K=3$ the network can carry up to $[3C/2]$ 2-link calls and hence

$$N(3,C) = \sum_{j=0}^{[3C/2]} \sum_{j_1+j_2+j_3=j} \prod_{i=1}^{3}(C+1+j_i-j)$$

[1]See Appendix B.1

where the second summation is over all distinct allocations of 2-link calls to the $K = 3$ links that satisfy $j_i + j_k \leq C$ and the product is the number of arrangements of 1-link calls in the free capacity left after the allocation of the 2-link calls.

We can describe the general case similarly. Index the $K(K-1)/2$ distinct 2-link call types by l and introduce mappings $1(*)$ and $2(*)$ to idenitfy from which links a specific 2-link call requires circuits.

Let $P(j)$ be the set of arithmetic partitions of j into at most $K(K-1)/2$ non-zero parts which satisfy all the constraints

$$\sum_{l \in L(i)} j_l \leq C, \quad i = 1, 2, .., K$$

where $L(i) = \{l : 1(l) = i \text{ or } 2(l) = i\}$ i.e. the collection of 2-link call types that require a circuit from link i. With this notation we can see that

$$N(K, C) = \sum_{j=0}^{[KC/2]} \sum_{P(j)} \prod_{l=1}^{K(K-1)/2} (C + 1 - j_{1(l)} - j_{2(l)})$$

and this shows the source of the difficulty very clearly.

The problem of counting arithmetic partitions of integers is an old one and has been much studied (see Hardy and Wright [1960]) but no good asymptotic results have been found for sets similar but simpler than $P(j)$.

An approximation to $N(K, C)$ for small K can be found by finding the volume of an appropriate set in $R^{K(K-1)/2}$. We will consider the case $K = 3$. The discrete state space is 6 dimensional so let x_1, x_2, x_3 correspond to the variables counting 1-link calls and y_1, y_2, y_3 correspond to the variables counting 2-link calls. We require $0 \leq x_i, y_i \leq C$, and

$$x_i + \sum_{L(i)} y_l \leq C, \quad i = 1, 2, 3.$$

For numbers a, b let $a \wedge b = \min(a, b)$.

The volume of the polyhedral region satisfying these constraints is

$$
\begin{aligned}
A_3 &= \int_{\underline{y}} \int_{\underline{x}} \prod_l (C - y_{1(l)} - y_{2(l)}) \, d\underline{x} \, d\underline{y} \\
&= \int \int \int_0^{(C-y1)\wedge(C-y2)} (C - y_1 - y_2)(C - y_1 - y_3)(C - y_2 - y_3) \, dy_3 dy_2 dy_1 \\
&= \ldots\ldots = k_3 \, C^6
\end{aligned}
$$

for some constant k_3.

In general it seems that when there are K links we will get

$$
A_K = k_K \, C^{K(K+1)/2}.
$$

$N(K, C)$ counts the number of lattice points in a region with volume A_K so as C becomes large we expect that for fixed K

$$
N(K, C) \approx k_K \, (C + 1)^{K(K+1)/2}
$$

where the constants k_K decreases as K grows.

The numerical calculations of $N(K, C)$ for small C when $K = 3, 4, 5$ accord reasonably well with this approximation.

K	$C = 4$	$C = 5$	$C = 6$	$C = 7$	estimated k_K
3	1023	2610	5860	11942	$4 \ 10^{-2}$
4	28746	124074	442918	1366806	$2 \ 10^{-3}$
5	1131389	8940840	54653970	273816800	$4 \ 10^{-6}$

Estimating k_K from $N(K, C)$.

From this we see that for $K = 6$ (and so $K(K + 1)/2 = 21$) the number of states grows as $(C + 1)^{21}$ and for larger K the rate of growth is even more rapid.

Appendix B

Computing

The following programs are written in Unix Berkeley Pascal 5.0.

B.1   Counting the States

The following program counts the number of possible states for a *Symmetric* star-shaped network with $K = 5$ of capacity $C = 3$. The program can only work with $C \leq 7$. Remember that for the reduced state-space model there is a formula to calculate the number of possible states; see §1.4.

A state is stored effectively as an array of $K(K + 1)/2$ elements which contains information on the number of single and 2-link calls present at this state. Every new state is given a 'registration' number and is then added into a hash table that contains the state space; see procedures `compactor`, `key`.

If the hash function was perfect, it would automatically put every incoming state into a different spot in the hash table. Unfortunately, hash functions tend to be imperfect. Unless we make the hash table excessively large, two or more different states will eventually be sent to the same spot. This is known as a *collision*. To solve this problem we have made each state entry the head of a linked list. We store incoming states by adding them to the appropriate linked list. If there is a

collision, we add a new element to the linked list associated with the particular hash value; see definition of `pothc` and procedures `addstatetohashtable, findstate, compactor, clearhashtore, inithashstore.`

Most of the procedures found in this programe can be found incorporated in the programs that perform the policy improvement and calculate the optimal policy as well as the Total Expected Discounted Reward (TEDR); see Appendix B.2.

Running the following program gives us an idea of the memory needed for storing the states as well as information about every state.

B.1.1 Program STATECOUNTER

```
program STATECOUNTER (input,output,State);
const

  NoLinks= 5;
  NoAllPos= 15;                           {the number K(K+1)/2}
  Cap= 3;


  hashmaxin= 1135333;
  compactlength= 5;        {bytes needed for compact storage =
                                  (3*NoAllPos)/8  rounded up }



type
  {Cap is as much as 7; i.e. octal}

  byte=0..225;
  Circuits = 0..Cap;
  Spare = array[1..NoLinks] of Circuits;
  Link = array[1..NoAllPos,1..2] of 1..NoLinks;
  Store = array[1..NoAllPos] of byte;
  compact=array[1..compactlength] of byte;
  hashnumber=0..hashmaxin;
  pothc=^collisionlist;
  ollisionlist=record
                    r:double;
                    state:compact;
                    hcnext:pothc;
               end;


var

  XXX,X : Store;
  Links : Link;
  F : Spare;
  Current,i,j,k,X1 : integer16;
  p:pothc;
```

```
    Counter : integer32;
    State: text;
    m:hashnumber;
    hashstore:array[hashnumber] of pothc;
    tadd,tcoll:integer32;
```

{***}

```
    procedure compactor(var s:Store;var h:compact);
    var
        bl,ch,cs:byte;

    begin

        h[1]:=s[NoAllPos];
        for ch:=2 to compactlength do h[ch]:=0;
        cs:=NoAllPos-1;
        ch:=1;
        bl:=5;
        while cs>0 do case bl of
          3..8:begin
                    h[ch]:=h[ch]+lshft(s[cs],(8-bl));
                    bl:=bl-3;
                    cs:=cs-1;
                end;
             2:begin
                    h[ch]:=h[ch]+(s[cs] mod 4)*64;
                    ch:=ch+1;
                    h[ch]:=(s[cs] div 4);
                    cs:=cs-1;
                    bl:=7;
                end;
             1:begin
                    h[ch]:=h[ch]+(s[cs] mod 2)*128;
                    ch:=ch+1;
                    h[ch]:=s[cs] div 2;
                    cs:=cs-1;
                    bl:=6;
                end;
             0:begin
                    ch:=ch+1;
                    bl:=8;
                end;
        end;
    end;
```

{***}

```
    procedure clearhashstore;

    var

        i:hashnumber;
        h,hh:pothc;

    begin

        for i:=0 to hashmaxin do
        begin
            h:=hashstore[i];
            while h<>nil do
            begin
                hh:=h;
                h:=h^.hcnext;
                dispose(hh);
            end;
        end;
```

```
    end;
```
{***}
```
    procedure inithashstore;
    var
        i:hashnumber;

    begin
        for i:=0 to hashmaxin do hashstore[i]:=nil;
    end;
```
{***}
```
    function key(var c:compact):hashnumber;          {need not be var}
    var
        a,k:integer32;
        i:1..compactlength;

    begin
        k:=0;
        for i:=1 to compactlength do
        begin
            a:=i*({not} c[i]);
            k:=k+sqr(a);
        end;
        key:=(k mod hashmaxin);
    end;
```
{***}
```
    procedure findstate(var y:Store;var h:pothc);   {need not be var}
    var
        found:boolean;
        j:integer16;
        c:compact;
    begin

        compactor(y,c);
        h:=hashstore[key(c)];
        m:=key(c);
        found:=false;

        while not ((h=nil) or found) do
        begin
            j:=0;
            with h^ do
                repeat
                    j:=j+1;
                    found:= c[j]=state[j];
                until found or (j=NoAllPos);
                if not found then h:=h^.hcnext;
        end;
    end;
```
{**}
```
    procedure addstatetohashtable(var y:Store);

    var
        kk:hashnumber;
        {j:integer16;}
        hh:pothc;
        c:compact;

    begin
```

```
        {if (tadd mod 100)=0 then writeln(memavail,' ',tadd,' ',tcoll);}
        tadd:=tadd+1;
        compactor(y,c);
        kk:=key(c);
        {writeln(kk);}
        new(hh);


        if hashstore[kk]=nil then
        begin
           hashstore[kk]:=hh;
           hh^.hcnext:=nil;
        end else begin
           tcoll:=tcoll+1;
           hh^.hcnext:=hashstore[kk];
           hashstore[kk]:=hh;
        end;

        with hh^ do
        begin
           state:=c;
        end;

        {write('HN ',kk:1,'    : ');
        for j:=1 to NoAllPos do write(' ',y[j]);
        writeln;
        write('compact: ');
        for j:=1 to compactlength do write(' ',c[j]:3);
        writeln;}

   end;

{****************************************************************}

{   The following procedure recursively counts the number    }
{ of states in a star network with number of links as above }
{ each having the same number (Cap) of circuits: the network}
{ carries 1 and 2-link calls.                               }


{****************************************************************}


procedure Count(F : Spare; Current : integer16);

var
     j,Limit: integer16;
     F1: Spare;
     Test: boolean;
     k:integer16;

begin

  X[Current] := 0; F1 := F;
  if F[Links[Current,1]] >= F[Links[Current,2]] then
    Limit := F[Links[Current,2]]
  else Limit := F[Links[Current,1]];

repeat


  begin
    F1[Links[Current,1]] := F[Links[Current,1]] - X[Current];
    if (Links[Current,2] <> Links[Current,1]) then
      F1[Links[Current,2]] := F[Links[Current,2]] - X[Current];
```

154

```
        Test := false;   j := NoAllPos + 1;

    repeat
      begin
        j:=j-1; Test := (F1[Links[j,1]]>0) and (F1[Links[j,2]]>0);
      end;
    until Test or (j = Current);


      if (j = Current) then
        begin Counter := Counter + Limit + 1;

        XXX:=X;
        for k:=0 to Limit do
        begin
          addstatetohashtable(XXX);
          XXX[Current]:=XXX[Current]+1;
        end;
        Limit := -1; end
      else
        begin
          Count(F1,Current+1);
          X[Current]:=X[Current]+1;
          Limit:=Limit-1;
        end;
    end;
until (Limit < 0);


end;          {This ends the recursive procedure.}


{********************* Main Program *********************************}


begin

  open(State,'State.txt','unknown');
  rewrite(State);


if Cap>7 then
      writeln('Compactor cannot handle Capacities larger than 7')
else    begin

      inithashstore;
      tcoll:=0;
      tadd:=0;

      {This identifies what type of calls X[k] records}

      for i := 1 to NoLinks do
        for j := i to NoLinks do
        begin
          X1 := i*(2*NoLinks - i + 1); X1 := (X1 div 2);
          k := X1 - (NoLinks - j);
          Links[k,1] := i; Links[k,2] := j;
        end;

      for i := 1 to NoLinks do F[i] := Cap;

      Counter:= 0;
      Current:= 1;
```

```
        Count(F,Current);

        writeln(State,'NoLinks = ',NoLinks,'  Capacity = ',Cap);
        writeln(State,'The number of possible states = ',Counter);
        writeln(State,'hashmaxin= ', hashmaxin);

        writeln('Therewere',tcoll:1,'collisionsalltoldin',tadd:1,'entrances.');
    clearhashstore;


end;

end.
```

B.2 Policy Improvement Program

The following program:

(a) counts the possible states;

(b) performs the value determination step discussed in §2.4.2; see procedure COST;

(c) performs the policy improvement analysed in §2.4.1 using the Succesive Over-Relaxation Algorithm (SOR) presented in §2.4.3; see procedures STATUS, IMPROVE;

(d) it effectively stores the optimal policy in the hash table of the states;

(e) calculates the *Admission Price* policy W_i as given in §4.1.1;

(e) Calculates and stores the optimal value function for every possible state; see procedure COST.

B.2.1 The Beginning

```
program vague33(input, output, File3);
{Output of states and their addresses, improvement of policy included}
{3,3-Case / UnixBerkeleyPascal / 1992-1993}


const
      NoLinks=3;
      Cap=3;

      ee=1E-6;                         {tolerance number}
      hashmaxin=953;

      NoAllPos=6;                      {number K(K+1)/2=  beta}
      NoPairs=3;                       {number K(K-1)/2= gamma}

      compactlength=3;
      {bytes needed for compact storage= 3*NoAllpos/8 rounded up}
      compactlengthplus1=4;

      discount=0.8;                    {the discount factor}
      omega=1.5;                       {the sor parameter}
type
      byte = 0..255;
      Circuits = 0..Cap;
      Spare = array[1..NoLinks] of Circuits;
      Sub = array[1..NoLinks] of double;
      Link = array[1..NoAllPos,1..2] of 1..NoLinks;
```

```pascal
Store = array[1..NoAllPos] of byte;
All=1..NoAllPos;
Pair=array [1..NoPairs] of boolean;
Extra=array [1..NoPairs] of integer16;
Index=array [1..NoPairs] of All;
Compact=array[1..compactlengthplus1] of byte;
hashnumber=0..hashmaxin;
pothc=^collisionlist;
collisionlist=record
                    value:double;
                    {difer:double;}
                    approx:Sub;
                    freecirc:integer16;
                    state:Compact;
                    hcnext:pothc;
                    Policy:Pair;
               end;
var

    File3:text;
    Links:Link;
    F:Spare;                 PolicyTest:boolean;
    Ind:Index;               X:Store;
    temp,h:pothc;
    tadd,tcoll,Counter:integer32;
    hashstore:array[hashnumber] of pothc;
    AveRew1,AvCostHome, RelRate1,RelRate2,N1,N2,a,rrr:double;
    ITER1,ITER2,ITER3,BETTER,Home,Current,Circ,repetition:integer16;
    Count,w,i,j,l,k,k1,k2,k3,control,Better,stop,Rew1,Rew2:integer16;
    Rate,RR,v,P1,P2,P3,XX,Dif,AbsDif,BigDif,AbsApprox,Par1,Par2:double;
    One,Two,Three,Onecount,Twocount,Threecount,f: double;

{************** procedures and functions **********************}


procedure COMPACTOR(var s:Store; var h:Compact);

var
    bl,ch,cs:byte;

begin

    h[1]:=s[NoAllPos];
    for ch:=2 to compactlength do h[ch]:=0;
    cs:=NoAllPos-1;
    ch:=1;
    bl:=5;
    while cs>0 do
     case bl of
      3..8:begin
              h[ch]:=h[ch]+lshft(s[cs],(8-bl));
              bl:=bl-3;
              cs:=cs-1;
           end;
         2:begin
              h[ch]:=h[ch]+(s[cs] mod 4)*64;
              ch:=ch+1;
              h[ch]:=(s[cs] div 4);
              cs:=cs-1;
              bl:=7;
           end;
         1:begin
              h[ch]:=h[ch]+(s[cs] mod 2)*128;
              ch:=ch+1;
              h[ch]:=s[cs] div 2;
              cs:=cs-1;
```

```
                           bl:=6;
                       end;
                   0:begin
                       ch:=ch+1;
                       bl:=8;
                     end;
              end;

      end;


{*****************************************************************}

procedure CLEARHASHStoRE;

var
    i:hashnumber;
    h,hh:pothc;

begin
      for i:=0 to hashmaxin do
      begin
          h:=hashstore[i];
          while h<>nil do
          begin
              hh:=h;
              h:=h^.hcnext;
              dispose(hh);
          end;
      end;
end;

{*****************************************************************}


procedure INITHASHStoRE;

var
      i:hashnumber;

begin

      for i:=0 to hashmaxin do hashstore[i]:=nil;

end;

{*****************************************************************}

function KEY(var c:Compact):hashnumber;   {need not be var}

var
    a,k:integer32;
    i:1..compactlength;

begin
      k:=0;
      for i:=1 to compactlength do
      begin
          a:=i*({not}c[i]);
          k:=k+sqr(a);
      end;
      KEY:=(k mod hashmaxin);

end;

{*****************************************************************}
```

```pascal
procedure FINDSTATE(var y:Store; var h:pothc);

var
     nfound:boolean;    j:integer16;
     c:Compact;

begin
     COMPACTOR(y,c);
     h:=hashstore[KEY(c)];
     c[compactlengthplus1]:=0;
     nfound:=true;
     while (h<>nil) and (nfound) do
     begin
          h^.state[compactlengthplus1]:=1;
          j:=0;
          repeat
               j:=j+1;
          until c[j]<>h^.state[j];

          nfound:=j<compactlengthplus1;
          if nfound then h:=h^.hcnext;
     end;
end;
{***************************************************************}


procedure ADDSTATEtoHASHTABLE(var y:Store);

var
     kk:hashnumber;
     j:integer16;
     hh:pothc;
     c:Compact;

begin
     {if (tadd mod 000)=0 then writeln(memavail,' ',tadd,' ',tcoll);}
     tadd:=tadd+1;
     COMPACTOR(y,c);
     kk:=KEY(c);
     new(hh);

     if hashstore[kk]=nil then
     begin
        hashstore[kk]:=hh;
        hh^.hcnext:=nil;
     end else begin
        tcoll:=tcoll+1;
        hh^.hcnext:=hashstore[kk];
        hashstore[kk]:=hh;
     end;

        with hh^ do
        begin
          state:=c;
          value:=20;  {initial value for policy improvement V_0}

          for j:=1 to NoPairs do Policy[j]:=false;
          for j:=1 to NoLinks do approx[j]:=0;
        end;

end;
{***************************************************************}
```

B.2.2 Value Determination Step

The following procedure performs the value determination step of the policy improvement by calculating the optimal value function for the state examined; this procedure is recursively called for all the possible states of the state-space. It also calculates the Average Reward by carrying 2-link traffic in the particular state examined.

```
{****************************************************************}

procedure COST(XLOCAL:Store; p:pothc; Count1:integer16; F1:Spare);

var
        i,which,Count2,find,multi,numero:integer16;
        Xpoint:pothc;
        Old,AvCostNeighbour,AveRew2:double;
        Neighbour,AbsApprox,pprox: Sub;
begin
        Count2:=0;  P1:=0; P2:=0; P3:=0; XX:=0;
        Xpoint:=p;
        Old:=Xpoint^.value;
        AvCostHome:=0; AveRew2:=0;
        F1:=F1;

        {if BETTER=0 then begin
           ACCEPT(F1,Xpoint);
        end;}

         for i:=1 to NoPairs do
         begin
            if Xpoint^.Policy[i] then Count2:=Count2+1;
         end;
   P1:=XLOCAL[1]+XLOCAL[2]+XLOCAL[3]+XLOCAL[4]+XLOCAL[5]+XLOCAL[6];
   P2:=NoLinks*Cap;
   P3:=(NoLinks-Count1)*N1+(NoPairs-Count2)*(N2/(NoLinks-1));

   XX:=(P2+P3-P1);
   XX:=XX*(1/Rate)*Old;

   AveRew2:=Rew2*N2*(Count2/RR);

   f:= 1-(a/Rate)*(P1+P2-P3);
   which:=1;


   for i:=1 to NoLinks do begin
           Neighbour[i]:=0; AbsApprox[i]:=0;
   end;



   for i:=1 to NoAllPos do       {investigates all the possible transitions}
      begin                      {from state XLOCAL}
        if XLOCAL[i]<Cap then
        begin
          XLOCAL[i]:=XLOCAL[i]+1;
          FINDSTATE(XLOCAL,p);
```

161

```
            XLOCAL[i]:=XLOCAL[i]-1;
            if (p<>nil)  then
            begin
                AvCostNeighbour:=p^.value;
                if (Links[i,1]=Links[i,2]) then        {1-link arrivals}
                begin
                  AvCostHome:=AvCostHome+a*RelRate1*AvCostNeighbour;
                  Neighbour[which]:=AvCostNeighbour;
                  which:=which+1;
                end else begin
                    find:=i-Links[i,1];                {2-link arrivals}
                    if Xpoint^.Policy[find] then
                    AvCostHome:=AvCostHome+a*RelRate2*AvCostNeighbour;
                end;  {if}
            end else begin
                if (Links[i,1]=Links[i,2]) then begin
                    Neighbour[which]:=0;
                    which:=which+1;
                end;     {if (Links}
            end;    {if p<>nil}
        end else begin
            if (Links[i,1]=Links[i,2]) then begin
                Neighbour[which]:=0;
                which:=which+1;  end;   {if Linl}
        end;      {if XLOC}

        if XLOCAL[i]>0 then                        {Possible departures}
        begin
            XLOCAL[i]:=XLOCAL[i]-1;
            FINDSTATE(XLOCAL,p);
            XLOCAL[i]:=XLOCAL[i]+1;
            if (p<>nil) then begin
            AvCostNeighbour:=p^.value;
            AvCostHome:=AvCostHome+a*XLOCAL[i]*AvCostNeighbour/Rate;
            end;
        end; {if}

end;  {for}



{**The optimality Equation under Succesive Over-Relaxation (SOR)**}

AvCostHome:=(1-omega)*Old+(omega/f)*(AveRew1+AveRew2+AvCostHome);



Dif:=AvCostHome-Old;

AbsDif:=abs(Dif);



for i:=1 to NoLinks do begin
    if Neighbour[i]<>0 then begin
        pprox[i]:=Neighbour[i]-AvCostHome;
        AbsApprox[i]:=abs(pprox[i]);
        Xpoint^.approx[i]:=AbsApprox[i];
    end else begin
        AbsApprox[i]:=0;
        Xpoint^.approx[i]:=0;
    end; {if}
end; {for}
```

```
          numero:=0;  multi:=1;
          for i:=NoLinks downto 1 do begin
                numero:=numero+multi*F1[i];
                multi:=multi*10;
          end;

          {Xpoint^.difer:=Dif;}

          Xpoint^.freecirc:=numero;
          Xpoint^.value:=AvCostHome;

          if AbsDif>BigDif then BigDif:=AbsDif;

          Old:=0;

  end;
{*************************************************************}

procedure COMMON(var F1:Spare;  F:Spare;
                 Current,Link1,Link2:integer16;
                 var j:integer16);
var
     Test:boolean;

begin

   F1[Link1]:=F[Link1]-X[Current];
   if (Link1<>Link2) then F1[Link2]:=F[Link2]-X[Current];
   Test:=false;  j:=NoAllPos+1;

   repeat
   begin
     j:=j-1;
     Test:=(F1[Links[j,1]]>0) and (F1[Links[j,2]]>0);
   end;
   until (Test) or (j=Current);

end;
{*************************************************************}

procedure COMMONHEAD(var Link1,Link2,Count1,Limit:integer16;
                     Current:integer16; var F1:Spare;F:Spare);

begin

   Link1:=Links[Current,1];
   Link2:=Links[Current,2];
   X[Current]:=0;
   Count1:=0;

   if F[Link1]>=F[Link2] then  Limit:=F[Link2]
                         else Limit:=F[Link1];
   F1:=F;

end;
{*************************************************************}

procedure  MAKESTATE(F:Spare; Current:integer16);

var
     Limit,i,j,Count1,Link1,Link2:integer16;
     F1:Spare;

begin
```

```
    COMMONHEAD(Link1,Link2,Count1,Limit,Current,F1,F);

  repeat

      COMMON(F1,F,Current,Link1,Link2,j);

      if        (j=Current) then
      begin
         Count1:=0;
         for i:=1 to NoLinks do if F1[i]>0 then Count1:=Count1+1;

         for j:=Current+1 to NoAllPos do  X[j]:=0;

         while    (X[Current]<=Limit) do
         begin
              if (X[Current]=Limit) and (Count1>0) and (Link1=Link2)
                  then Count1:=Count1-1;        {computeValueFncts}

              ADDSTATEtoHASHTABLE(X);

              X[Current]:=X[Current]+1;

              if X[Current]<=Limit then
              begin
               if F1[Link1]>0 then F1[Link1]:=F1[Link1]-1;
               if (Link1<>Link2) and (F1[Link2]>0) then
                             F1[Link2]:=F1[Link2]-1;
              end;

         end; {while}
      end else begin    {if}
         MAKESTATE(F1,Current+1);
         X[Current]:=X[Current]+1;
      end; {if}

    until (X[Current]>Limit);    {repeat}
end;
{*****************************************************************}
```

B.2.3 Policy Improvement

The following two procedures perform the policy improvemnt step. In changing the
policy:

(a) In procedure IMPROVE it works out all the policy actions when we are on a
particular state

(b) For each action calculates a value and remembers which action does 'best'. At
the end and in procedure STATUS.

Note the termination criterion if `BigDif>ee` in the main program.

```
{***************************************************************}
procedure IMPROVE(Ypoint:pothc; CurrentPair:integer16;
                  Z:double;   N:Extra;
                  var Action,BestAction:Pair;
                  var BigSum:real);
var
      w,i,No:integer16;  Y,Sum,AvCostNeighbour:double;  p:pothc;

begin

        w:=0;


repeat
        begin
                Action[CurrentPair]:=false;
                if w=1 then Action[CurrentPair]:=true;

                if w=0 then
                begin
                  AvCostHome:=Ypoint^.value;
                  Y:=Z+RelRate2*AvCostHome;
                end else begin
                  X[Ind[CurrentPair]]:=X[Ind[CurrentPair]]+1;
                  FINDSTATE(X,p);
                  X[Ind[CurrentPair]]:=X[Ind[CurrentPair]]-1;

                  if p<>nil then begin
                    AvCostNeighbour:=p^.value;
                    Y:=Z+RelRate2*AvCostNeighbour;
                  end else begin Y:=Z; end;
                end;  {if}

            if CurrentPair<NoPairs then
            begin
              IMPROVE(Ypoint,CurrentPair+1,Y,N,Action,BestAction,BigSum);
            end else begin
                    No:=0;
                    for i:=1 to NoPairs do
                    begin
                        if Action[i] then No:=No+1;
                    end;
                    Sum:=Rew2*N2*No/RR+a*Y;
                    if Sum>BigSum then
                    begin
                        BigSum:=Sum;
                        BestAction:=Action;
                    end;
            end;    {if}
        end;        {begin}
        w:=w+1;

until w>N[CurrentPair];

 end;


{***************************************************************}

procedure STATUS(F1:Spare; var p:pothc);
```

165

```pascal
var
        w,l1,l2,CurrentPair:integer16;   Output:boolean;
        Action,BestAction:Pair;   N:Extra; BigSum,Y,Sum:double;
        Ypoint:pothc;

begin
        for w:=1 to NoPairs do
        begin
            Action[w]:=false;
            BestAction[w]:=false;
            N[w]:=0;
            k1:=Ind[w];
            l1:=Links[k1,1];    l2:=Links[k1,2];
            if (F1[l1]>0) and (F1[l2]>0) then N[w]:=N[w]+1;
        end;

        CurrentPair:=1;   Y:=0;  Sum:=0;  BigSum:=0;

        Ypoint:=p;
        IMPROVE(Ypoint,CurrentPair,Y,N,Action,BestAction,BigSum);

        Output:=false;
        for w:=1 to NoPairs do
        begin
            if p^.Policy[w]<>BestAction[w] then
            begin
                p^.Policy[w]:=BestAction[w];
                PolicyTest:=false;
            end;

            if ITER1>=1 then}
                    if (not BestAction[w]) and (N[w]=1) then
                    begin
                            Output:=true;
                            writeln(File3,w);
                    end;
        end;

        if Output then begin
            writeln(File3,'X =',X[1],X[2],X[3],X[4],X[5],X[6]);
            writeln(File3,'F=',F1[1],F1[2],F1[3]);
        end;
end;



{***********************************************************}


procedure FINDAVCOST(F:Spare;  Current:integer16);

var
     Limit,i,j,Count1,Link1,Link2:integer16;
     F1:Spare;  p:pothc;

begin

   COMMONHEAD(Link1,Link2,Count1,Limit,Current,F1,F);

repeat

        COMMON(F1,F,Current,Link1,Link2,j);

        if       (j=Current) then
        begin
            Count1:=0;
```

166

```
                for i:=1 to NoLinks do if F1[i]>0 then Count1:=Count1+1;

                for j:=Current+1 to NoAllPos do  X[j]:=0;

                while   (X[Current]<=Limit) do
                begin
                  if (X[Current]=Limit) and (Count1>0) and (Link1=Link2)
                        then Count1:=Count1-1;

                    FINDSTATE(X,p);

                    if p<>nil then
                    begin
                       AveRew1:=Rew1*N1*(Count1/Rate);
                       COST(X,p,Count1,F1);
                    end else begin
                       writeln('ERRor-NIL POINTER');
                    end;

                    X[Current]:=X[Current]+1;

                    if X[Current]<=Limit then
                    begin
                     if F1[Link1]>0 then    F1[Link1]:=F1[Link1]-1;
                     if (Link1<>Link2) and (F1[Link2]>0) then
                                           F1[Link2]:=F1[Link2]-1;
                    end;

                end; {while}

          end else begin    {if}
             FINDAVCOST(F1,Current+1);
             X[Current]:=X[Current]+1;
          end; {if}

    until (X[Current]>Limit);   {repeat}

    end;

{*****************************************************************}

procedure  USEAVCOST(F:Spare;  Current:integer16);

var
      Limit,j,Count1,Link1,Link2:integer16;
      F1:Spare;  p:pothc;

begin

   COMMONHEAD(Link1,Link2,Count1,Limit,Current,F1,F);

repeat

      COMMON(F1,F,Current,Link1,Link2,j);

      if (j=Current) then
      begin
         for j:=Current+1 to NoAllPos do  X[j]:=0;

         while   (X[Current]<=Limit) do
         begin
             FINDSTATE(X,p);

             if p<>nil then STATUS(F1,p);

             X[Current]:=X[Current]+1;
```

```
                    if X[Current]<=Limit then
                    begin
                     if F1[Link1]>0 then F1[Link1]:=F1[Link1]-1;
                     if (Link1<>Link2) and (F1[Link2]>0) then
                                     F1[Link2]:=F1[Link2]-1;
                    end;

              end; {while}
          end else begin    {if}
             USEAVCOST(F1,Current+1);
             X[Current]:=X[Current]+1;

          end; {if}

    until (X[Current]>Limit);    {repeat}

    end;

{***************************************************************}
{----------------------------MAIN PROGRAM--------------------}
{***************************************************************}

begin

          N1:=0.5;  N2:=4.9;    {arrival rates for 1 and 2-link trafic}

          Rew1:=2;  Rew2:=1;    {rewards for 1 and 2-link traffic}

          stop:=1;

          open(File3,'File3.txt','unknown');
          rewrite(File3);


repeat

          Rate:=(N1+N2/2+Cap)*NoLinks;    {The rates of events}


          RelRate1:=N1/Rate;
          RelRate2:=N2/((NoLinks-1)*Rate);
          RR:=Rate*(NoLinks-1);
          Par1:=N1/N2;
          Par2:=N1+(N2/2);


          rrr:=1/Rate;
          a:=exp(rrr*ln(discount));  {the alpha^(1/Rate)}
          writeln(File3,'a is=',a);

          ITER1:=0;     BETTER:=0;

          INITHASHStoRE;

          tcoll:=0;  tadd:=0;

          for i:=1 to NoLinks do  for j:=i to NoLinks do
          begin
            k1:=i*(2*NoLinks-i+1);  k2:=(k1 div 2);
            k:=k2-(NoLinks-j);        Links[k,1]:=i;
            Links[k,2]:=j;    k3:=k-i;
            if i<>j then Ind[k3]:=k;
          end;

          for i:=1 to NoLinks do F[i]:=Cap;
```

```
        Counter:=0;  Current:=1;

            MAKESTATE(F,Current);

        writeln(File3,'NoLinks',NoLinks);
        writeln(File3,' Capacity',Cap);
        writeln(File3,'ArrivalRates for 1 and 2 linkCalls =',N1,N2);
        writeln(File3,'Rewards Rew1, Rew2=',Rew1,Rew2);
        writeln(File3,'ee =',ee);
        writeln(File3,'n1/n2 =',Par1,',','n1+n2/2 =',Par2);
        writeln(File3,'hasmaxin' ,hashmaxin);
        writeln(File3,'There were',tcoll:1,'collisions');
        writeln(File3,'All told in ',tadd:1,'entrances');

        {CLEARHASHStoRE;}

        ITER1:=0;          BETTER:=0;       ITER3:=0;


repeat

          BigDif:=0;  Count:=0;  ITER2:=0;


          {*****starting value determination for every state*****}


          while (ITER2=0) do
          begin
              for i:=1 to NoLinks do  F[i]:=Cap;
              for w:=1 to NoAllPos do   X[w]:=0;
              Current:=1;                BigDif:=0;

              FINDAVCOST(F,Current);

              Count:=Count+1;

              if BigDif>ee then
              begin
                  BigDif:=0;
              end else begin
                  ITER2:=ITER2+1;
                  h:=hashstore[0];
                  writeln(File3,'AvCost[0]=',h^.value);
              end;  {if}
          end;   {while}


          {**********starting policy improvement******************}


      PolicyTest:=true;  {BETTER:=1;}

       for i:=1 to NoLinks do F[i]:=Cap;
       for i:=1 to NoAllPos do X[i]:=0;
       Current:=1;

       USEAVCOST(F,Current);

       ITER1:=ITER1+1;

  until (ITER1>4) or (PolicyTest);


{end of policy improvement}
```

```
{Calculate the Admission Price Policy w's}
```

B.2.4   Calculating the Ω Policy

```
{***************  Search for states with part. F1's **************}


for i:=0 to hashmaxin do
begin


if hashstore[i]<>nil then begin   {through all the possible states}
temp:=hashstore[i];
repetition:=0;

repeat

 Circ:=temp^.freecirc;     {keep track of their number of free circuits}
        if Circ=013 then begin
           Three:=Three+temp^.approx[3];
           One:=One+temp^.approx[2];
           Onecount:=Onecount+1;
           Threecount:=Threecount+1;   end;

        if Circ=033 then begin
           Three:=Three+temp^.approx[3];
           Three:=Three+temp^.approx[2];
           Threecount:=Threecount+1;
           Threecount:=Threecount+1;   end;

        if Circ=023 then begin
           Two:=Two+temp^.approx[2];
           Three:=Three+temp^.approx[3];
           Twocount:=Twocount+1;
           Threecount:=Threecount+1;   end;

        if Circ=011 then begin
           One:=One+temp^.approx[2];
           One:=One+temp^.approx[3];
           Onecount:=Onecount+1;
           Onecount:=Onecount+1;        end;

        if Circ=010 then begin
           One:=One+temp^.approx[2];
           Onecount:=Onecount+1;        end;

        if Circ=022 then begin
           Two:=Two+temp^.approx[3];
           Two:=Two+temp^.approx[2];
           Twocount:=Twocount+1;
           Twocount:=Twocount+1;        end;

        if Circ=021 then begin
           Two:=Two+temp^.approx[2];
           One:=One+temp^.approx[3];
           Onecount:=Onecount+1;
```

```
            Twocount:=Twocount+1;        end;

      if Circ= 030 then begin
         Three:=Three+temp^.approx[2];
         Threecount:=Threecount+1;   end;

      if Circ= 012 then begin
         One:=One+temp^.approx[2];
         Two:=Two+temp^.approx[3];
         Onecount:=Onecount+1;
         Twocount:=Twocount+1;        end;

      if Circ=020 then begin
         Two:=Two+temp^.approx[2];
         Twocount:=Twocount+1;        end;

      if Circ= 003 then begin
         Three:=Three+temp^.approx[3];
         Threecount:=Threecount+1;   end;

      temp:=temp^.hcnext;
                                       {check if there was a collision}
until (temp=nil);

   end; {if}

 end; {for}

{**********************************************************************}

W1:=One/Onecount;
W2:=Two/Twocount;                 {calculates the w's}
W3:=Three/Threecount;


writeln(File3,'Ws =',W1,',',W2);
writeln(File3,W3);




{**********************************************************************}

 stop:=stop+1;                       {repeat for different arrival rates}

 if (stop=2) then begin  N1:=0.7;  N2:=4.5;  Rew1:=2;  Rew2:=1; end;
 if (stop=3) then begin  N1:=1;    N2:=3.9;  Rew1:=2;  Rew2:=1; end;
 if (stop=4) then begin  N1:=1.3;  N2:=3.3;  Rew1:=2;  Rew2:=1; end;
 if (stop=5) then begin  N1:=1.5;  N2:=2.9;  Rew1:=2;  Rew2:=1; end;

      .........................................

 if (stop=19) then begin N1:=0.5;  N2:=6;    Rew1:=2;  Rew2:=1; end;


      .........................................

 if (stop=36) then begin N1:=2;    N2:=16;   Rew1:=2;  Rew2:=1; end;

until (stop>36);

end.
```

B.3 The Reduced State-Space Model

The following program contains the policy improvement for a reduced state-space network as presented in Chapter 5.

```
program SPLITimprovement(input,output,Split43);
{2-link calls splitting model; Calculates the relative values w }
const
        NoLinks = 4;
        Cap = 3;

        NoPairs = 6 ;                               {the number K(K-1)/2}

        Nostates = 256 ;
        ee= 1E-6;
        discount= 0.8;

type
        Circuits = 0..Cap;
        NoPostates = array [0..Cap,0..Cap,0..Cap,0..Cap] of real;
        Strategy = array [0..Cap,0..Cap,0..Cap,0..Cap,1..6] of integer16;
        Sub = array [1..NoLinks] of real;
        Spare = array [1..NoLinks] of Circuits;
        pothc = record
                        freecirc: integer16;
                        approx: Sub
                    end;

var

        Split43: text;
        PolicyTest: boolean;
        Policy,CheckPolicy: Strategy;
        AvCost,AveRew1,AveRew: NoPostates;
        state: array [0..Cap,0..Cap,0..Cap,0..Cap] of pothc;
        X,Y,Z,U,l,Count,BigITER,Count1,Count2,w: integer16;
        ITER,i,j,k,Rew1,Rew2,stop,which,I5pass: integer16;
        multi,numero,Circ:  integer16;
        X1,X2,X3,X4,X5,X6,X7,RelRate1,RelRate2,Y1,Y2,Y3,Y4,Y5,Y6: real;
        Dif,AbsDif,Num,Sum,BigSum,AveRew2,X8,X9,X10,X11,a: real;
        Rate,P1,P2,P3,P4,XX,RR,Costs,BigDif,Old,N1,N2,rrr: real;
        Neighbour, AbsApprox, pprox: Sub;
        F1: Spare;
        status: pothc;
        OneSum,TwoSum,ThreeSum: real;
        OneCount,TwoCount,ThreeCount,w1,w2,w3: real;
        I1,I2,I3,I4,I5,I6,I1pass,I2pass,I3pass,I4pass: integer16;
        Post1,Post2,Post3,Post4,Post5,Post6,I6pass: integer16;

{**********************************************************************}

begin
 open(Split43,'Split43.txt','unknown');
 rewrite(Split43);

 Rew1:=2;    Rew2:=1;

 N1:=0.5;    N2:=4.9;
```

172

```
    stop:= 1;
  repeat                              {for various pairs of N1 and N2}

   Rate:=(N1+N2/2+Cap)*NoLinks;         {calculate the rates of events}

   rrr:=1/Rate;
   a:=exp(rrr*ln(discount));
   writeln(Split43,'a is=',a);
   writeln(Split43,'arrivals are=',N1,',',N2);

   RelRate1:=N1/Rate;
   RelRate2:=N2/((NoLinks-1)*Rate);
   RR:=Rate*(NoLinks-1);
                                        {initialize the arrays}

   for i:= 0 to Cap do
   for j:= 0 to Cap do
   for k:= 0 to Cap do
   for l:= 0 to Cap do
   begin
       AveRew[i,j,k,l]:= 0;
       AveRew1[i,j,k,l]:= 0;
       for w:=1 to NoPairs do Policy[i,j,k,l,w]:=0;
       AvCost[i,j,k,l]:= 20;
   end;


   for X:=0 to Cap do       {calculates the AveRew1 for single traffic}
   for Y:=0 to Cap do
   for Z:=0 to Cap do
   for U:=0 to Cap do
   begin
       Count1:=0;
       if (X<=Cap) and (Y<=Cap) and (Z<=Cap) and (U<=Cap) then
       begin
          if (X<Cap) then Count1:=Count1+1;
          if (Y<Cap) then Count1:=Count1+1;
          if (Z<Cap) then Count1:=Count1+1;
          if (U<Cap) then Count1:=Count1+1;

          AveRew1[X,Y,Z,U]:=Rew1*N1*(Count1/Rate);
       end;
   end;

BigITER:=0;

{***********************************************************************}

repeat                                   {including the improvments}

 BigDif:=0;  ITER:=0;   Count:=0;
```

B.3.1   Value Determination Step

```
  repeat                               {just the finavCost routines}
```

```
Count:=Count+1;
for X:=0 to Cap do
for Y:=0 to Cap do
for Z:=0 to Cap do
for U:=0 to Cap do
begin

 if (X<=Cap) and  (Y<=Cap) and (Z<=Cap) and (U<=Cap) then
 begin

   Count1:=0;
   status:= state[X,Y,Z,U];

   if (X<Cap) then  Count1:=Count1+1;
   if (Y<Cap) then  Count1:=Count1+1;
   if (Z<Cap) then  Count1:=Count1+1;
   if (U<Cap) then  Count1:=Count1+1;

   F1[1]:= Cap - X;
   F1[2]:= Cap - Y;
   F1[3]:= Cap - Z;
   F1[4]:= Cap - U;

   Old:= AvCost[X,Y,Z,U];
   AvCost[X,Y,Z,U]:= 0;

   Count2:=0;
   which:= 1;

   for j:=1 to NoLinks do
   begin
       Neighbour[j]:=0;
       AbsApprox[j]:=0;
   end;


   X1:=0;   X2:=0; X3:=0; X4:=0;  X5:=0; X6:=0;
   X7:=0;   X8:=0; X9:=0; X10:=0; X11:=0;

   if (X<Cap) then
   begin
      X1:= a*RelRate1*AvCost[X+1,Y,Z,U];
      Neighbour[which]:= AvCost[X+1,Y,Z,U];
      which:= which + 1;
      if (Y<Cap) then begin
        X2:= a*RelRate2*Policy[X,Y,Z,U,1]*AvCost[X+1,Y+1,Z,U];
      end;
      if (Z<Cap) then begin
        X3:= a*RelRate2*Policy[X,Y,Z,U,2]*AvCost[X+1,Y,Z+1,U];
      end;
      if (U<Cap) then begin
        X4:= a*RelRate2*Policy[X,Y,Z,U,3]*AvCost[X+1,Y,Z,U+1];
      end;
   end else begin
     Neighbour[which]:= 0;
     which:= which+1;
   end;


    if (Y<Cap) then
    begin
      X5:= a*RelRate1*AvCost[X,Y+1,Z,U];
      Neighbour[which]:= AvCost[X,Y+1,Z,U];
      which:= which + 1;
```

174

```
                   if (Z<Cap) then begin
                       X6:= a*RelRate2*Policy[X,Y,Z,U,4]*AvCost[X,Y+1,Z+1,U];
                   end;
                   if (U<Cap) then begin
                       X7:= a*RelRate2*Policy[X,Y,Z,U,5]*AvCost[X,Y+1,Z,U+1];
                    end;
                 end else begin
                   Neighbour[which]:= 0;
                   which:= which+1;
              end;


              if (Z<Cap) then
              begin
                  X8:= a*RelRate1*AvCost[X,Y,Z+1,U];
                  Neighbour[which]:= AvCost[X,Y,Z+1,U];
                  which:= which + 1;
                  if (U<Cap) then begin
                      X9:= a*RelRate2*Policy[X,Y,Z,U,6]*AvCost[X,Y,Z+1,U+1];
                  end;
              end else begin
                  Neighbour[which]:= 0;
                  which:= which+1;
              end;


              if (U<Cap) then
              begin
                  X10:= a*RelRate1*AvCost[X,Y,Z,U+1];
                  Neighbour[which]:= AvCost[X,Y,Z,U+1];
                  which:= which + 1;
              end else begin
                  Neighbour[which]:= 0;
                  which:= which+1;
              end;



X11:= X*AvCost[X-1,Y,Z,U] + Y*AvCost[X,Y-1,Z,U] + Z*AvCost[X,Y,Z-1,U];
X11:= X11 + U*AvCost[X,Y,Z,U-1];
X11:= a*(X11/Rate);

Count2:= Policy[X,Y,Z,U,1] + Policy[X,Y,Z,U,2] + Policy[X,Y,Z,U,3];
Count2:= Count2 + Policy[X,Y,Z,U,4] + Policy[X,Y,Z,U,5];
Count2:= Count2 + Policy[X,Y,Z,U,6];

P1:= X+Y+Z+U;
P2:= NoLinks*Cap;
P3:= (NoLinks-Count1)*N1+(NoPairs-Count2)*(N2/(NoLinks-1));
XX:= (P3+P2-P1);
XX:= XX*(1/Rate)*Old;
AveRew2:= Rew2*N2*(Count2/RR);


Costs:= X1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9 + X10 + X11 + a*XX;

AvCost[X,Y,Z,U]:= AveRew1[X,Y,Z,U] + AveRew2 + Costs;

   Dif:= AvCost[X,Y,Z,U] - Old;
   AbsDif:= abs(Dif);
   Old:= 0;

   if AbsDif>BigDif then BigDif:=AbsDif;


         for j:= 1 to NoLinks do
```

```
               begin
                 if Neighbour[j]<>0 then
                 begin
                    pprox[j]:= Neighbour[j] - AvCost[X,Y,Z,U];
                    AbsApprox[j]:= abs(pprox[j]);
                    state[X,Y,Z,U].approx[j]:= AbsApprox[j];
                 end else
                 begin
                  AbsApprox[j]:= 0;
                  state[X,Y,Z,U].approx[j]:= 0;
                 end;
               end;


             numero:= 0;         multi:= 1;
             for j:= NoLinks  downto 1 do begin
                   numero:= numero + multi*F1[j];
                   multi:= multi*10;
             end;

             state[X,Y,Z,U].freecirc:= numero;

          end; end;


          if BigDif>ee  then
          begin
            BigDif:=0;
          end else begin
            ITER:=ITER+1;
            writeln(Split43,'AvCost[0]=',AvCost[0,0,0,0]);
            {writeln(Split43,'Count is=',Count);}
            {writeln(Split43,'ee is=',ee);}
            BigDif:=0;
          end;


until (ITER>0);
```

B.3.2   Policy Improvement

```
PolicyTest:= true;
BigSum:= 0;

for i:=0 to Cap do
for j:=0 to Cap do
for k:=0 to Cap do
for l:=0 to Cap do
begin
 CheckPolicy[i,j,k,l,1]:=0;
 CheckPolicy[i,j,k,l,2]:=0;
 CheckPolicy[i,j,k,l,3]:=0;
 CheckPolicy[i,j,k,l,4]:=0;
 CheckPolicy[i,j,k,l,5]:=0;
 CheckPolicy[i,j,k,l,6]:=0;
end;
```

```
for X:=0 to Cap do
for Y:=0 to Cap do
for Z:=0 to Cap do
for U:=0 to Cap do begin

if (X<=Cap) and (Y<=Cap) and (Z<=Cap) and (U<=Cap) then begin

        I1pass:=0;  I2pass:=0;  I3pass:=0;
        I4pass:=0;  I5pass:=0;  I6pass:=0;

    if (X<Cap) and (Y<Cap) then  I1pass:=1;
    if (X<Cap) and (Z<Cap) then  I2pass:=1;
    if (X<Cap) and (U<Cap) then  I3pass:=1;
    if (Y<Cap) and (Z<Cap) then  I4pass:=1;
    if (Y<Cap) and (U<Cap) then  I5pass:=1;
    if (Z<Cap) and (U<Cap) then  I6pass:=1;


            BigSum:=0;
            for I1:=0 to I1pass do
            for I2:=0 to I2pass do
            for I3:=0 to I3pass do
            for I4:=0 to I4pass do
            for I5:=0 to I5pass do
            for I6:=0 to I6pass do  begin

                Y1:=RelRate2*I1*AvCost[X+1,Y+1,Z,U];
                Y2:=RelRate2*I2*AvCost[X+1,Y,Z+1,U];
                Y3:=RelRate2*I3*AvCost[X+1,Y,Z,U+1];
                Y4:=RelRate2*I4*AvCost[X,Y+1,Z+1,U];
                Y5:=RelRate2*I5*AvCost[X,Y+1,Z,U+1];
                Y6:=RelRate2*I6*AvCost[X,Y,Z+1,U+1];


    Num:= Y1+Y2+Y3+Y4+Y5+Y6;
    Num:= Num + (NoPairs-I1-I2-I3-I4-I5-I6)*AvCost[X,Y,Z,U]*RelRate2;

        AveRew2:= Rew2*N2*(I1+I2+I3+I4+I5+I6)/RR;

        Sum:= AveRew2 + a*Num;

                {find the maximum sum}
                if Sum>BigSum then
                begin
                        BigSum:= Sum;
                        Post1:=I1;
                        Post2:=I2;
                        Post3:=I3;
                        Post4:=I4;
                        Post5:=I5;
                        Post6:=I6;
                end;

            end;  {for loop}

        CheckPolicy[X,Y,Z,U,1]:= Post1;
        CheckPolicy[X,Y,Z,U,2]:= Post2;
        CheckPolicy[X,Y,Z,U,3]:= Post3;
        CheckPolicy[X,Y,Z,U,4]:= Post4;
        CheckPolicy[X,Y,Z,U,5]:= Post5;
        CheckPolicy[X,Y,Z,U,6]:= Post6;


        for w:= 1 to NoPairs do
        begin
```

```
                    if CheckPolicy[X,Y,Z,U,w]<>Policy[X,Y,Z,U,w] then
                    begin
                        Policy[X,Y,Z,U,w]:= CheckPolicy[X,Y,Z,U,w];
                        PolicyTest:= false;
                    end;
                end;

  end; end;

  BigITER:=BigITER+1;


until (BigITER>4) or (PolicyTest);

{***********************************************************}

{end of improvement}

    OneSum:=0;        TwoSum:=0;      ThreeSum:=0;
    OneCount:=0;      TwoCount:=0;    ThreeCount:=0;


  for X:=0 to Cap do
  for Y:=0 to Cap do
  for Z:=0 to Cap do
  for U:=0 to Cap do begin

  if (X<=Cap) and  (Y<=Cap) and (Z<=Cap)  and (U<=Cap) then begin


        {through all the possible states}

        status:= state[X,Y,Z,U];

        Circ:= status.freecirc;            {keep track of their F's}

        if Circ=0013 then begin
            OneSum:= OneSum+state[X,Y,Z,U].approx[3];
            OneCount:= OneCount+1;
            ThreeSum:= ThreeSum+state[X,Y,Z,U].approx[4];
            ThreeCount:= ThreeCount+1;
        end;

        ..............................................

        if Circ=0020 then begin
            TwoSum:= TwoSum+state[X,Y,Z,U].approx[3];
            TwoCount:= TwoCount+1;
        end;

  end;  end; {for}

  W1:= OneSum/OneCount;   W2:= TwoSum/TwoCount;
  W3:= ThreeSum/ThreeCount;

  writeln(Split43,W1,',',W2,',',W3);

  stop:= stop + 1;

      if (stop=2) then begin  N1:=0.7;  N2:=4.5;  end;

      ..............................................

until (stop>30);

end.
```

B.4   Employing the Ω Policy

The following sample contains procedures as well as part of the main program which
is used to employ the Ω policy calculated from programs similar to those in Appendix
B.2 as the admission rule on accepting or rejecting 2-link calls.

```
program vague35readwValues(input, output, File35w);

{3-5 case; w's read from vague35.pas, stored in File35;
read the w values and find  the w-optimal V_w(0)}

const
        NoLinks=3;       Cap=5;
        ee=1E-6;
        hashmaxin=5531;
        NoAllPos=6;      NoPairs=3;
        compactlength=3;
        {bytes needed for compact storage= 3*NoAllpos/8 rounded up}
        compactlengthplus1=4;
        discount=0.8;
type
        byte = 0..255;
        Circuits = 0..Cap;
        Capnumb=0..Cap;
        Spare = array[1..NoLinks] of Circuits;
        Walues = array[Capnumb] of real;
        Link = array[1..NoAllPos,1..2] of 1..NoLinks;
        Store = array[1..NoAllPos] of byte;
        All=1..NoAllPos;
        Extra=array [1..NoPairs] of integer16;
        Index=array [1..NoPairs] of All;
        Compact=array[1..compactlengthplus1] of byte;
        hashnumber=0..hashmaxin;
        pothc=^collisionlist;
        collisionlist=record
                        value:real;
                        state:Compact;
                        hcnext:pothc;
                    end;


. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

procedure COMPACTOR.............

procedure CLEARHASHSTORE........

procedure KEY...................

procedure FINDSTATE.............

procedure ADDSTATEtoHASHSTABLE..
```

```
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

procedure COST(XLOCAL:Store; p:pothc; Count1:integer16; F1:Spare);
var     i,k1,l1,l2,Count2:integer16;        Xpoint:pothc;
        Old,AvCostNeighbour,AveRew2:real;

begin
        Count2:=0;  P1:=0; P2:=0; P3:=0; XX:=0;
        Xpoint:=p;
        Old:=Xpoint^.value;
        AvCostHome:=0; AveRew2:=0;
        F1:=F1;

        for i:=1 to NoPairs do
        begin
          k1:=Ind[i];
          l1:=Links[k1,1];  l2:=Links[k1,2];
          if W[F1[l1]]+W[F1[l2]]<=Rew2 then Count2:=Count2+1;
        end;

        . . . . . . . . . . . . . . . . . . . . . . . . . . .

  for i:=1 to NoAllPos do
     begin
        if XLOCAL[i]<Cap then
        begin
           XLOCAL[i]:=XLOCAL[i]+1;
           FINDSTATE(XLOCAL,p);
           XLOCAL[i]:=XLOCAL[i]-1;
           if (p<>nil)  then
           begin
             AvCostNeighbour:=p^.value;
             if (Links[i,1]=Links[i,2]) then
             begin
               AvCostHome:=AvCostHome+a*RelRate1*AvCostNeighbour;
             end else begin
                 l1:=Links[i,1];              {employing the Omega policy}
                 l2:=Links[i,2];
                 if W[F1[l1]]+W[F1[l2]]<=Rew2 then
                     AvCostHome:=AvCostHome+a*RelRate2*AvCostNeighbour;
             end;   {if (Lin}
           end;     {if p<>nil}
        end;        {if XLOCAL}


. . . . . . . . . . . . . . . . . . . . . . . . .

procedure COMMONHEAD(var Link1,Link2,Count1,Limit:integer16;
                    Current:integer16; var F1:Spare;F:Spare);

. . . . . . . . . . . . . . . . . . . . . . . . .

procedure  MAKESTATE(F:Spare; Current:integer16);

. . . . . . . . . . . . . . . . . . . . . . . . .

{-----------------------------MAIN PROGRAM-----------------------------}

begin
        N1:=0.5;    N2:=4.9;  Rew1:=2;  Rew2:=1;

        stop:=25;        W[0]:=99999;

        W[1]:=0.5699;   W[2]:=0.2766;
        W[3]:=0.2307;   W[4]:=0.1566;
```

```
        W[5]:=0.1321;

        open(File35w,'File35w.txt','unknown');
        rewrite(File35w);

repeat

        Rate:=(N1+N2/2+Cap)*NoLinks;

.........................................

        if (stop=28) then begin
           N1:=1.3;   N2:=5.4;
           W[1]:=0.7249;  W[2]:=0.3673;  W[3]:=0.3055;  W[4]:=0.2203;
           W[5]:=0.1891;  end;
        if (stop=29) then begin
           N1:=1.6;   N2:=4.8;
           W[1]:=0.7776;  W[2]:=0.3927;  W[3]:=0.3030;  W[4]:=0.2193;
           W[5]:=0.1850;  end;
        if (stop=30) then begin
           N1:=2;     N2:=4;
           W[1]:=0.8508;  W[2]:=0.4403;  W[3]:=0.3174;  W[4]:=0.2262;
           W[5]:=0.1811;  end;

until (stop>30);

end.
```

B.5 What Happens as $R_2 \approx 2R_1$

In this section we present parts of an un-finished program in which after the optimal policy onrejecting 2-link calls, we examine policy improvement by rejecting 1-link calls in order to maximise the TEDR; see also in §3.4. Most parts of the following program are included in program in Appendix B.2.

. . . .

```
procedure IMPROVEB(Ypoint: pothc; CurrentSingle: integer16;
             Z: double;   N: Extra;
             var Actionb,BestActionb: Traffic;
             var BigSumb: double);

var
      w,i,No: integer16;  Y,Sumb,AvCostNeighbour: double;  p: pothc;

begin

      w:=0;
      repeat
      begin
            Actionb[CurrentSingle]:=false;
            if w=1 then Actionb[CurrentSingle]:=true;

            if w=0 then
            begin
              AvCostHome:=Ypoint^.value;
              Y:=Z+RelRate1*AvCostHome;
            end else begin
              X[Ind[CurrentSingle]]:=X[Ind[CurrentSingle]]+1;
              FINDSTATE(X,p);
              X[Ind[CurrentSingle]]:=X[Ind[CurrentSingle]]-1;
              if p<>nil then begin
                AvCostNeighbour:=p^.value;
                Y:=Z+RelRate1*AvCostNeighbour;
              end else begin Y:=Z; end;
            end;   {if}

          if CurrentSingle<NoLinks then
          begin

            IMPROVEB(Ypoint,CurrentSingle+1,Y,N,Actionb,BestActionb,BigSumb

          end else begin
                No:=0;
                for i:=1 to NoLinks do
                begin
                    if Actionb[i] then No:=No+1;
                end;
                Sumb:=Rew1*N1*(No/Rate)+a*Y;
                if Sumb>BigSumb then
                begin
                    BigSumb:=Sumb;
                    BestActionb:=Actionb;
                end;
          end;     {if}
      end;         {begin}
```

```
                w:=w+1;
                until w>N[CurrentSingle];

    end;

        ............................................

procedure STATUSB(F1:Spare; var p:pothc);

var
        w, CurrentSingle: integer16;  Output: boolean;
        Actionb,BestActionb: Traffic;
        N: Extra;
        BigSumb,Y,Sumb: double;  Ypoint: pothc;

begin

        for w:=1 to NoLinks do
        begin
            Actionb[w]:=false;
            BestActionb[w]:=false;
            N[w]:=0;
            if (F1[w]>0) then N[w]:=N[w]+1;
        end;

        CurrentSingle:=1;   Y:=0;   Sumb:=0;   BigSumb:=0;

        Ypoint:=p;

        IMPROVEB(Ypoint,CurrentSingle,Y,N,Actionb,BestActionb,BigSumb);


        Output:=false;
        for w:=1 to NoLinks do
        begin
            if p^.Single[w]<>BestActionb[w] then
            begin
                p^.Single[w]:=BestActionb[w];
                PolicySingle:=false;
            end;

            if ITER1>=1 then
                    if (not BestActionb[w]) and (N[w]=1) then
                    begin
                        Output:=true;
                        writeln(change,'oops change=',w);
                    end;
        end;

  if Output then begin
    writeln(change,'CHANGING the OPTIMAL POLICY');
    writeln(change,'Xb =',X[1],X[2],X[3],X[4],X[5],X[6]);
    writeln(change,'Fb=',F1[1],F1[2],F1[3]);
    writeln(change,'-----------------------');
  end;

        ............................................

{In the main program}

    RejectXY:=0;
            PolicyTest:=true;

            for i:=1 to NoLinks do F[i]:=Cap;
            for i:=1 to NoAllPos do X[i]:=0;
```

183

```
                    Current:=1;

                    USEAVCOST(F,Current);    ITER1:=ITER1+1;

                    writeln(change,'No States XY was optimally rejected=',RejectXY);

        until (ITER1>4) or (PolicyTest);
                    writeln(change,'Check if there is a change in Optimal Policy');
                  ITERb:=0;
      repeat
                    PolicySingle:=true;

                    for i:=1 to NoLinks do  F[i]:=Cap;
                    for i:=1 to NoAllPos do X[i]:=0;
                    Current:=1;

                    USEAVCOST2(F,Current);

                    ITERb:=ITERb+1;      ITER2:=0;

                    BigDif:=0;  Count:=0;   ITER2:=0;

                    while (ITER2=0) do
                    begin
                        for i:=1 to NoLinks do  F[i]:=Cap;
                        for w:=1 to NoAllPos do   X[w]:=0;
                        Current:=1;                BigDif:=0;

                        FINDAVCOST2(F,Current);

                        Count:=Count+1;

                        if BigDif>ee then
                        begin
                            BigDif:=0;
                        end else begin
                            ITER2:=ITER2+1;
                            h:=hashstore[0];
                            writeln(change,'Changing rejection of Single');
                            writeln(change,'AvCost[0]bb = ', h^.value);
                        end;   {if}
                    end;    {while}

      until (ITERb>3) or (PolicySingle);

      repetition:=repetition+1;  Rew1:=1;

        if (repetition=2) then begin  Rew2:=1.9;     end;
        if (repetition=3) then begin  Rew2:=2;       end;
        if (repetition=4) then begin  Rew2:=2.3;     end;

      until (repetition>4);
```

Bibliography

ACKERLEY, R.G.

(1987) Hysterisis-type Behaviour in Networks with Extensive Overflow. *British Telecom Technol. J., 5.*

AKINPELU, J.M.

(1984) The Overload Performance of Engineered Networks with Nonhierarchical and Hierarchical Routing. *Bell System Technical Journal, 63, 126-181.*

ASH, G.R.

(1985) Use of Trunk Status Map for Real-Time DNHR. *Proceedings of the 11th Teletraffic Congress, Kyoto.*

ASH, G.R., CARDWELL, R.H. and MURRAY, R.P.

(1981) Design and Optimisation of Networks with Dynamic Routing. *Bell Systems Technical Journal, Vol.60, 1787-1820.*

BLACKWELL, D.

(1965) Discounted Dynamic Programming. *The Annals Of Mathematical Statistics, Vol.36, 226-235, University of California, Berkeley.*

(1967) Possitive Dynamic Programming. *Proc. Berkeley Symp. Math. Stat. Probability, 5th, 415-418.*

BURDEN, R.L., FAIRES, J.D.

(1989) *Numerical Analysis.* (4th Edition). PWS-KENT Publishing Company,

Boston.

BURMAN, D.Y., LEHOCZKY, J.P. and LIM, Y.

(1984) Insensitivity of blocking probabilities in a circuit- switched Network. *Journal of Applied Probability 21: 850-859.*

CARROLL, L.

(1865) *Alice's Adventures in Wonderland.* Oxford University Press, 1982.

FELLER, W.

An Introduction to Probability theory and its Applications. Vol. I (1966), Vol. II (1968). John Wiley & Sons.

FORSYTH, G.E. and MOLER, C.B.

(1967) *Computer Solution of Linear Algebraic Systems.* Prentice-Hall.

FOSCHINI, G.J., GOPINATH, B. and HAYES, J.F.

(1981) Optimum Allocation of Servers to Two Types of Competing customers. *IEEE Transactions on Communications 29: 1051-1055.*

GIBBENS, R.J.

(1986) Some Aspects Of Dynamic Routing In circuit- switched Telecommunication Networks. *Statistical Laboratory, University of Cambridge.*

GIBBENS, R.J. and KELLY, F.P.

(1990) Dynamic Routing In Fully Connected Networks. *IMA Journal of Mathematical Control and Information, 7, 77-111.*

GIBBENS, R.J., KELLY, F.P. and KEY, P.B.

(1989) Dynamic Alternative Routing - Modelling and Behaviour. *Teletraffic Science for New Cost-Effective Systems, Networks and Services, 12th International Teletraffic Congress, 1019-1025, Amsterdam: Elsevier.*

186

GIRARD, A.

(1985) Blocking Probability of noninteger groups with trunk reservation. *IEEE Transactions on Communications 33:113-120.*

HAGEMAN, L. A., and YOUNG, D. M.

(1981) *Applied Iterative Methods.* Academic Press.

HARDY, G.H. and WRIGHT, E.M.

(1960) *An Introduction to the Theory of Numbers.* Oxford Academic Press, 4th edition.

HARVEY, C. and HILLS, C.R.

(1979) Determining Grades of Service in a Network. *9th International Teletraffic Congress.*

HEYMAN, D.P.

(1985) Asymptotic Marginal Independence in Large Networks of Loss Systems.*Presented at ORSA/TIMS Applied Probability Conference, Williamsburg, Va. January. Bell Communications Research, Homdel.*

HOWARD, R.A.

(1960) *Dynamic Programming and Markov Processes.* The M.I.T. Press.

HUNT, P.J. and LAWS, C.N.

(1993) Asymptotically Optimal Loss Network Control. *Mathematics of Operations Research, Vol. 18, No. 4.*

JAGERMAN, D.L.

(1974) Some Properties of the Erlang Loss Formula. *The Bell System Techn. Journal, Vol.53, No.3, 525-551.*

KAHAN

[See Burden and Faires (1989)].

KARLIN, S. and TAYLOR, H.M.

(1975) *A first Course in Stochastic Processes.* Academic Press.

(1981) *A second Course in Stochastic Processes.* Academic Press.

KELLY, F.P.

(1979) *Reversibility and Stochastic Networks.* John Wiley & Sons.

(1985) Stochastic Models of Computer Communications systems. *J. R. Statist. Soc. 47, No.3, 379-395.*

(1986) Blocking Probabilities in large Circuit-Switched Networks. *Adv. Appl. Prob., 18, 473-505.*

(1987) One-dimensional Circuit-Switched Networks. *Ann. Prob. 15, 1166-1179.*

(1988) Routing iIn Circuit-Switched Networks: Optimization, Shadow Prices and Decentralization. *Adv. Appl. Prob., 20, 112-144.*

(1990) Routing and Capacity Allocation in Networks with Trunk Reservation. *Math. Oper. Res., 15, No. 4, 771-793.*

(1991) Loss Networks. *Annals of Applied Probability,1, 319-378.*

(1994) Bounds on the Performance of Dynamic Routing Schemes for Highly Connected Networks. *Math. Oper. Res., 19, No. 1.*

KEY, P.B.

(1987) Optimal Control in Circuit-Switched Networks. *British Telecom Research Laboratories.*

(1988) Markov Decision Processes and Optimal Control in circuit- switched Networks. *Proc. of 5th UK Teletraffic Congress, IEEE Publications.*

(1990) Optimal Control and Trunk Reservation in Loss Networks. *Probability in the Engineering and Information Sciences, 4, 203-242.*

(1994) Some Control Issues in Telecommunication Networks. Due to appear in

Probability, Statistics and Optimization, Ed. by F.P.Kelly, John Wiley & Sons.

KRISHNAN, K.R. and OTT, T.J.

(1986) State Dependent Routing of Telephone Traffic: Theory and Results. *25th IEEE Control and Design Conference, Athens* .

(1989) Forward-Looking Routing: A New State-Dependent Routing Scheme. *Teletraffic Science for New Cost-Effective Systems, Networks and Services, ITC-12, Elsevier Science Publishers, 1026-10320.*

LEMBERSKY, M.R.

(1974) On Maximal Rewards and ϵ-Optimal Policies In Continuous Time Markov Decision Chains. *The Annals of Statistics, vol. 2, No.1, 159-169, Oregon State University.*

LIN, P.M, LEON, B.J. and STEWART, C.R.

(1978) Analysis of Circuit-Switched Networks Employing Originating-Office Control with Spill-Forward. *IEEE Transactions on Communications, Com-26, No.26.*

LIPPMAN, S.A.

(1975) Applying a New Device in the Optimization of Exponential Queueing Systems. *Operations Research, Vol. 23, No.4, 687-710.*

LIPPMAN, S.A. and STIDHAM, J.R.

(1977) Individual versus Social Optimization in Exponential Congestion Systems. *Opeartions Research, Vol. 25, No.2.*

LOUTH, G.M., MITZENMACHER, M. and KELLY, F.P.

(1994) Computational Complexity of Loss Networks. *Theor. Comp. Sci.*

MacPHEE, I. M.

Personal communication.

MITRA, D.

(1985) Probablistic Models and Asymptotic results for Concurrent Processing with Exclusive and Non-Exclusive locks. *SIAM J. Comp., 14, 1030-1051.*

(1987) Asymptotic Analysis And Computational Methods For A Class Of Simple, circuit- switched Networks With Blocking. *Adv. Appl. Prob., 19, 219-239.*

MITRA, D. and WEINBERGER, P.J.

(1984) Probabilistic Models of Database Locking: Solutions, Computational Algorithms and Asymptotics. *J. Assoc. Comp. Mach., 31, 855-878.*

ORTEGA, J.M. and RHEINBOLDT, W.G.

(1970) *Iterative Solution of Numerical Equations in Several Variables.* Academic Press.

ORTEGA. J.M.

(1972) *Numerical Analysis - A Second Course.* Academic Press.

OTT, T.J. and KRISHNAN, K.R.

(1985) State Dependent Routing of Telephone Traffic and the Use of Separable Routing Schemes. *Proc. 11th International Teletraffic Congress, Kyoto, Amsterdam: Elsevier.*

ROSS, S.M.

(1970) *Applied Probability Models with Optimization Applications.* Holden-Day.

(1983) *Introduction to Stochastic Dynamic Programming.* Academic Press.

(1983) *Stochastic Processes.* John Wiley & Sons..

(1985) *Introduction to Probability Models.* Academic Press.

ROSS, K.W. and TSANG, D.

(1988) Optimal Circuit-Switched Access Policies in an ISDN Environment. *IEEE Transactions on Communications 37: 934-940.*

SERFOZO, R.

(1981) Optimal Control of Random Walks, Birth and Death Processes, and Queues. *Adv. Appl. Prob., 13, 61-83.*

(1979) An Equivalence Between Continuous And Discrete Time Markov Decission Processes. *Operations Research, Volume 27, No.3, May-June 616-620.*

SONGHURST, D.J.

(1980) Protection against Traffic Overload in Hierarchical Networks employing Alternative Routing. *Telecommunication networks Planning Symposium, Paris.*

TIJMS, H.C

(1988) *Stochastic Modelling and Analysis: A Computational Approach.* John Wiley & Sons.

WHITT, W.

(1985) Blocking when Service is Required from Several Facilities Simultaneously. *A.T. & T Technical Journal, 1807-1856.*

YOUNG, D.M.

(1971) *Iterative Solution of Large Linear Systems.* Academic Press.

ZACHARY, S.

(1988) Control of Stochastic Networks with Applications. *Journal of the Royal Statistical Society Series B 50: 61-73.*

ZIEDINS, I.B. and KELLY, F.P

(1989) Limit Theorems for Loss Networks with Diverse Routing. *Adv. Appl. Prob., 21, 804-830.*

ZIEDINS, I.B. and MacPHEE, I. M.

In preparation.