An Admission Control Look Ahead Policy in a Loss Network

M. Wingender
University of Twente

January 20, 2007

Contents

Summary 1 *This report consideres an admission control policy in a loss network. The networks nowdays are expected to carry multimedia traffic such as voice only, mixed voice and data, image transmission, WWW browsing, E-mail etc. In order to support such a wide range of traffic the network needs a large capacity to satisfy all the quality of services. Since capacity is scarce, we need an admission control policy with a high revenue rate and simple heuristics. In this report the family of look ahead policies are introduced. The policy have good performance measures in the both the transient period and on the long run. The policies' computational complexity is at best for large capacity systems with a small number of different classes.*

1 Introduction

For many years now traffic lights and more recently traffic circles controls the streams of traffic during the day on very crowded roads. This would not be necessary if there were less traffic or more roads, then there would not be any congestion or traffic jams. But nowadays, without any control at traffic interchanges, there would be complete chaos. The same control is needed in many applications were items arrive at random times and occupying some resource allocation for a random time. And of course only limited recourses are available. Applications in the telecommunication, transportation industry and many more experience the same problems. They need an admission control policy which has the function of limiting the number of traffic flows in a class such that the required Quality of Service (QoS) can be satisfied. Since the capacity is always limited, it is necessary to develop policies that effectively use the network while satisfying the QoS requirements. An admission control policy is needed in order to satisfy these QoS requirements. This means that various traffic should get predictable service from the available resources in the network. Another requirement for an admission control policy is that it has to be capable of handling several classes, since each type of traffic has its own characteristics. The design of admission control policies has important consequences for network performance, since an policy that unnecessarily denies access to traffic flows that could be admitted will under-utilize the recourses, but a policy that incorrectly admits too many flows will induce the QoS violations. Therefore it is very important to develop a good admission control policy applied to the problem.

1.1 Telecommunication Branch

As mentioned before one of the main applications for admission control is the telecommunication branch. Here it is called call admission control (CAC). With the appearance of the fiberglass industry it seemed like there was plenty enough capacity in the telecommunication network, but during the last decade there has been a rapid growth of (wireless) communication technology. The networks are expected to carry multimedia traffic such as voice only, mixed voice and data, image transmission, WWW browsing, E-mail, etc. In order to support such a wide range of traffic over the network, the network should be capable of satisfying various QoS requirements in terms of the call blocking probability and the dropping probability. Of course, capacity is the limiting factor again.

Examples of telecommunication networks are (virtual) circuit-switched networks, mobile cellular systems or content delivery networks (CDN). In circuit-switched networks, an access link with certain amount of bandwidth needs to accommodate different service types including data, audio, video, each with heterogeneous bandwidth requirement and traffic statistics. The goal of CAC is to maximize the throughput of the link for example. In mobile cellular systems, at each cell, a base station with a limited number of channels needs to handle both new calls and handoff calls. Normally handoff calls have higher priority over new calls because one does not want to terminate an ongoing call. The

goal of the CAC policy is to minimize the new and handoff call blocking probabilities while giving priority to handoff calls for example. In multimedia content delivery networks, a CDN server with a certain amount of streaming bandwidth needs to serve user requests for different types of multimedia objects that have different session times and streaming bandwidth requirements. The CDN server collects a revenue from each accepted server. The goal of the CAC policy is to maximize the revenue rate of the CDN server for example.

1.2 Overview

Like the CDN server, the goal in this report is also to maximize the revenue rate in a telephone network with different types of classes. To develop a good admission control policy this report is build up with the following sections. The first section of this report discusses the well-known information about this topic. It tells us what kinds of models are used in common and what the achievements are. It appears that a lot of articles make some assumptions on the problem in practice to develop an optimal policy. All the variables and assumptions are list up in the second section. Next the decision model is intensively discussed. It explains the main problem of developing a good admission control policy and investigates all the pros and cons. This section ends with the problem formulation. Section five starts with mentioning three well-known classic policies, the *threshold, complete partitioning* and the *reservation* policy and the policy from *Iyengar & Sigman*, see [22]. Then the *penalty* policy and *look ahead* policy are discussed. The penalty policy is not simulated, because a part of the calculation was very hard to determine and therefore it is not simulated. But since it took a lot of effort to develop the policy, it is discussed in section five. The end of section five discusses how the optimal policy should look like, but never be useful in practice. Next we analyze the *look ahead* policy and try to find the bounds on the revenue rate. In section seven the computational complexity of the simulated policies are investigated of the policies. Section eight starts with the discussion about the transient period and explains the choice of the length of the transient period. Next it shows the results of the total reward in the transient period and the long-run revenue rates. Section eight ends with a deeper insight of the differences in the long-run revenue rates and an overview of all results. Subsequently the conclusion and some recommendations are discussed. This report ends with the appendices and the references.

2 Literature

The access control problem where a decision-rule decides which calls are send over the network with arbitrary arrivals and departures is called the stochastic knapsack problem. One can find many policies in the literature which tries to solve this problem for different kinds of interest. Some are maximizing the reward or the reward rate where others are minimizing the blocking probabilities of high priority classes or some other QoS measurements. Most papers model the problem as a Semi Markov Decision Problem (SMDP). In [3], [37], [25], [26], [40] and [42] they posed the access control problem as a SMDP. A Markov Decision Process (MDP) is a discrete time stochastic control process characterized by a set of states, actions, and a state transition function (usually a transition probability matrix for discrete state- and action-spaces). An MDP also possesses the Markov property. This means that if the current state of the MDP at time t is known, transitions to a new state at time $t + 1$ are independent of all previous states. A semi-Markov process is a stochastic process that, when it enters state i, spends a random time having some distribution depending on state i before making a transition. For a SMDP the Markov property holds in discrete time. In [48] they apply both the linear programming (LP) and value-iteration algorithms of SMDP's to the network access problem for maximizing utilization, and for maximizing utilization subject to constraints on the probability of blocking of certain classes. The theory of SMDP is used to construct an optimal CAC policy for wireless cellular networks (WCN) that supports multimedia services, see [37]. The multi-class guard channel CAC policy is formulated as a SMDP with constraints on the dropping probabilities of multimedia handoff calls. The optimal multi-class guard channel policy decisions are obtained by applying SMDP linear programming formulation. The optimal CAC decisions for each state are found by solving the linear programming formulation with the objectives of maximizing the system utilization and guaranteeing QoS of handoff calls. In [3] they showed also that a linear programming method for solving the SMDP is employed to find out the optimal CAC decisions for each state. A dynamic call admission control scheme is proposed for voice calls in cellular mobile communication networks and modeled as a SMDP [40]. They claimed that this policy significantly reduces the dropping probability of handoff calls, while it marginally increases the blocking probability of new calls and suppresses the channel utilization. The admission control problem is modelled as a continuous time Markov decision process with an expected total discount reward criterion, see [42], and an equivalent model is developed in discrete time with an undiscounted expected reward objective. From this the existence of an optimal monotone policy is established. More papers do not try to maximize the reward rate, but show monotonicity results. These articles like [1],[46],[29] proof with event based dynamic programming that classes can be ordered, which means that: if it is optimal to accept a class, then to accept a more profitable class is optimal too. These results are also used in this report to develop the policy.

The optimal admission control policy for all kinds of interest faces the "curse

of dimensionality", because the number of states grows exponentially with the number of classes and the available capacity. This will be explained later, see section 5.8. In the literature all models make some assumptions to deal with the numerical overflow problem. Below is a list of the most common assumptions to find a solution. Every assumption will be discussed on the basis of the accompanying papers. So to reduce the computational complexity some papers

- give in on the number of classes [49],[3],[43]
- reduces the word optimal to near-optimal in order to model the optimality problem [39],[22]
- do not find the optimal reward, but decreases the computational complexity by aiming at some QoS measures [44],[37],[4],[40]
- scale the system parameters to the limiting regime [24]
- claim to know the service times of the arriving call [38],[2] or solve the problem with fractional costumers [16],[12].

There are more assumptions that reduce the computational complexity, but these assumptions have very specific properties. The above assumptions are now discussed in further details.

Only Two Types of Classes When the objects arrive and depart at random times the problem is a stochastic knapsack problem. In [49] they showed that when there are only two types of classes, the object volumes are integer multiples of each other and the inter-arrival times are exponential (i.e. Poisson arrivals), the optimal policy is one of a threshold-type. Moreover, the sojourn times are permitted to have an arbitrary distribution. They only looked at coordinate convex policies. They distinguish two types of *threshold* policies. There are threshold type-k policies and double threshold-type policies. A threshold type-1 policy limits the number of class-1 object permitted in the system and always accepts a class-2 object when sufficient volume exists. This is similar to a threshold type-2 policy. A policy is said to be a double-threshold policy when it limits both classes of objects.

For the same system assumptions as above but now with random revenues, there are also conditions where preferred jobs exist from either class, see [43].

Near Optimal Reward Another way to reduce the computational complexity is to search the maximize reward rate in a limited number of policies, because the number of structured policies grows exponentially with the number of classes and the capacity of the system. Two types of policies which are intensively investigated are the *threshold* and the *reservation* policy, see [39]. The search algorithms for finding the optimal parameters of the *threshold* and *reservation* policy are very quick and they work for large capacity systems and many call classes. The rewards that these policies gain are near optimal and outperform

the structured policy with parameters chosen based on simple heuristics. Another policy which acceptance rule included multi-classes but is not optimal is the penalty-function-based admission control policy in [22].

Limiting Regimes The optimal steady state reward can be determined by making an strong assumption on the system parameters. The arrival rate tends to infinity for each class [24] (i.e. the inter-arrival times tends to zero) and the sizes and the reward rates of each class tend to zero in the same order. In this manner the total workload for each class $b_i\rho_i$ and the total reward rate $r_i\rho_i$ remain the same.

Aim at other QoS At call-level, the two important QoS parameters are the call blocking probability and the call dropping probability. An active mobile user in a cellular network may move from one cell to another. The continuity of service to the mobile user in the new cell requires a successful handoff from the previous cell to the new cell. During the life of a call, a mobile user may cross several cell boundaries and may require several successful handoffs. Failure to get a successful handoff at any cell in the path forces the network to drop the call. Since dropping a call in progress has a more negative impact from the user perspective, handoff calls are given higher priority than new calls in accessing the wireless resources. This preferential treatment of handoffs increases the probability of blocking new calls and may degrade the bandwidth utilization. The most popular approach to prioritize handoff calls over new calls is by reserving a portion of available bandwidth in each cell to be used exclusively for handoffs. Based on this idea, a number of call admission control (CAC) schemes have been proposed which basically differ from each other in the way they calculate the reservation threshold [5],[39],[43],[49]. The generalization to multi-classes, where are N classes of traffic and each having a different level of QoS needs a multi-threshold guard channel [4],[7].

Minimizing the blocking probability instead of maximizing the average reward is the goal in [44]. Suppose there is a two class system in which class one calls are more important than class two calls. A system with a guard channel is considered, which is whenever the channel utilization exceeds a certain threshold it always rejects calls of the lower class and accepts the main prior class. It is shown that there is an optimal threshold in which the blocking probability of the class two calls is minimized subject to the constraint on the dropping probability of the class one calls. In order to have more control on both the blocking probability of class one calls and class two calls, limited fractional guard channel policy is proposed. They show that analyzing the problem as a continuous time Markov chain the optimal control policy for the MINOBJ is of the threshold type. MINOBJ: minimizing a linear objective function of the two blocking probabilities. Then they find an algorithm for the MINBLOCK problem. MINBLOCK: for a given number of channels, minimizing the new call blocking probability subject to a hard constraint on the handoff blocking probability. Finally they find an algorithm for the MINC. MINC: minimizing

the number of channels subject to hard constraints on the new and handoff call blocking probabilities.

3 Preliminarily

In this section one can find all the needed variables and parameters in table **1**. Every variable or parameter will be explained the first time it occurs in this report.

3.1 System Variables and Parameters

C	maximum capacity of the network
D_i	minimum capacity for blocking a class $i \in \{1, 2, .., m\}$ call
C_i	minimum capacity where the policy may reject a class $i \in \{2, .., m\}$ call
m	number of different classes, $m \in \mathbb{N}$
X_i	stochastic variable of the inter-arrival time of class $i \in \{1, 2, .., m\}$
$X_{\min}$	stochastic variable of the minimum inter-arrival time of all classes
Y_i	stochastic variable of the service time of class $i \in \{1, 2, .., m\}$
$Y_{\min}$	stochastic variable of the minimum service time of all classes
λ_i	inter-arrival rate of class $i \in \{1, 2, .., m\}$
μ_i	service rate of class $i \in \{1, 2, .., m\}$
$\lambda^{\mathbf{m}}$	$\lambda_1 + \lambda_2 + .. + \lambda_m = \sum_{i=1}^{m} \lambda_i,$
$\mu^{\mathbf{m}}$	$\mu_1 + \mu_2 + .. + \mu_m = \sum_{i=1}^{m} \mu_i$
EA_i	mean inter-arrival time of class $i \in \{1, 2, .., m\}$
EB_i	mean service time of class $i \in \{1, 2, .., m\}$
b_i	the size of class i call which occupies b_i of the capacity
r_i	reward that is gained when class $i \in \{1, 2, .., m\}$call is accepted
n_i	number of class i calls in the system, $n_i \in \mathbb{N}$
$\mathbf{n}$	the system state with $\mathbf{n} = (n_1, n_2, .., n_m)$
$d_i(\mathbf{n})$	decision parameter for class i in state $\mathbf{n}$ with $d_i = 1$ (accept) or $d_i = 0$ (reject)
$N_i(t)$	number of units capacity that has become available from class i calls in $[0, t]$
$F_Q(t)$	cumulative distribution function of the stochastic variable Q
$f_q(t)$	probability density function of the stochastic variable Q
T	critical time period
$ER_{\mathbf{n}}^{\zeta}$	the expected reward in state $\mathbf{n}$ when the classes $\{1, 2, .., \zeta\}$ are active
$ET_{\mathbf{n}}^{\zeta}$	the expected time period in state $\mathbf{n}$ when the classes $\{1, 2, .., \zeta\}$ are active
$P(BL_i^{\mathbf{n}}, T)$	probability that class $i \in (1, 2, .., m)$ call in state $\mathbf{n}$ is blocked upon arrival in a time period T
$P(TBL_j^{\mathbf{n}}, T)$	total probability that in state $\mathbf{n}$ the system blocks any of the classes $\{1, 2, .., j\}$ upon arrival in a time period T
a_k, T	the event that a class $k \in (1, 2, .., m)$ call arrives in a time period T
Ω	all the possible states, $\Omega = \{\mathbf{n} : \mathbf{n} \cdot \mathbf{b} \leq C\}$
δ	number of steps to look ahead for the *look ahead policy*
ζ	least profitable active class $\{1, 2, .., \zeta\}$
θ	scale parameter

(**1**)

3.2 System Assumptions

- the inter-arrival times are exponential and independent for each class
- the service times are exponential and independent for each class and the number of calls in the system
- the number of calls in the system when the capacity constraint is almost violated is large enough. This means that after a call is leaving the system, the rate of departures is unchanged
- the capacity constraint can never be violated, $\sum_{i=1}^{m} b_i n_i \leq C$
- reject means not accepted because of the decision
- blocked means not accepted because of the capacity constraint

4 Decision Model

4.1 Main Idea

A good admission control policy could be judged on two topics. First, the goal of a good admission control policy should approach the goal of the optimal admission control policy for the same problem. Second, the computational complexity of the construction of the policy should be polynomial. The construction of the policy consists in general of the calculation of the decision parameters. In many applications the goal is to gain as much profit as possible on the long run or in a fixed time period. In mathematical terms this is the same as maximizing the total expected reward or the reward rate. To do so, the approach in this report is based on minimizing the rejections of the more profitable classes in the system by looking only a few events ahead or a fixed time period. Looking more events ahead would give better results, but also a higher computational complexity. Therefore the main idea is to look only a few events ahead in order to have a near optimal policy with a low computational complexity. There are many policies known that are based on this approach, for example the *threshold* policies or the trunk *reservation* policies. But they only make there decision based on the capacity that is left on the moment of an arrival. So it does not really matter if there is a high or low probability that in a short moment of time more capacity becomes available. One can imagine that if the system has a lot of calls that has a large size but small expected service durations, then there will be more capacity available in a shorter time period then if the system is filled with calls which has a small size and long expected service durations. To understand the main idea of minimizing the blocking-probabilities of the more profitable classes look at figure 1.

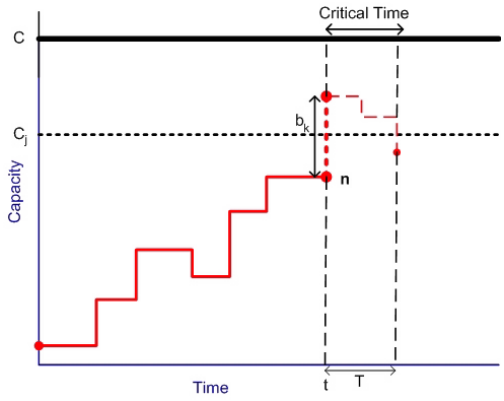

Figure 1: sample path of the *look ahead* policy

This picture shows a realization of a telephone network with maximum capacity C. The red line indicates the total capacity that is used. At time t the

system in a state $\mathbf{n}$ and a class k call arrives with size b_k. Now one have to decide to accept this class k call and gains r_k amount of profit or to reject this call and gains nothing. The disadvantage of accepting this class k call is that the system has not enough capacity available for of a more profitable class j. One can see this in figure 1 as the interrupted black line at high C_j. The size of a class j call is $b_j = C - C_j$. When such a class j arrives within the time that the system has not enough capacity, then this call has to be rejected because the capacity constraint C is violated.

So in order to avoid this event, one could always reject such a class k call if the system has not enough capacity left for a more profitable class. But if the blocking-probability of a more profitable class j is very small, then accepting this class k call is more profitable than blocking this class k call. The probability that a class j call is blocked depends on the available capacity, the size of class j, the total departure rate and the class j arrival rate. The admission control policy accepts an arriving call if the expected reward rate is higher than the expected reward when the policy rejects the arriving call. The expected reward rate in the latter can be calculated when the policy pretends to reject the arriving call and waits for a more profitable class to accept. So we make the decision to accept or reject based on the blocking-probability of the more profitable classes. In this way the policy is conditioning on the events in the future.

The first policy that is developed is based on the difference between the profit of accepting an arrival and the losses caused by the accepted arrival in a certain time period. From now on this is the penalty policy. The profit of accepting an arrival is very easy to calculate, since the policy has accepted this arrival and only takes this call into account. The losses have to be predicted and are much harder to calculate. A call is blocked or lost if it arrives when the system has not enough capacity available. We could calculate the expected losses in a finite or infinite time period. It is logical to calculate the losses in an infinite time period because in theory an arrival can be blocked in any moment of time. But the calculations could be much faster when the time period is finite. Suppose this time period is called the critical time period. An example of a fixed critical time period which is chosen based on the parameters of the system is shown in figure 1. This critical time period can be determined by the extra capacity that is needed divided by the total departure rate times the capacity per class call. Equation (1) shows the formula of this critical time

$$T = \frac{(\mathbf{n} + \mathbf{e}_k)\cdot\mathbf{b} - C_j}{\sum_{k=1}^{m} \mu_k n_k b_k} \tag{1}$$

where $(\mathbf{n} + \mathbf{e}_k)\cdot\mathbf{b}$ is the dot product of the number of calls in the system (inclusive the accepted class k call) with the capacity per class. In the next section the policy is discussed in further details.

The decisions in the second policy that is developed are based on maximizing the expected reward looking a few events ahead at the decision epoch. The policy pretends that the next arrival is accepted or rejected and determines the expected reward rate based on the assumptions at the decision epoch.

In section 2 some monotonicity results are discussed. A class with a higher reward is more profitable than a class with a lower reward, both with equal service durations and sizes. Here is assumed that the reward is gained instantaneous at the moment the call is in the system. The *look ahead* policy also uses the instantaneous rewards in their decision. But in practical it is more common that the customer pays per unit time (seconds). In this case a class with a higher reward rate is more profitable than a class with a lower reward rate, both with equal sizes. In the simulations the total reward is used to compare the policies with each other. This is the summation of the reward that is gained per unit time. On the long run both methods generate the same total reward.

To order the classes we need to know if the reward is given as the reward rate or as an instantaneous reward. In the latter the most profitable class is the class for which i in equation (2) holds that

$$\max_{i=1,..,m} \frac{r_i \mu_i}{b_i} \tag{2}$$

which could be considered as the class with highest reward per unit time per capacity. The second most profitable class is of course the same formula as above, but now without the most profitable class.

If the reward rate is given the most profitable class is

$$\max_{i=1,..,m} \frac{r_i}{b_i} \tag{3}$$

which has the same meaning as before.

4.2 Transient Period

In most papers the main and only goal is to maximize the long-run revenue rate. But in almost all applications the problem of making the optimal decisions is over a fixed time period. Therefore the start up of the system, also known as the transient period, is also important. When the simulation period is relatively short, i.e. the total load of capacity fits in the total capacity of the system with high probability, the *accept all* strategy would be optimal. Because the probability of blocking a class is very low and therefore it is a sin to reject any call. In case of a relatively short simulation period the transient period is more important than the reward rate.

For problems where the period is relatively large, the *accept all* strategy is in general not optimal. On the other extreme, the *Kelly* policy rejects already calls in the very beginning of the simulation period. It aims at the optimal average class ratio. The class ratio is the ratio of the number of calls of all types of classes. Therefore on the long run it catch up with the *accept all* policy when both use the same amount of capacity, but it has a very slow start, see figure 2.

The main idea in the transient period of the *look ahead* policies is to compromise with both extremes, see figure 2. As one can see, the rejection of calls increases when the available capacity becomes less. The next section discusses this idea in further details.

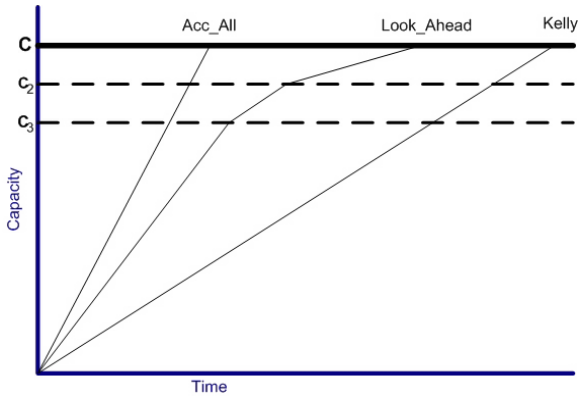

Figure 2: average occupation rate of the *accept all* policy, *look ahead* policy and the *Kelly* policy over time

As mentioned before, the policy that accepts all classes independent of the system state is called the *accept all* policy. When more or equal capacity is needed ($\rho_i \cdot b_i$) than the total capacity (C), some arrivals are blocked because not enough capacity is available. A good policy tries to avoid that the most profitable classes are blocked. The probability that a more profitable class will be blocked can be decreased by saving some capacity for the more profitable classes. The question is: *how much capacity should the policy reserve and for which classes.* Or in other words: *at what level should the policy rejects the less profitable classes in order to reserve some capacity for the more profitable classes.* Because different types of classes have normally different sizes, a good policy reserves a different amount of capacity of every class. A critical transition is denoted as a transition from state $\mathbf{n} \rightarrow \mathbf{n}+e_i$ (acceptance of class i call) where in state $\mathbf{n}+e_i$ is a possibility is that a more profitable class will be blocked. A state is called a critical state when at least one critical transition is possible. The set of all critical states is

$$\Omega^* = \left\{ \mathbf{n} \in \Omega | (\mathbf{n}+\mathbf{e}_i) \cdot \mathbf{b} > C - \delta \max_{j=1..i-1} b_j \right\} \text{ for some } i \in 2,.,m \qquad (4)$$

In case that δ is one, then all the critical states have a more than zero probability that the first more profitable class which arrives after the decision epoch is blocked in a certain time period after accepting a (less profitable) call. In case that δ is m then all the critical states have a more than zero probability that the m-th call which arrives after the decision epoch is blocked in a certain time period after accepting a (less profitable) call.

Since is it not fixed how many steps the policy should look ahead in order to get a (near) optimal policy, many levels are simulated. The calculations are faster if the critical level is such that the policy reserves some capacity for only one more profitable class ($\delta = 1$) than for two ($\delta = 2$). But there are

disadvantages of reserving capacity for only one more profitable class. If for example after the moment that a class k call is accepted two more profitable classes arrive with a very short inter-arrival time, then the second one could be rejected. So in this case it was better to reject the class k call. This disadvantage could be avoided if the policy looks two arrivals ahead. But than again, it could be that three more profitable classes arrive and the third should be rejected. But the more steps the policy looks ahead the higher the computational complexity. This balancing between a better policy and fast decisions is the main problem for all policies.

Another decision is how important are events in future for the decision that has to be made. In other words what should be the length of the time period in which the policy have to make its decision? The impact of accepting and rejecting a call is visible till the system is in the same state again. It could take a very long time to happen. Therefore the computational complexity is very high if the policy wants to calculate the reward rate till the system state is in the same state again as at the moment of the decision epoch. To avoid this, the policy should reduce the time period which it looks ahead. The policy needs a good time period in order to keep the calculations fast but the information valid about the difference in reward rate for both decisions. The information could not be completely valid when the policy looks one event ahead and reserves for two more profitable classes, because there is a possibility that two profitable classes arrive consecutively close. Therefore it should compromise with the validity of the information on the reward rates and the computational complexity. In this report the policy is simulated for $\delta = \{1, .., 5\}$.

4.3 Problem Formulation

In this report the calculations are done by looking only one arrival ahead after the decision epoch, but there are different sets of critical levels simulated. The main goal of this report is to develop a policy which has a good balance between generating the (near) optimal reward and having a reasonable computational complexity. Therefore the problem formulation for this report becomes

Problem 2 *Construct an admission control policy which generates near optimal rewards in both transient and steady state period with a reasonable computational complexity.*

5 Policies

In this section the most important policies are described. The section starts with the four policies which already exist. We shortly mention the policies and describing how the policies make their decision. The first two are from the family of the structured call admission control policies. These are the well-known *threshold* and *reservation* policies. These policies are easy to implement and generate pretty good revenues in practice. Next are two policies which are also been simulated and compared with other policies, namely the policy from *Kelly* and the policy from *Iyengar and Sigman.* This sections ends with the *penalty* policy and the *look ahead* policies. The *look ahead* policies are also simulated and compared with the previous two policies.

5.1 Threshold Policy

The *threshold* policy shares the capacity among all classes. This type of policy is called a sharing policy. The *accept all* policy is also a complete sharing policy, where all the capacity is shared with no restrictions. The *threshold* policy shares also the capacity but restricts the number of calls of each class to a certain threshold.

For an admission control problem with only two types of classes the optimal *threshold* policy always accepts one type of class and the other one is accepted up to a threshold. When the optimization is carried out over the class of coordinate convex policies, then the optimal policy is one of a threshold type [49]. In case of more than two classes, the optimal policy is in general not one of a threshold type. For an admission control problem with m types of classes, each class k is associated with a threshold parameter t_k. A class k call is accepted upon arrival if and only if there are sufficient resources and the number of class k calls in the system does not exceed t_k after acceptance. Under a *threshold* policy with parameters $t_1, t_2, .., t_m$, the policy accepts the class k call if and only if the class k call does not violate the capacity constraint and the number of class k calls in the system is below the threshold t_k, see equation (5).

$$d_k(\mathbf{n}) = 1 \Leftrightarrow (\mathbf{n} \cdot \mathbf{b} + b_k \leq C) \cap (n_k + 1 \leq t_k) \tag{5}$$

where $d_k(\mathbf{n}) = 1$ means that the system accepts the class k call arrival when the system is in state $\mathbf{n} \in (n_1, .., n_m)$ and C is the total capacity. In order to simulate the *threshold* policy we use the algorithm of [39] to find the parameters $t_1, t_2, .., t_m$. Once the parameters are determined the policy is fixed and can be simulated.

5.2 Complete Partitioning Policy

Unlike the *threshold* policy which is a sharing policy, the *complete partitioning* policy is not a sharing policy. The *complete partitioning* policy allocates a fixed amount of capacity to each class. Denote s_k, $k = \{1, .., m\}$, as the number of volume units allocated to class k calls. Any allocation $\mathbf{s} = (s_1, ..., s_k)$ must

satisfy $s_1 + s_2 + .. + s_m = C$. A given allocation $\mathbf{s}$ uncouples the capacity into m smaller capacities where capacity k has s_k volume units fully dedicated to class k calls. If a class k call in state $\mathbf{n}$ arrives then for the decision parameter $d_k(\mathbf{n})$ under a *complete partitioning* policy with parameters $s_1, s_2, .., s_m$ holds

$$d_k(\mathbf{n}) = 1 \Leftrightarrow (n_k + 1)b_k \leq s_k \tag{6}$$

where n_k denotes the number of calls in the system of class k upon arrival.

5.3 Reservation Policy

Under a *reservation* policy, each class k is associated with a reservation parameter s_k. A class k call is accepted upon arrival if and only if the system has s_k or more free resources (reserved for other classes) after acceptance. The *reservation* policies are also known as trunk reservation policy in the literature of telephone or circuit-switched networks and guard channel policy in the literature of mobile cellular systems [44].

For homogeneous cases in which all classes have the same size $b = b_k$ and mean service time $\mu = \mu_k$, the optimal *reservation* policy is always better than the optimal *threshold* policy [1]. But in our problem not all the sizes are the same nor the mean service times are equal. Therefore both policies will be simulated. Under a *reservation* policy with parameters $s_1, s_2, .., s_m$ the policy accepts the class k call if and only if the used capacity plus the size of class k is at most the total capacity minus s_k, see equation 7.

$$d_k(\mathbf{n}) = 1 \Leftrightarrow \mathbf{n} \cdot \mathbf{b} + b_k \leq C - s_k \tag{7}$$

where the system upon arrival is in state $\mathbf{n} \in (n_1, .., n_m)$ and C is the total capacity. In order to simulate the *threshold* policy we use the algorithm of [39] to find the parameters $s_1, s_2, .., s_m$. Once the parameters are determined the policy is fixed and can be simulated.

5.4 Kelly

The policy of *Kelly* is also very simple to describe. It always accepts the most profitable classes and always rejects the least profitable classes. Only one type of class is accepted with a certain probability. In this manner the policy generates the maximum expected reward rate on the long run when the scale parameter θ goes to infinity. Then the arrival intensity for every class tends to infinity, but the amount of work remains the same and the reward per capacity remains also the same. The new system parameters are

$$\begin{aligned} \lambda_i^\theta &= \theta\lambda_i \\ b_i^\theta &= \frac{b_i}{\theta} \\ r_i^\theta &= \frac{r_i}{\theta} \end{aligned} \tag{8}$$

where also the b_i^θ could remain constant and then the capacity $C^\theta = \theta C$. This policy performs very well in the steady state period but not in the transient period. Equation (9) determines which are accepted and rejected and which class is accepted with a certain probability.

$$\begin{aligned} & \max \sum_{i=1}^{m} r_i \rho_i d_i \qquad (9) \\ \text{s.t.} \sum b_i \rho_i d_i &\leq b \\ 0 &\leq d_i \leq 1, \text{ for } i = 1, .., m \end{aligned}$$

The $d_i's$ denote which classes are accepted and rejected. If $d_i = 1$ then this class i is always accepted and if $d_i = 0$ then this class i is always rejected. In general, there is one d_i that is between zero and one. This number is also the fraction of calls which are accepted. The idea is that the most profitable class is always accepted as longs as it does not violate the capacity constraint. Then they look for the second best class and do the same. Till for one class not all the load can be accepted on the long run. Therefore only a percentage of the requests of the extra class which violates the capacity constraint have access. All the classes which are less profitable are always rejected.

5.5 Iyengar & Sigman

This policy extends the information from the previous results from equation (9) and the paper from Kelly [24]. They introduce penalty-function-based admission control policies to approximately maximize the expected reward rate in a loss network. The exponential penalty function (10) penalizes deviations from the desired target set. Thinning policies perform well in steady state but under-utilizes in the transient period, but the penalty policy does not suffer from these drawbacks. They claim that the use of this policy leads to an expected reward rate which is close to the expected reward rate of [24], when the requests sizes are small compared to the total capacity. They introduce an alternate system, where each request which is blocked goes into service in the alternate system. In this way, the penalty function depends on the requests in the normal system and the amount of requests in the alternate system. The policy tries to limit the accepted requests. They accept a request if the costs of accepting a request are lower than blocking the request and if the capacity constraint is not violated (11). The policy is a threshold-type policy in the expanded state space, because an alternate system has added. The penalty function is

$$\Psi(\mathbf{s}) = \sum_{i=1}^{m} \underbrace{\left[\exp(\beta \frac{b_i x_i}{c_i^0}) + \exp(\beta \frac{b_i y_i}{c_i^1}) \right]}_{\Psi_i(\mathbf{s}_i)} \qquad (10)$$

where $\mathbf{s} = (\mathbf{x}, \mathbf{y})$,the capacities (c^0, c^1) and β come from (12) and (13). The vector $\mathbf{x} = (x_1, x_2, ..x_m)$ describes the state of the accepted requests and the

vector $\mathbf{y} = (y_1, y_2, .., y_m)$ describes the state of all requests that have been rejected. The decision rule for the penalty control policy is for class i

$$d_i(\mathbf{s}) = 1 \Leftrightarrow \left(\frac{\partial \Psi_i(\mathbf{s}_i)}{\partial x_i} \leq \frac{\partial \Psi_i(\mathbf{s}_i)}{\partial y_i} \right) \cap \left(\sum_{j=1}^{m} b_j x_j + b_i \leq b \right) \tag{11}$$

The latter of the union of (11) means that the capacity constraint can not be violated. An ϵ-perturbation of the steady state LP is defined as follows

$$\begin{aligned} & \max \sum_{i=1}^{m} r_i \rho_i d_i \\ s.t. \sum b_i \rho_i d_i &\leq \frac{b}{1+4\epsilon} \\ 0 &\leq d_i \leq 1, \quad i = 1, .., m \end{aligned} \tag{12}$$

where $\boldsymbol{\alpha}^{\epsilon}$ is the solution of (12). The bound on β must satisfy

$$\beta \leq \epsilon(1+4\epsilon) \min \left(\min_{i:1 \leq i \leq m} \{d_i^{\epsilon} \rho_i\}, \min_{i:i \in U_{\epsilon}^c} \{1 - d_i^{\epsilon}) \rho_i\} \right) \tag{13}$$

where $U_{\epsilon}^c = \{i : d_i^{\epsilon} < 1, i = 1, .., m\}$ and they have dropped all classes with $d_i^* = 0$.

5.6 Penalty Policy

A very simple and logic way the make the decision rule is to compare the profit of accepting an arrival with the losses caused by the accepted arrival in a certain time period. After an acceptance, the available capacity decreases and the probability that the first call will be blocked could increase, especially when the system has almost used all the capacity. When the size of class call is bigger the available capacity, this call is lost and more important is that the policy misses the reward of this call. The higher the reward that the policy misses, the more important is it to avoid. The decision of the *penalty* policy is based on the lost losses caused by the acceptance of an arrival. Of course the reward that is gained by accepting this arrival is also important. In other words, after every acceptance there is a risk that another call is blocked because the capacity constraint is violated. The question reads: is it worth to take that risk? The reward that is lost is determined as the reward of the first arrival after the decision epoch caused by the acceptance of the arrival. This means that it is important whether the system is already in a state where for some classes the first arrival could be blocked. Because these reward losses in state $\mathbf{n}$ must be extracted from the reward losses in state $\mathbf{n} + e_k$.

Hence, if the instantaneous reward is more than the average reward that is lost, the policy accepts the arrival and if the average reward losses are more than the instantaneous reward then the policy rejects the arrival. The policy

does not need to order the classes, because any class is part of the decision at any time. The instantaneous reward of class k call is of course

$$ER = r_k \tag{14}$$

The average reward that is lost in a time period T after the acceptance of a class k arrival in state $\mathbf{n}$ becomes

$$ELost_k^{\mathbf{n}} = \sum_{i=1}^{m} \left[P(TBL_{k-1}^{\mathbf{n}+e_k}, T) - P(TBL_{k-1}^{\mathbf{n}}, T)\right] \frac{\lambda_i}{\sum_{i=1}^{m} \lambda_i} r_i \tag{15}$$

where $P(TBL_{k-1}^{\mathbf{n}})$ is the total blocking probability that the first call of any the classes $\{1, 2.., k-1\}$ is blocked in state $\mathbf{n}$.This can be determined with the general rule of sum. Using the "*or*" statement the total blocking probability is

$$P(TBL_k^{\mathbf{n}}, T) = P(BL_1^{\mathbf{n}}, T) \vee P(BL_2^{\mathbf{n}}, T) \vee .. \vee P(BL_{k-1}^{\mathbf{n}}, T) \tag{16}$$

where the probability that a class j call is blocked in a time period T in state $\mathbf{n}$ is

$$P(BL_j^{\mathbf{n}}, T) = 1\text{-}\exp(-\lambda_j T)\text{-}\sum_{l_1=0}^{\infty} .. \sum_{l_m=\left\lceil \frac{b_j-(C-\mathbf{n}\cdot\mathbf{b}_j)-b_1 l_1-..-b_{m-1}l_{m-1}}{b_m} \right\rceil}^{\infty} \frac{\lambda_j (\mu_1)^{l_1} .. (\mu_m)^{l_m}}{l_1!..l_m!}$$
$$\left(\frac{(l_1 + .. + l_m)! - \sum_{i=0}^{l_1+..+l_m} \frac{(l_1+..+l_m)!}{i!} \exp(-T(\mu^{\mathbf{m}} + \lambda_j))(T(\mu^{\mathbf{m}} + \lambda_j))^i}{(\mu^{\mathbf{m}} + \lambda_j)^{l_1+..+l_m+1}} \right) \tag{17}$$

For further details we refer to Appendix 11.1 equation (92), where the blocking probability for equal and unequal sizes is extensively discussed.

Hence the overall strategy using this policy is

$$d_k(\mathbf{n}) = 1 \Leftrightarrow ER \geq ELost_k^{\mathbf{n}} \tag{18}$$

5.7 Look Ahead Policy

In this section the family of *look ahead* policies will be discussed. The section starts with the *one step look* policy. Next, the *two steps look ahead* policy will be mentioned and the section ends the δ *steps look ahead* policies. The δ indicates the number the steps which the policy looks ahead.

5.7.1 One Step Look Ahead

In the previous section the decision was based on the expected reward losses caused by the accepted arrival and the instantaneous reward of the accepted arrival. In this section the decision is based on looking a few events ahead. Later

this section the meaning of "a few events ahead" will be intensively discussed. But from now on consider "a few events ahead" as an unspecified period. The expected reward rate for both decisions is calculated in the time period from the decision epoch till a few events ahead. The expected reward rate is the expected total reward divided by the expected time period. The total reward can be calculated by adding all the rewards that are gained by the policy, i.e. all the accepted arrivals (or calls which are not blocked or rejected). The time period is from the decision epoch until all accepted arrivals are out of the system. Which arrivals are accepted will be specified later. Since the policy makes the decision based on looking a few events ahead, the decisions are not necessary optimal. The optimal decision is the one that maximizes the average reward rate on the long run. To calculate the optimal decision one has dimensionality problems, see section **5.8.**

Because the time periods are defined from the decision epoch until some specified event happened, the time periods in general differ for both decisions. This could be a subject for discussion since it is not unquestioned that a small time period with a high reward rate is better than a long time period with a little reward rate. But the purpose of the time period is that it reflects all advantages and disadvantages of the decision, so the period with the highest reward rate is the best.

The most fair strategy is to compare two sample paths after the decision, but this involves too complex calculations. As mentioned before, the decision is in general based on two different time periods. The reason why the time periods differ is that after the decision epoch not all classes are active. Active classes are denoted as classes which are seen by the system. The policy could accept, reject or block these classes. Thus even when the system has not enough capacity for a class it could be an active class upon arrival. Inactive classes are not taken into account by the system and therefore they could not be accepted, but only rejected. When the policy rejects a class k call at least all the classes from $\{k, ..m\}$ are inactive until enough capacity has become available. When the policy accepts a class k call then only a class call could be active but also all classes $\{1, 2, .., m\}$ could be active. The policy pretends in advance which classes are active and calculated the reward rate. He does so for all the combinations of the ordered active classes. This means that every combination is calculated but if a class k is active then also a class $k-1$ is active. Remember that the class one calls are the most profitable ones and the class m calls are the least profitable ones. There are m different combinations. In case the policy rejects a class k call there are $k-1$ combinations. Therefore the event of the first accepted arrival is different for both situations after the decision epoch.

Also the system state ends different in general for both decisions. To force the policy to look ahead until the system is in the same state again after the decision epoch is too hard to solve. Then all the probabilities of all the states where the system could end have to be known. This are many states, in theory the system could be even empty again (i.e. all call are departed before the first call arrives). Therefore it is to hard too calculate what the reward rates are until the system is in the same state again. In appendix 11.2 a model can be found

where the time period is from the decision epoch till the system is again in a critical state. But to determine the expected reward rate a lot of assumptions are made and therefore this policy is not simulated or analyzed.

As mentioned before, the problem now is to find a good time period in which both advantages and disadvantages become visible after accepting or rejecting an arrival. The disadvantages and advantages of accepting and rejecting an arrival have already been discussed. The advantage of accepting a class k arrival is the guarantee of the reward r_k.The disadvantage is that a more profitable class arrives which could be blocked. The most simple case is where one just calculates the expected reward rate of the first class k arrival, see equation 19.

$$ERR_{acc} = \frac{ER}{ET} = \frac{r_k}{EB_k} \tag{19}$$

But this is not completely fair, because in this time period a more profitable class could be blocked because the capacity constraint is violated. Hence, the time period should be longer to see if the acceptance was a good decision.

The advantage of rejecting a class k call is that it has a higher acceptance-probability of a more profitable class when it arrives. The disadvantage is that it could take a very long time until it arrives. Therefore the time period of the policy after accepting a call is until a next call arrives which should be accepted, even when it is blocked. In the latter the policy is waiting for the next arrival but gains nothing. If the policy accepts the other arrival, then the time period is till both calls are out of the system. The reward of the policy after rejecting a call is the reward from the next arrival which is accepted. The time period is from the decision epoch until the accepted call is out of the system. Now all disadvantages and advantages are exposed in this time period and the average reward rate is fast to calculate.

The last problem is: which classes should be active after the decision epoch? When the policy accepts a class k call, the system has less capacity available and therefore the highest reward rate could be reached by accepting all classes or only the class one calls. Therefore one has to split the problem in m smaller problems. First, the policy determines the expected reward rate by pretending that only the class one calls are active. Then the policy determines the expected reward rate by pretending that the classes $\{1, 2\}$ are active. The policy continues this process till all classes $\{1, 2, .., m\}$ are active. The expected reward rate is the maximum over all m reward rates. In other words the policy pretends if it knows what to do when the second call arrives, but it makes the decision only for the first arrival.

Sample Path The underneath figures 3 and 4 explain the main idea of the policy when the decision is to accept the arriving call. Figure 3 below shows two sample paths with in top the events. At t_0 a class k call arrives and the policy has to make the decision to accept or reject this call. There are four types of classes and ordered alphabetically. The sooner the letter in the alphabet the more profitable the class $\{i, j, k, l\}$.

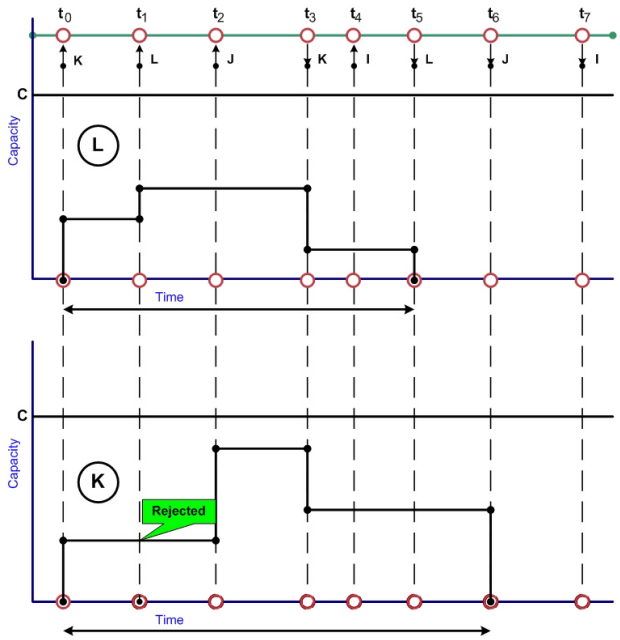

Figure 3: s*ample path: in the above graphic the classes* $\{i, j, l\}$ *are accepted and in the graphic below all classes* $\{i, j, k, l\}$ *are accepted*

The top graphic in figure 3 shows that the class k call is accepted and the big L in the black circle shows that all the classes $\{i, j, k, l\}$ are also accepted on arrival, which in equation (22) means that $\zeta = l$. At t_1 a class l call arrives and is also accepted. The total reward is $r_k + r_l$. The time period is until all accepted calls are out of the system. This is at t_5. In the lower graphic the policy also accepts the class k call but accepts only the classes $\{1, .., k\}$. At time t_1 the class l call is not seen by the system, therefore it has to wait until the next arrival at t_2. At this moment a class j call arrives and is accepted. Now the total reward is $r_k + r_j$ and the time period is till t_6.

In the first sample path in figure 3 the second arrival is accepted, because there was enough capacity available. In figure 4 a sample path is shown as above but now with less available capacity at t_0.

On top the same events occur and in upper graphic at t_0 a class k call is accepted. At t_1 a class l call arrives and would be accepted if enough capacity was available, but in this case the call is blocked. The end of this period is at t_3, the departure of the class k call. In the graphic below is also at t_0 a class k call accepted. But this time at t_1 the arrival of the class l call is not seen by the system and thus it waits till t_2, the arrival of a class j call. Also the class j call is blocked, because there is not enough capacity left. The period ends at t_3. Suppose the class j call was at t_4 arrived and still blocked then the time period

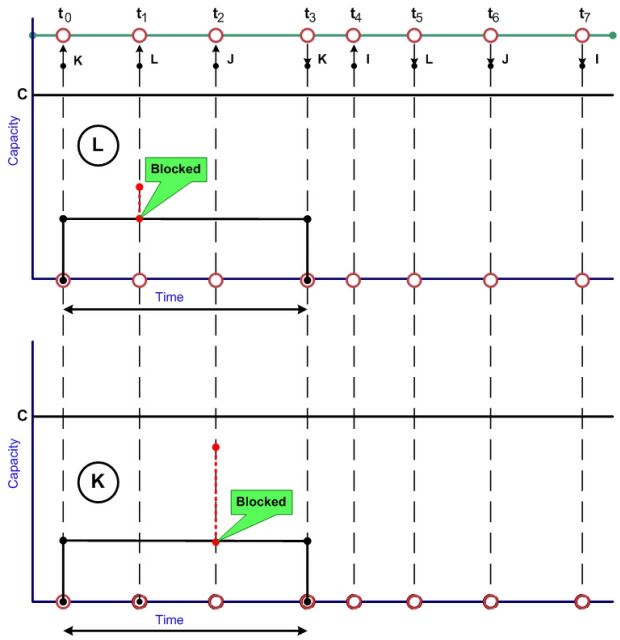

Figure 4: s*ample path: in the above graphic the classes* $\{k, l\}$ *are accepted and in the graphic below the classes* $\{k\}$ *are accepted*

ended at t_4.

When the policy rejects a class k call it should reject at least all classes from $\{k, ..m\}$. But should it reject also the classes $\{k-1, k-2, ..\}$, even when there is enough capacity left. This problem can be split in $k-1$ smaller problems. First the policy determines the expected reward rate by pretending that only the class one calls are active. Next, the policy determines the expected reward rate by pretending that the classes $\{1, 2\}$ calls are active. The policy continues this process till all the classes $\{1, 2, ..k-1\}$ are active. The expected reward rate is denoted as the maximum over all $k-1$ reward rates. Figure 5 uses the same path as in figures 3 and 4 to explain the main idea when the arrival is rejected at time t_0.

Since the class k call is rejected, the policy surely rejects the classes $\{k, l\}$. In the above graphic the policy accepts all the classes $\{i, j\}$. The arrival of class l is not seen by the system, so its wait till t_2. Here the system accepts the class j call. The end of the period is at t_6, the departure of the class j call. In the graphic below the classes $\{j, k, l\}$ are rejected, because they are inactive. Thus the system has to wait till t_4, the arrival of the class i call. The time period ends at t_7. When comparing the sample paths of the figures 3 and 5, the strategy

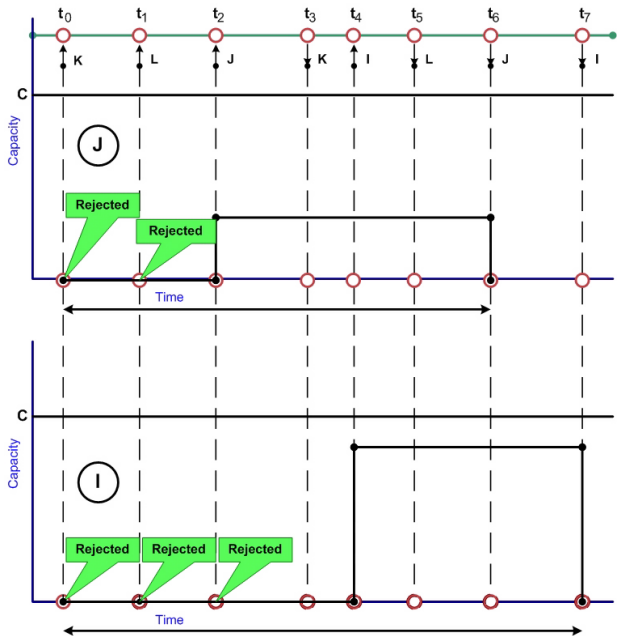

Figure 5: s*ample path: in the above graphic the classes* $\{i, j\}$ *are accepted and in the graphic below the class* $\{i\}$ *is accepted*

would be

$$d_k(\mathbf{n}) = 1 \Leftrightarrow \max\left\{\frac{r_k + r_l}{t_5}, \frac{r_k + r_j}{t_6}\right\} \geq \max\left\{\frac{r_j}{t_6}, \frac{r_i}{t_7}\right\} \tag{20}$$

and if the figures 4 and 5 would be compared, the strategy would be

$$d_k(\mathbf{n}) = 1 \Leftrightarrow \max\left\{\frac{r_k}{t_3}, \frac{r_k}{t_3}\right\} \geq \max\left\{\frac{r_j}{t_6}, \frac{r_i}{t_7}\right\} \tag{21}$$

where is assumed that at t_0 the same capacity is available.

Overall Strategy Now the main idea is intensively discussed, the overall strategy can be formulated. The expected reward rate after both decisions is determined for different combinations of active classes. The expected reward that the policy gains in state $\mathbf{n}$ is denoted as $ER^{\zeta}_{\mathbf{n}}$. The expected time period from the decision epoch until the first accepted arrival is out of the system (this could be immediately after the arrival when the policy blocks the arriving call) in state $\mathbf{n}$ when the classes $\{1, 2, .., \zeta\}$ are active is denoted as $ET^{\zeta}_{\mathbf{n}}$. The least profitable class that is still active is ζ. Now we search the maximum value of equation (22) over $\zeta \in \{1, 2, ..m\}$ for both decisions. Then the overall strategy becomes

$$d_k(\mathbf{n}) = 1 \Leftrightarrow \left\{ \max_{\zeta \epsilon \{1,2,..,m\}} \frac{r_k + ER^{\zeta}_{\mathbf{n}+e_k}}{E \max\{B_k, T^{\zeta}_{\mathbf{n}+e_k}\}} \geq \max_{\zeta \epsilon \{1,2,..,m\}} \frac{ER^{\zeta}_{\mathbf{n}}}{ET^{\zeta}_{\mathbf{n}}} \right\} \tag{22}$$

where the nominator is the expected reward and the denominator is the expected time period. In the next sections the calculations for $ER^{\zeta}_{\mathbf{n}}$ and $ET^{\zeta}_{\mathbf{n}}$ are extensively discussed.

Expected Reward There are two possibilities to interpret the decision rule of the policy. The first possibility is that once the policy accepts or rejects a class k call it does not change its decision when no arrival has been accepted, even when the system reaches a non-critical state. This means that if the policy accepts the classes $\{1, 2, .., \zeta\}$ or rejects the classes $\{\zeta, \zeta + 1, .., m\}$, the policy still accepts or rejects these classes even when the system state reaches a non-critical state. The second possibility is the main rule is still applied. This means that once the system is in a non-critical state, the policy always accepts all calls.

The reason for the first option is that once the system has been in a critical state, we know that the system has almost used all the capacity. Therefore one might think to hold back the number of calls and only accepts the more profitable classes. On the other hand, once the system reaches a non-critical state, more capacity becomes available and the benefit of rejecting becomes more negligible. Hence, both possibilities have their pros and cons. They are both worked out in this section. The first part discusses the policy which holds on to its decision. The second part discusses the policy which applies the main rule.

First Decision Holds In this part the expected reward that is gained after the decision epoch could easily be determined, because the active classes do not change in time and only the reward from the accepted call is taken into account. Suppose all classes $\{1, ..\zeta\}$ are active, which means that these classes should be accepted upon arrival if enough capacity is available. The probability that the first arrival is blocked is denoted as $P(TBL^{\mathbf{n}}_{\zeta})$. The probability that a class i is the first arrival is $\frac{\lambda_1}{\lambda^{\zeta}}$. Thus the probability that a class i is blocked is $P(TBL^{\mathbf{n}}_{\zeta})$ times $\frac{\lambda_1}{\lambda^{\zeta}}$ and the probability that the class i is accepted is one minus the probability that the class i is blocked. The total expected reward is the sum of the probability that a class i is accepted times the reward over all active classes. The calculation of the total expected reward is for both decisions the same, only the number of active classes and the system state differ. Hence, the total expected reward is

$$ER^{\zeta}_{\mathbf{n}} = \sum_{i=1}^{\zeta} \left(1 - P(TBL^{\mathbf{n}}_{\zeta}, \infty) \frac{\lambda_i}{\lambda^{\mathbf{m}}}\right) r_i \tag{23}$$

where $P(TBL^{\mathbf{n}}_{\zeta}, T)$ is the probability that the first arrival is blocked in a certain time period T. This is calculated in Appendix 11.1. For a lot of different types

of classes this is not easy to calculate, see equation (**??**) and therefore this policy is not simulated or analyzed.

Main Rule Holds Unlike in the section where the first decision holds, the policy has the main rule that if the system state is not in a critical state after accepting an arrival, then the arrival must be accepted. The problem of finding the expected reward in this case is more difficult, because for each different state the active classes are different. When the first (or more) events are departures, the system state changes in time. Therefore the active classes change in general, because more capacity becomes available.

Figure 6 shows an example where the active classes change in time.

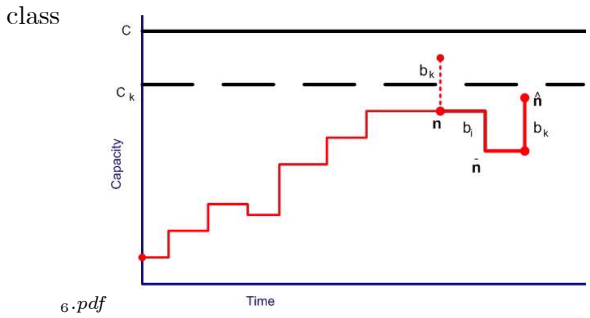

Figure 6: sample path where in state $\mathbf{n}$ a class k is rejected and in state $\bar{\mathbf{n}}$ a class k is accepted

At a certain moment in state $\mathbf{n}$ a class k arrives and it has been rejected. At this moment only the classes $\{1,..,k-1\}$ are accepted. A little later (no arrival has occurred) the system is in state $\overline{\mathbf{n}}$. (departure of a class i). Now a class k can be accepted because after accepting a class k call the system is in state $\widehat{\mathbf{n}}$ which is not critical. Therefore one has to know which classes are active if a class k call arrives in state $\mathbf{n}$. Whether a class is active or not depends on the index ζ in (22) and the total used capacity c. Therefore we develop a tensor with ones and zeros. A "1" means that class i is active and a "0" means that class i is inactive for some index ζ and the used capacity c. The tensor Active-Classes can be developed by

$$\begin{aligned} AC_{i,\zeta,c} &= \begin{cases} 1 & \text{,if } (i \leq \zeta) \cup (c < C_i) \\ 0 & \text{,otherwise} \end{cases} \\ C_i &= C - \max\{b_1, b_2, .., b_{i-1}\} - b_i + \frac{1}{\theta} \text{ ,for } i \in \{2, 3, .., m\} \end{aligned} \tag{24}$$

where $c = \mathbf{n} \cdot \mathbf{b}$, θ is the scale parameter and C_1 is set at C, because a class one call is always accepted if the capacity constraint is not violated. The critical level C_i is the start where all the critical states begin. We add the term $\frac{1}{\theta}$ instead of 1, because we assume that the every size b is an integer for $\theta = 1$.

If $\theta = 2$ then all sizes are divided into half. Figure 7 shows an example for $\mathbf{b} = (2, 3, 2, 5)$.

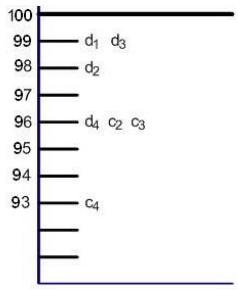

Figure 7: example of the dead levels and the critical levels when $\mathbf{b} = (2, 3, 2, 5)$

In this example all states with at least 8 unit's capacity available accepts all classes. Thus for $c^* = \min\{C_2, .., C_m\} - 1 = 8$ holds that $AC_{i,\zeta,c^*} = 1$, where c^* is the minimum amount of capacity where all classes are active. For $\theta = 1$, (and $\delta = 1$) $C_4 = 100 - \max\{2, 3, 2\} - 5 + 1 = 93$ is a critical level, which is correct because when the system has 93 units capacity used and after a class 4 is accepted then the system has 98 units capacity used and a class 2 call will be blocked. For $\theta = 2, C_4 = 100 - \max\{\frac{2}{2}, \frac{3}{2}, \frac{2}{2}\} - \frac{5}{2} + \frac{1}{2} = 96.5$ is a critical level, which is correct because when the system has 96.5 units capacity used and after a class 4 is accepted then the system has 99 units capacity used and a class 2 call will be blocked. Thus when the system is in a state $\mathbf{n}$ which has at least c^* units capacity available, then the active classes do not change anymore. In fact, all classes are active. In these states the expected reward can easily be determined by

$$ER^{\zeta}_{\mathbf{n}} = \sum_{i=1}^{m} \frac{\lambda_i}{\lambda^{\mathbf{m}}} r_i \ ,\{\forall \mathbf{n} | c^* \geq \mathbf{n} \cdot \mathbf{b}\} \tag{25}$$

Not all active classes are always accepted. It could happen that an active class is blocked because of the capacity constraint. In this case the time period where classes can be accepted ends here and the policy gains nothing. To take this in consideration, we develop a matrix which has also only ones and zeros. A "1" means that a class i is accepted and a "0" means that a class i is blocked by the capacity constraint. The Non-Block-Class matrix can be determined by

$$\begin{aligned} NBC_{i,c} &= \begin{cases} 1 & ,\text{if } c < D_i \\ 0 & ,\text{otherwise} \end{cases} \\ D_i &= C - b_i + 1 \ ,\text{for } i \in \{1, 2, .., m\} \end{aligned} \tag{26}$$

where $c = \mathbf{n} \cdot \mathbf{b}$.

To determine the expected reward in the critical states one has to distinguish whether the first event is an arrival or a departure, because these have different influence on the reward.

When the first event is an active class i arrival and it fits into the system than the policy gains r_i. When the call belongs to an inactive class or the call does not fit into the system, then no reward is gained.

When the first event is a departure then the available capacity increases which gives a possibility that more classes become active. The reward must be determined again, but now for a different state with may-be different classes. Applying all the above information one can develop a recursive formula for the expected reward. The expected reward for the critical states is

$$ER^{\zeta}_{\mathbf{n}}=\sum_{i=1}^{m}\frac{\lambda_i AC_{i,\zeta,c}}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}NBC_{i,c}r_i+\sum_{i=1}^{m}\frac{n_i\mu_i}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}ER^{\zeta}_{\mathbf{n}-e_i} \qquad (27)$$

where the term before the plus is the probability that the first event is an arrival and the term after the plus is the probability that the first event is a departure.

Formula (27) could be verified in the set of the non critical states using formula (25). The following theorem is developed.

Theorem 3 *If* $ER^{\zeta}_{\mathbf{n}}=\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^{\mathbf{m}}}r_i$ *for* $\{\forall\mathbf{n}|c^*\geq\mathbf{n}\cdot\mathbf{b}\}\Longrightarrow ER^{\zeta}_{\mathbf{n}}=ER^{\zeta}_{\mathbf{n}-\mathbf{m}}$ *,with* $\mathbf{m}\subseteq\mathbf{n}\in R^m$ *and* $\mathbf{m}\neq\mathbf{0}$.

Theorem 1 shows that equation (27) holds for all the non-critical states.

Proof.

$$\begin{aligned}
ER^{\zeta}_{\mathbf{n}} &= \sum_{i=1}^{m}\frac{\lambda_i AC_{i,\zeta,c}}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}NBC_{i,c}\frac{r_i}{b_i}+\sum_{i=1}^{m}\frac{n_i\mu_i}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}ER^{\zeta}_{\mathbf{n}-e_i} \qquad (28)\\
&= \sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\frac{r_i}{b_i}+\sum_{i=1}^{m}\frac{n_i\mu_i}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\left(\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m}\frac{r_i}{b_i}\right)\\
&= \frac{\lambda^m\left(\frac{r_1}{b_1}\lambda_1+..+\frac{r_m}{b_m}\lambda_m\right)+(\mu_1n_1+..+\mu_mn_m)\left(\frac{r_1}{b_1}\lambda_1+..+\frac{r_m}{b_m}\lambda_m\right)}{\lambda^m}\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\\
&= \frac{\frac{(\lambda^m+n_1\mu_1+..+n_m\mu_m)\left(\frac{r_1}{b_1}\lambda_1+..+\frac{r_m}{b_m}\lambda_m\right)}{\lambda^m}}{\lambda^m+n\cdot\mu}=\frac{\left(\frac{r_1}{b_1}\lambda_1+..+\frac{r_m}{b_m}\lambda_m\right)}{\lambda^m}=\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m}\frac{r_i}{b_i}
\end{aligned}$$

■

Expected Time Period The two possibilities to interpret the decision rule are also of influence for the expected time period. Therefore both possibilities are worked out in the next sections.

First Decision Holds Like in the previous section where the reward is calculated for the first decision holds, the active classes do not change in time. Now we have to find the expected time period. First, the expected time period is determined for the second arrival if the policy accepts the first arrival. In this

case the expected time period is from the decision epoch until the arrival is out of the system again. This could be immediately after its arrival if the call is blocked. Then the time period is from the decision epoch until the second call arrives. Otherwise the time period is until its departure. The calculation for the expected time period of the second arrival if the policy rejects the first arrival is the same as if the policy accepts the first arrival, but only for a different system state.

The expected time period of the second arrival consists of two parts. The first part is the average residual inter-arrival time, which is for exponential distributions just the average inter-arrival time of the of the second arrival. Second is the service time of the arrival which could be zero. In the same manner as in (23) the probability that the first arrival is not blocked could be determined. The expected time period is the expected time period of the first active class arrival plus its expected service time.

$$ET_{\mathbf{n}}^{\zeta} = EA_{\min}^{\zeta} + \sum_{i=1}^{\zeta} \left(1 - P(TBL_{\zeta}^{\mathbf{n}}, \infty) \frac{\lambda_i}{\lambda^{\boldsymbol{\zeta}}}\right) EB_i \tag{29}$$

where $EA_{\min}^{\zeta}$ is the minimum expected arrival time of all $\{1, 2, ..\zeta\}$ active classes.

The expected time of the first arrival which is accepted is simply its expected service time. The total expected time period is when both calls are out of the system. This is the same as the expected maximum of the time period of the first arrival and the second arrival. To calculate this, one needs the density functions of both time periods. The expected time period of the first arrival has of course an exponential distribution. The expected time period of the second arrival is the sum of one exponential distribution (the arrival time) and a hyper-exponential distribution (the second arrival's service time). To find the density function of the total time period of the second arrival one can use the convolution integral [54]. Let T_1 be the stochastic variable of time period from decision epoch till the arrival of second arrival. Let T_2 be the stochastic variable for the service time of the second arrival. The density function of T_2 is

$$f_{\mathbf{n}}^{\zeta}(t) = \sum_{i=1}^{\zeta} \left(1 - P(TBL_{\zeta}^{\mathbf{n}}, \infty) \frac{\lambda_i}{\lambda^{\boldsymbol{\zeta}}}\right) \mu_i \exp\left(-\mu_i t\right) \tag{30}$$

where the integral of $f_{\mathbf{n}}^{\zeta}(t)$ from zero to infinity not necessary adds to one, because an arrival could be blocked. The density function of T_1 is

$$g_{\mathbf{n}}^{\zeta}(t) = \sum_{i=1}^{\zeta} \lambda_i \exp\left(-t \sum_{i=1}^{\zeta} \lambda_i\right) \tag{31}$$

Using the theory about the sum of two stochastic variables

$$\begin{aligned} f_{X_2+X_1} &= f_{\mathbf{n}}^{\zeta}(t) * g_{\mathbf{n}}^{\zeta}(t) \\ f_{ET_{\mathbf{n}}^{\zeta}}(z) &= \int_{t=0}^{z} f_{\mathbf{n}}^{\zeta}(t) g_{\mathbf{n}}^{\zeta}(z-t) dt \end{aligned} \tag{32}$$

where $f_{ET_{\mathbf{n}}^{\zeta}}(z)$ is the density function of the total time period of the second arrival and " $*$ " is the convolution product. Let $h(x)$ be the density function of the total time period of the first arrival in the system. The maximum of the total expected time period could be determined by

$$E \max \left\{ B_k, ET_{\mathbf{n}}^{\zeta} \right\} = \int_{z=0}^{\infty} \int_{x=0}^{\infty} \max(x,z) h_{B_k}(x) f_{ET_{\mathbf{n}}^{\zeta}}(z) dx dz \tag{33}$$

But since $P(TBL_{\zeta}^{\mathbf{n}}, \infty)$ is very hard to determine for systems with many different types of classes, we do not analyze or simulate this policy. In appendix 11.3 equation (115) one can find the further details of (33) in case of the two exponential distributions.

Main Rule Holds For calculating the expected time period where the main rule is still applied, the same problems occur as in the previous section for the expected reward, the active classes change in time. For both decisions the calculation of the expected time period is again the same, only the system starts in a different state and when the policy accepts a class k arrival, the policy has to wait till all accepted calls are out of the system. To find the expected time period, we condition on the events and make a discretization of the time interval, where each interval is the expected time period between two events. When the event is an arrival of an active class, the time period is over after its departure. When the event is a departure the system state changes and again the expected time period can be calculated. Till the system has c^* units of capacity available which means all classes are active. From this moment a departure is not of interest anymore. Figure 8 shows an example.

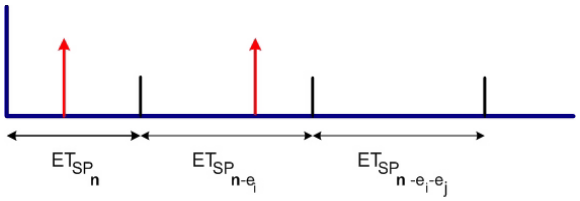

Figure 8: example how ET_{sp} changes in time when departures occur

The red arrows denote the departures. The expected time period from the moment that the system is in state $\mathbf{n}$ until the system state changes (the arrival of an active class or any departure) is

$$ET_{sp_{\mathbf{n}}}^{\zeta} = \frac{1}{\mathbf{AC}_{\zeta,c} \cdot \boldsymbol{\lambda} + \mathbf{n} \cdot \boldsymbol{\mu}} \tag{34}$$

where $AC_{i,\zeta,c}$ is the tensor from (24) with the information about the active classes. As mentioned before, the expected time period in state $\mathbf{n}$ where at

least c^* units of capacity are available consists of two parts. The expected time period in a non critical state $\mathbf{n}$ is

$$ET_n^{\zeta} = \frac{1}{\lambda^{\mathbf{m}}} + \sum_{i=1}^{m} \frac{\lambda_i}{\lambda^{\mathbf{m}}} \frac{1}{\mu_i} \ , \{\forall \mathbf{n} | c^* = \mathbf{n} \cdot \mathbf{b}\} \tag{35}$$

where the first term is the expected time of first arrival of all m classes and the second term is the expected service time of the arrival. To determine the expected time period the same argumentation can be used as for the expected reward. We also distinguish the first event is an arrival or a departure.

When the first event is an active class i arrival and it fits into the system then the call is accepted and the time period is till the departure of class i, see equation (34). In case of an inactive class arrival this call should be rejected. This causes no jump. In case an active call does not fit into the system then the call is blocked and the time period ends. Of course when an inactive call arrives which not fit into the system the policy just simply rejects this call. It does not matter whether it fits into the system or not.

When the first event is a departure then the available capacity increases which gives a possibility that more classes become active. The time period must be determined again, but now for a different state with may-be different classes. The average inter-arrival time of a jump is now increased, because a call has left the system. Applying all the above information one can develop a recursive formula again for the expected time period for the critical states. The expected time period in state $\mathbf{n}$ is

$$ET_n^{\zeta} = \sum_{i=1}^{m} \frac{\lambda_i AC_{i,\zeta,c}}{\mathbf{AC}_{\zeta,c} \cdot \boldsymbol{\lambda} + \mathbf{n} \cdot \boldsymbol{\mu}} \left[NBC_{i,c} \frac{1}{\mu_i} + ET_{sp_{\mathbf{n}}}^{\zeta} \right] + \sum_{i=1}^{m} \frac{n_i \mu_i}{\mathbf{AC}_{\zeta,c} \cdot \boldsymbol{\lambda} + \mathbf{n} \cdot \boldsymbol{\mu}} \left[ET_{\mathbf{n}-e_i}^{\zeta} + ET_{sp_{\mathbf{n}}}^{\zeta} \right] \tag{36}$$

where the term before the plus is the probability that the first event is an arrival and the term after the plus is probability that the first event is a departure.

At the moment that the policy rejects the arrival, the reward rate can be determined by dividing $ER_{\mathbf{n}}^{\zeta}$ over $ET_{\mathbf{n}}^{\zeta}$, see equation (22). But when the policy has accepted the first two arrivals, the time period last till both calls are out of the system. To find a formula for the expected maximum time period of both possible accepted arrivals, one needs to know both density functions of the time period as mentioned before in the previous section. For the first accepted arrival is it very easy, because all service times are assumed to be exponential. For the second call the distribution is much harder to find. It consists of a sum of exponential distributions plus some constant. Assuming that the second distribution is also an exponential distribution with the same mean as the real distribution it is not so hard to calculate the maximum expected time period. Let Z_1 be the time period from accepting the arrival at the decision epoch till its departure with mean $\frac{1}{\mu_1}$and let Z_2 be the time period from the decision epoch until the second arrival is out of the system with mean $\frac{1}{\mu_2}$. Assuming that the latter is exponential, then $Z_1 \tilde{} Exp(\mu_1)$, $Z_2 \tilde{} Exp(\mu_2)$ and can the expected

maximum time period be calculated via

$$E\max\{Z_1,Z_2\}=\frac{(\mu_1)^2+(\mu_2)^2+\mu_1\mu_2}{\mu_1\mu_2(\mu_1+\mu_2)}=\frac{(EZ_1)^2+(EZ_2)^2+EZ_1EZ_2}{EZ_1+EZ_2} \quad (37)$$

One can find the further details of (37) in appendix 11.3 equation (115). Now the expected reward rate when the policy accepts an arrival can also be calculated.

Formula (36) could be verified in the set of the non critical states using formula (35). The following theorem is developed.

Theorem 4 *If* $ET^{\zeta}_{\mathbf{n}}=\frac{1}{\lambda^{\mathbf{m}}}+\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^{\mathbf{m}}}\frac{1}{\mu_i}$ *for*$\{\forall\mathbf{n}|c^*=\mathbf{n}\cdot\mathbf{b}\}\Longrightarrow ET^{\zeta}_{\mathbf{n}}=ET^{\zeta}_{\mathbf{n}-\mathbf{m}}$ *,with* $\mathbf{m}\subseteq\mathbf{n}\in R^m$ *and* $\mathbf{m}\neq\mathbf{0}$.

Theorem 2 shows that equation (36) holds for all the non-critical states.

Theorem 5 ***Proof.***

$$\begin{aligned}
ET^{\zeta}_{\mathbf{n}}=&\sum_{i=1}^{m}\frac{\lambda_i AC_{i,\zeta,c}}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}\left[BC_{i,c}\frac{1}{\mu_i}+ET^{\zeta}_{sp_{\mathbf{n}}}\right]+\sum_{i=1}^{m}\frac{n_i\mu_i}{\mathbf{AC}_{\zeta,c}\cdot\boldsymbol{\lambda}+\mathbf{n}\cdot\boldsymbol{\mu}}\left[ET^{\zeta}_{\mathbf{n}-e_i}+ET^{\zeta}_{sp_{\mathbf{n}}}\right]\\
=&\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\left[\frac{1}{\mu_i}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right]+\sum_{i=1}^{m}\frac{n_i\mu_i}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\left(\frac{1}{\lambda^m}+\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m}\frac{1}{\mu_i}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)\\
=&\frac{\left(\left(\frac{1}{\mu_1}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)\lambda_1+..+\left(\frac{1}{\mu_m}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)\lambda_m\right)}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\\
&+\frac{n_1\mu_1\left(\frac{1}{\lambda^m}+\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)+..+n_m\mu_m\left(\frac{1}{\lambda^m}+\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\\
=&\frac{\lambda^m\frac{1}{\lambda_1+..+\lambda_m+\mathbf{n}\cdot\boldsymbol{\mu}}+\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}+(\mu_1n_1+..+\mu_mn_m)\left(\frac{1}{\lambda^m}+\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\right)}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\\
=&\frac{(\lambda^m+\mu_1n_1+..+\mu_mn_m)\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}+\frac{\lambda^m\left(\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}\right)+(\mu_1n_1+..+\mu_mn_m)\left(1+\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}\right)}{\lambda^m}}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}\\
=&\frac{1}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}+\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{(\mu_1n_1+..+\mu_mn_m)}{\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu}}=\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{\lambda^m+\mu_1n_1+..+\mu_mn_m}{\lambda^m(\lambda^m+\mathbf{n}\cdot\boldsymbol{\mu})}\\
=&\frac{\frac{\lambda_1}{\mu_1}+..+\frac{\lambda_m}{\mu_m}}{\lambda^m}+\frac{1}{\lambda^m}=\frac{1}{\lambda^m}+\sum_{i=1}^{m}\frac{\lambda_i}{\lambda^m}\frac{1}{\mu_i}
\end{aligned}$$

■

Now the overall strategy (22) can be simulated by combining the equations (27),(36),(37) and assuming that the time period of the second arrival is exponential.

5.7.2 Two Steps Ahead

In the previous section the critical levels are chosen in such a way that at least for one profitable class enough capacity is left after accepting a call. A critical level i corresponds with a class i call. In this section the set of non critical states decreases and the set of critical states increases. The levels are chosen in such a way that at least two profitable classes can be expected after accepting a call ($\delta = 2$). By doing this more capacity could be reserved for the profitable classes which could lead to an increasing number of the profitable calls. The number of less profitable calls could decrease. In general, this leads to less reward rates in the transient period and to better reward rates in the steady state distribution. But the latter is not necessary, because may-be this leads to an under-utilization of the system. In this way the extra reward rate that is gained by the increasing number of the more profitable calls is less then the reward rate that is lost by rejecting the less profitable calls.

The calculations for the tensor Active Classes $AC_{i,\zeta,c}$ of equation (24) are the same, but the critical levels are different. The critical level i can be determined via

$$C_i = C - 2\max\{b_1, b_2, .., b_{i-1}\} - b_i + \frac{1}{\theta} \text{ ,for } i \in \{2, 3, .., m\} \tag{38}$$

This policy is also simulated and worked out in the results section.

5.7.3 δ Steps Ahead

Of course, the critical levels could also be chosen in such a way that more then two profitable classes can be accepted after accepting a call. Suppose the policy wants that at least δ profitable classes can be accepted after the acceptance of a class i call, then the critical level i can be determined via

$$C_i = C - \delta\max\{b_1, b_2, .., b_{i-1}\} - b_i + \frac{1}{\theta} \text{ ,for } i \in \{2, 3, .., m\} \tag{39}$$

where for $\delta = \{1, 2, 3, 4, 5\}$ this policy is simulated.

5.7.4 Look Ahead Policy Compares to other Policies

Unlike the *threshold* policy, this policy is not a coordinate convex policy. Figure 9 indicates the difference between the *threshold* and the look ahead policy.

If for a *threshold* policy an arrow (transition) from state $(n_1, n_2 + 1)$ to $(n_1 - 1, n_2 + 1)$ exists then the dotted arrow from state $(n_1 - 1, n_2 + 1)$ to $(n_1, n_2 + 1)$ also exists. But the dotted arrow does not have to exist in the *look ahead* policies and therefore the *look ahead* policies are in general not coordinate convex.

The *look ahead* policies are also not a *reservation* policy. When the system has c units of capacity used and a class k will be accepted then it will always be accepted independent of the system state, because only the total used capacity matters. The *look ahead* policies do take the system state into account. It could happen that the system accepts a class k call when c units capacity are used in

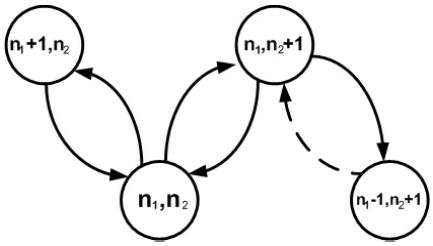

Figure 9: part of a Markov chain with two types of classes

state $\mathbf{n}$, but that it rejects the class k call when c units capacity is used in state $\widetilde{\mathbf{n}}$. Suppose in state (n_1, n_2) in figure 9 the system uses $c = n_1 b_1 + n_2 b_2$ capacity and in state $(n_1 - 1, n_2 + 1)$ the system also uses $c = (n_1 - 1)b_1 + (n_2 + 1)b_2$ capacity. Then if for a *reservation* policy an arrow (transition) from state $(n_1, n_2 + 1)$ to $(n_1 - 1, n_2 + 1)$ exists then the dotted arrow from state $(n_1 - 1, n_2 + 1)$ to $(n_1, n_2 + 1)$ also exists. But the dotted arrow does not have to exist for a *look ahead* policy. Therefore the *look ahead* policies are in general not a *reservation* policy.

The *look ahead* policies are also not like a *complete partitioning* policy, because it does not reserve a fixed amount of capacity for any type of class. It does sometimes reserve some capacity for the high profitable classes by rejecting the low profitable classes. This depends on the system state and revenue rate.

5.8 The Optimal Policy

The policy with the highest reward rate in steady state is defined as the optimal policy. When the scale parameter θ goes to infinity, the reward rate is maximal using the strategy of *Kelly*. But in this case we drop the assumption that θ goes to infinity.

Let $\pi_{\mathbf{n}}$ be the steady state probability that the system is in state $\mathbf{n}$. Let Ω be the set of all possible states then the reward rate in the steady state period is

$$\sum_{\mathbf{n} \in \Omega} n \cdot r \tag{40}$$

To find the optimal set of possible states, we need to know what the steady state probabilities are. The set Ω has $\frac{C}{b_1} \times \frac{C}{b_2} \times .. \times \frac{C}{b_m} = o(C^m)$ states. To determine the steady state probabilities one have to solve as many equations as the number of states. These have the from

$$(\lambda^{\mathbf{m}} + n_1\mu_1 + ..n_m\mu_m)\pi_{\mathbf{n}} = \sum_{i=1}^{m} \lambda \pi_{\mathbf{n}-e_i} + \sum_{i=1}^{m} (n_i - 1)\mu_i \pi_{\mathbf{n}+e_i} \tag{41}$$

Equation (41) is reflected in figure 10.

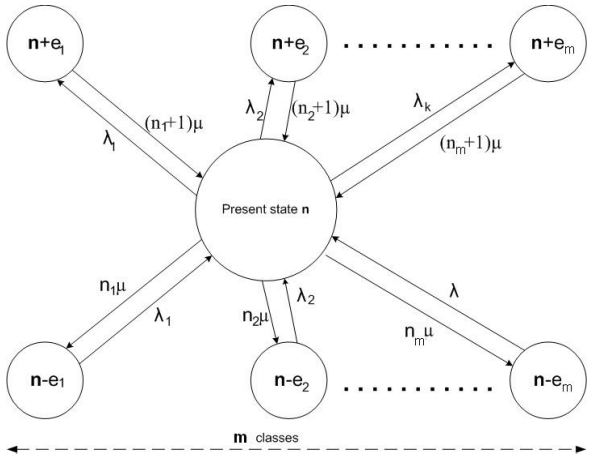

Figure 10: a part of a Markov chain in system state $\mathbf{n}$ with m type of classes

In order to compute the number of different Markov chains, we could remove any combination of arrows. Some combinations lead to the same Markov chain, because the Markov chain has to remain connected. Which Markov chains are the same is not so easy to determine, therefore we use a different approach.

The number of Markov chains can also be calculated once we know the number of different transition states.

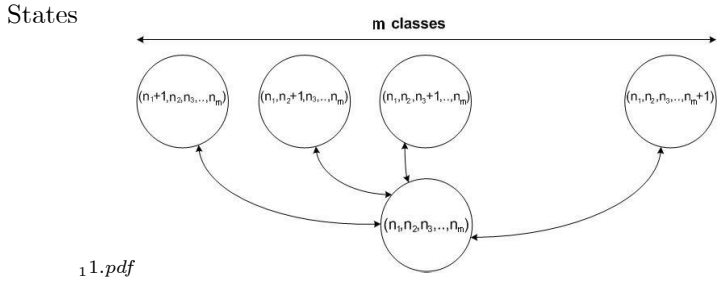

Figure 11: part of a Markov chain with m types of classes

Figure 11 shows a part of the Markov chain with m different types of classes. The set Ω has at most $\left(\frac{C}{\min\{b_1,b_2,..,b_m\}}\right)^m$. Now suppose without loss of generality that $\min\{b_1, b_2, .., b_m\} = 1$ then the system state can be presented with a vector $(n_1, n_2, ..n_m)$ where n_i is the number of calls of class i. Let the zero vector $\mathbf{0}$ represent an empty system and let R_1 be the set of all states which uses one unit capacity. This is the first row in figure 11. Then let R_2 be the of all states which uses two units of capacity and let R_i be the set of states which

uses i units of capacity, see equation (42).

$$R_i = \{\mathbf{n}|\mathbf{n}\cdot\mathbf{b} = i\},\ \text{for } i \in \{0,..C\} \tag{42}$$

The number of transitions equals two times the number of system states (every arrow is double). To count the number system states with m classes we can use the theory of combinations with repetition, see [19]. The number of different states with m classes and C units of capacity is

$$\binom{m+C-1}{C} = \frac{(m+C-1)!}{C!(m-1)!} \tag{43}$$

To determine all the numbers of different states, one have to sum over C. Thus

$$\sum_{c=0}^{C} \frac{(m+c-1)!}{c!(m-1)!} = \frac{(m+C)!}{m!C!} \tag{44}$$

and the number of arrows is two times the number of system states, $2\frac{(m+c)!}{C!m!}$. Since each arrow can be switch on and off, the number of possible Markov chains is at most $2^{2\frac{(m+c)!}{C!m!}} = 4^{\frac{(m+c)!}{C!m!}}$. This is an upper bound, because we have assumed that very call has the minimum size of all calls. The order of the different types of Markov chains is still $o(4^{\frac{(m+c)!}{C!m!}})$. These calculations have a very worse computational complexity.

6 Bounds

In this section we try find a lower and upper bound for the long run average revenue of the *look ahead* policy. We start this section with finding an expression for the total reward in the transient states of the *look ahead* policy. The next sections discusses the long-run revenue rates.

6.1 Transient State

Since an upper or lower bound is very hard to found in the transient period, we find an expression for the total average reward. In this section the transient period is until the first call is rejected or blocked. Now let EQ_i be the average number of calls in the system of class i and suppose without loss of generality that the system start at $t = 0$. Then the average number of class i calls in the system at time t is

$$EQ_i(t) = \frac{\lambda_i}{\mu_i}\left(1 - \exp(-\mu_i t)\right) \tag{45}$$

and the total average reward till the first class is blocked is

$$ER^*(t) = \int_{u=0}^{t} \sum_{i=1}^{m} EQ_i(u)du \tag{46}$$

where t is at most till the first class is blocked or rejected. In case of the *look ahead* policy all classes are accepted until the first critical level is reached. Let T^* be the moment at the end of this period where the lowest critical level is reached. The expected moment in time can be determined by the equation

$$\sum_{i=1}^{m} EQ_i(T^*)b_i = \min_{i\in\{1,2,..m\}} C_i \tag{47}$$

where the $C_i's$ are the critical level of the *look ahead* policies.

Now the total average reward is

$$ER^*(t) = \int_{u=0}^{t} \sum_{i=1}^{m} EQ_i(u)du \text{ for } t \in [0, T^*] \tag{48}$$

6.2 Long-run Revenue Rate

In the next sections the long run average revenue will be investigated. First we mention the upper bound for every policy and second we try to find a lower bound for the *look ahead* policy.

6.2.1 Upper Bound

An upper bound can be found when θ tends to infinity. The policy of *Kelly* generates the maximum revenue rate on the long run, see section 5.4. Once we drop these assumptions the optimal average reward rate can be determined as in section 5.8. The optimal policy makes for every arriving call the optimal decision upon arrival. When we use the optimal policy in general, it could never have a higher average reward rate than the policy of *Kelly*. And since the optimal policy generates at least a higher average reward rate than every policy, the upper bound of *Kelly* is an upper bound in general. Thus the upper bound can be calculated with equation (9).

6.2.2 Lower Bound

To find a lower bound for the *look ahead* policy we split the total capacity. The first part is the capacity till the first critical level, $\min\{C_1, C_2, .., C_k\}$. Until this critical level is reached all types of classes are accepted upon arrival. Now we want to know the average time that the system is below this critical level and in which state it is on average. Figure 12 reflects this idea.

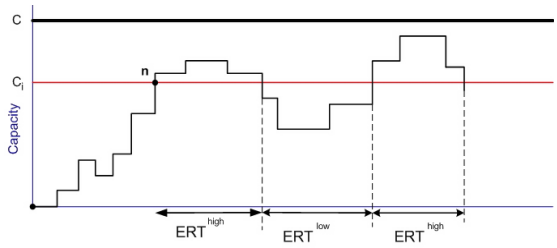

Figure 12: sample path where C_i denotes the lowest critical level

Let RT_i be the random variable of the next return time to state i

$$RT_i = \min\{n : X_n = i | X_0 = i\} \tag{49}$$

where n is the number of time steps. Let $ERT^h_{\mathbf{n}}$ be the expected return time when the system is in state $\mathbf{n}$ and the first event is an arrival. The $ERT^l_{\mathbf{n}}$ is the first return time when the system is in state $\mathbf{n}$ and the first event is a departure. Let us start with finding the ERT^l.

There are a few problems that arise by determining the first expected return time. First we have to distinguish either the system has to return to the exact same level or that the system has to return to the same or higher level. Reaching a higher level means that less capacity is available then in state $\mathbf{n}$. Now suppose that the first return time is when the system is at exact the same level. Should the system also be in exact the same state $\mathbf{n}$ or just at the exact level. The latter means the first return time is at the moment that $\mathbf{n} \cdot \mathbf{b}$ is the same again.

It does not matter which choice is made, for all cases the steady state probabilities have to be known. Since this is the main thing to avoid we have to decrease the number the states. Therefore the first assumption is that all sizes are equal. Then we could say without loss of generality that all sizes are one, $b_i = 1$ for all $i \in \{1, 2, .., m\}$. But since the order of equations that have to be solved is still $O(c^m)$, the second assumption is that all classes have the same departure rate too. Now only the total used capacity is important and we could model the problem as a Markov chain where every state represents a number of units used capacity, see figure 13. Now the number of equations that have to be solved is $O(c)$.

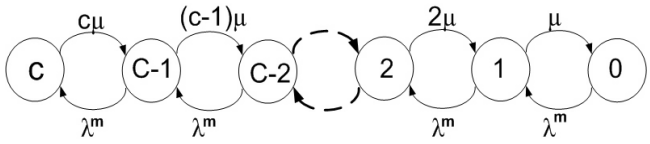

Figure 13: a Markov chain with c different states and where $\lambda^{\mathbf{m}} = \sum_{i=1}^{m} \lambda_i$

The departure rate decreases every time that a call departures, but the arrival rate is the same in every state. The corresponding local balance equations of figure 13 are

$$(k\mu)\pi_k = \lambda^{\mathbf{m}} \pi_{k-1} \text{ for } k \in \{c, c-1, .., 1\} \tag{50}$$

where π_k is the steady state probability that the system has used k units of capacity. Equation (50) together with $\sum_{k=0}^{c} \pi_k = 1$ becomes

$$\pi_k = \frac{(\frac{\lambda^{\mathbf{m}}}{\mu})^k}{k!} \pi_0 \text{ for } k \in \{c, c-1, .., 1\} \tag{51}$$

where $\pi_0 = \frac{1}{1+\frac{\mu}{\lambda^{\mathbf{m}}}+\left(\frac{2\mu}{\lambda^{\mathbf{m}}}\right)^2+\left(\frac{3\mu}{\lambda^{\mathbf{m}}}\right)^3+..+\left(\frac{c\mu}{\lambda^{\mathbf{m}}}\right)^c}$.

According to [51] the expected first return time in state c can be calculated with the transition rates and the steady state probabilities via

$$ERT^l = ERT_c = \frac{1}{\sum_{i=0}^{c} p_{ic} \pi_i} \tag{52}$$

where p_{ic} is the transition rate from state i to c. The transition rates from k to $k-1$ are always $k\mu$ and the transition rates from $k-1$ to k are always $\lambda^{\mathbf{m}}$. The other rates are always 0. Now all the information is available for calculating the expected first return time.

Now we can find a lower bound for long run revenue rate in the period that the available capacity is below the lowest critical level. During this period the

long run revenue rate is at least $k \min\{r_1, r_2, .., r_m\}$ in each state k. The long run revenue rate is therefore at least

$$ERR^l = \sum_{k=0}^{c} k \min\{r_1, r_2, .., r_k\} \pi_k \tag{53}$$

Unfortunately a lower bound for the long run revenue rate in the period that the available capacity is equal or above the critical level can not be found in the same way, because not all arrivals are accepted upon arrival. The transition rates depend on the system state and not on the available capacity. Therefore we find a minimum expected return time. The system has to wait at least until the first departure. On average this period has length $\frac{1}{c\mu}$. The long run revenue rate during this period is at least

$$ERR^h = k \min\{r_1, r_2, .., r_k\} \tag{54}$$

where the length of this period is on average $\frac{1}{c\mu}$, thus $ERT^h = \frac{1}{c\mu}$.

At the moment that the system is in state $\mathbf{n}$ and the total capacity that is used is c then the first event is with probability $\frac{\lambda^{\mathbf{m}}}{\lambda^{\mathbf{m}}+c\mu}$ an arrival and with probability $\frac{c\mu}{\lambda^{\mathbf{m}}+c\mu}$ a departure. So a lower bound for the long run revenue rate is

$$ERR^{lb} = \frac{\frac{\lambda^{\mathbf{m}}}{\lambda^{\mathbf{m}}+c\mu} ERR^h ERT^h + \frac{c\mu}{\lambda^{\mathbf{m}}+c\mu} ERR^l ERT^l}{ERT^h \frac{\lambda^{\mathbf{m}}}{\lambda^{\mathbf{m}}+c\mu} + ERT^l \frac{c\mu}{\lambda^{\mathbf{m}}+c\mu}} \tag{55}$$

Since in general not only the least profitable class in the system, the lower bound can be increased. Moreover this policy aims at a better average class ratio then the *accept all* strategy. The class ratio is the ratio of the number of calls of all types of classes. Thus the assumption that the class ratio is equal to the *accept all* strategy gives a lower bound. Therefore the lower bound can be stretched. On average the class ratio depends only on the arrival rate, since all sizes and departure rates are the same for all classes. Therefore on average the lower bounds can be extended to

$$\begin{aligned} ERR^l &= \sum_{k=0}^{c} k\pi_k \left(\sum_{i=1}^{m} r_i \frac{\lambda_i}{\lambda^{\mathbf{m}}} \right) \\ ERR^h &= k \left(\sum_{i=1}^{m} r_i \frac{\lambda_i}{\lambda^{\mathbf{m}}} \right) \end{aligned} \tag{56}$$

which leads to an higher ERR^{lb}

7 Computation Time

In the previous sections the main goal was to maximize the long run revenue rate and to get a good performance in the transient period. Since the goal of this report is also to minimize computational complexity, we intensively compare all policies. The computational complexity is a measure for the number of calculations of the parameters from the corresponding policy. The goal in this section is to measure the sensitivity of the parameters that depends on the computational complexity. In mathematical words, we determine the order of the algorithm that gives the policy parameters. Almost every policy's computational complexity depends on the number of classes and the total capacity. So the order of the computational complexity will be expressed in C, the total capacity and m, the number different classes. During this period we assume that for all policies the time scale parameter is one ($\theta = 1$), because in principle θ also depends linear on the capacity. This section starts with the well-known policies and finishes with the *look ahead* policies.

7.1 The Optimal Threshold Policy

The decision model of a *threshold* policy is already described in section 5.1. When the system contains of m different types of classes then there are m parameters to be calculated, since every class has it own parameter t_k.This parameter denotes the threshold for class k and there are C^m different types of *threshold* policies. The optimal *threshold* policy has m different parameters to be calculated. Finding the optimal policy using brute-force search becomes intractable for large C and especially for large m. Therefore in [39] they proposed an iterative coordinate search algorithm to find a coordinate optimal *threshold* policy among all *threshold* policies. This algorithm should reduce the computational complexity. The algorithm searches $O(Cm)$ different *threshold* policies. The number of iterations required by the algorithm is $O(m)$ and the computational complexity of the evaluating the performance of a *threshold* policy is $O(C^2 m \log m)$ with binary tree implementation proposed in [45]. Hence the practical computational complexity is $O(C^3 m^3 \log m)$.

7.2 The Optimal Complete Partitioning Policy

The decision model of a *complete partitioning* policy is already described in section 5.2. In [49] the optimal *complete partitioning* policy is given by the following resource allocation problem

$$\begin{aligned} \max J(s) &= R_1(s_1) + R_2(s_2) + .. + R_m(s_m) && (57) \\ s.t.\ s_1{+}s_2{+}..+s_k &= C \\ 0 &\leq s_k \leq F\ ,\ s_k \in \mathbb{N} \text{ for } k = \{1, .., m\} && (58) \end{aligned}$$

where $R_k(s_k)$ is the average revenue generated by class k. Let $f_k(y)$ be the maximum revenue generated by classes 1 through m with y units of capacity

available. To find the parameters $s_1, s_2, ..s_m$ one have to solve the corresponding dynamic programming equations

$$\begin{aligned} f_1(y) &= R_1(y) \qquad\qquad , 0 \leq y \leq C \\ f_k(y) &= \max_{0 \leq s \leq y} \left\{ R_k(s) + f_{k-1}(y-s) \right\}, 0 \leq y \leq C,\ k = 2, .., m \end{aligned} \tag{59}$$

The above recursive equations have complexity $O(C^2 m)$ and gives as by-product the optimal *complete partitioning* policy $\mathbf{s} = (s_1, .., s_m)$. Since equation (59) already gives the performance evaluating of the *complete partitioning* policy, the practical computational complexity is $O(C^2 m)$.

7.3 The Optimal Reservation Policy

The decision model of the *reservation* policy is already described in section 5.3. When the system contains of m different types of classes then there are m parameters to be calculated, since every class has it own parameter r_k and there are C^m different *reservation* policies, like the *threshold* policy. But unlike the *threshold* policy, the *reservation* policy is in general not coordinate convex. So the binary tree implementation for evaluating the performance of the *reservation* policy can not be used. Therefore in [39] they proposed a different algorithm for the performance evaluation. The computational complexity of this algorithm is $O(C)$. They also developed an iterative coordinate search algorithm for reducing the computational complexity. This algorithm does not search through all different *reservation* policies C^m, but searches at most in $(C+1)(m-1)$.policies. The number of iterations of the algorithm is $O(m)$. Hence the practical computational complexity is $O(C^2 m^2)$.

7.4 One Step Look Ahead Policy

The decision model of the *one step look ahead* policy is already described in section 5.7. Let us first find an upper bound for the number of parameters for the *one step look ahead* policy. Let $|d_k(\mathbf{n})|$ be the number of parameters for the *look ahead* policy. Since the decision parameter depends on the system state in the critical states and the class arrival, an upper bound for the number of parameters can be determined via

$$\begin{aligned} |d_i(\mathbf{n})| &= \sum_{i=2}^{m} \{ |\mathbf{n}| \in \Omega | D_i > \mathbf{n} \cdot \mathbf{b} \geq C_i \} \\ &\leq \sum_{i=2}^{m} \{ |\mathbf{n}| \in \Omega | \mathbf{n} \cdot \mathbf{b} \geq C_i \} \\ &\leq (m-1) \left\{ |\mathbf{n}| \in \Omega | n \cdot b \geq \min_{i \in \{2,..,m\}} C_i \right\} \\ &\leq (m-1)(C - \min_{i \in \{2,..,m\}} C_i)^m \\ &\leq (C - \min_{i \in \{2,..,m\}} C_i)^{m+1} \end{aligned} \tag{60}$$

where $C_i = C - \max\{b_1, b_2, .., b_{i-1}\} - b_i + 1$, $D_i = C - b_i + 1$ and $|\mathbf{n}|$ means the number of states. The critical factor is of course the number of critical levels, $C - \min\{C_2, .., C_m\}$. Let $b_{\max}$ be the maximum size over all classes and let $\varkappa$ be the number of critical levels. Of course the decision parameters for all class k calls which can not be accepted because of the capacity constraint, have not to be calculated. Then the maximum number of critical levels becomes

$$\begin{aligned} \varkappa &= \max_{i\in\{2,..,m\}} \{D_i - C_i\} \\ &= \max_{i\in\{2,..,m\}} \left\{(C - b_i{+}1) - \left(C{-}\max\left\{b_1, b_2, .., b_{i-1}\right\} {-}b_i{+}1\right)\right\} \\ &= \max_{i\in\{2,..,m\}} \{\max\{b_1, b_2, .., b_{i-1}\}\} \\ &= \max_{i\in\{1,..m-1\}} b_i \\ &= b_{\max} \end{aligned} \tag{61}$$

Hence the maximum number of parameters is

$$|d_i(\mathbf{n})| \leq (b_{\max})^{m+1} = \varkappa^{m+1} \tag{62}$$

Let us now investigate in the computational complexity of the decision parameter. For every decision parameters the expected reward rate and the expected time period have to be calculated for different $\zeta \in \{1, ..m\}$, see equation (22). The reward rate is recursively determined, see equation (27). The number of recursions is maximal $\varkappa$. Because when the system is not in a critical state anymore the reward rate is known. Also the expected time period is recursively determined, see equation (36) with a maximum of $\varkappa$ recursions. Hence the practical computational complexity is $O(m\varkappa^{m+2}) = O(m(b_{\max})^{m+2})$

7.5 Two Steps Look Ahead Policy

Like in the previous section, all calculations and equations remain the same with the *one step look ahead* policy, except for the critical level $C_i = C - 2\max\{b_1, b_2, .., b_{i-1}\} - b_i + 1$. The number of critical states becomes

$$\begin{aligned} \varkappa{=}\, 2 \qquad & \max_{i\in\{2,..,m\}} \{D_i - C_i\} \\ = \;& 2b_{\max} \end{aligned} \tag{63}$$

The maximum number of recursions in equations (27) and (36) is $\varkappa$. Thus the practical computational complexity is $O(m\varkappa^{m+2}) = O(m(2b_{\max})^{m+2})$

7.6 δ Steps Look Ahead Policy

The critical level now becomes $C_i = C - \delta\max\{b_1, b_2, .., b_{i-1}\} - b_i + 1$. The number of critical states becomes

$$\begin{aligned} \varkappa{=}\, \delta \qquad & \max_{i\in\{2,..,m\}} \{D_i - C_i\} \\ = \;& \delta b_{\max} \end{aligned} \tag{64}$$

The maximum number of recursions in equations (27) and (36) is $\varkappa$. Thus the practical computational complexity is $O(m\varkappa^{m+2}) = O(m(\delta b_{\max})^{m+2})$

7.7 The Optimal Policy

The total number of different Markov chains is already calculated in section 5.8. The computational complexity of calculating the reward rate given such a Markov chain is not yet calculated. The number of equations to solve the according Markov chain is C^m, see figure 10 and equation 41. Since for every different Markov chain the reward rate has to be calculated the computational time is $o(4^{\frac{(m+c)!}{m!C!}} C^m)$.

8 Simulation Results

In this section, we describe some numerical experiments with a call admission control problem. This problem arises when a service provider with limited resources (capacity) has to accept or reject incoming calls of several types. The objective is to maximize long-term average revenue with also a good performance in the transient period. The latter is intensively investigated in the transient period subsection and the long-term average revenue will be discussed in the steady state section. This section ends with some overall conclusions about the numerical experiments of all the simulated policies.

8.1 Simulation Parameters

The system parameters are from [22]. The system has 100 units of capacity available and there three types of classes.

Class	λ	μ	Size (b)	Reward Rate (r)
1	40	0.5	0.1	1
2	80	2	0.15	0.25
3	60	0.3	0.55	0.75

(65)

To highlight the differences between all policies in one picture, all the policies are compared with the *accept all* strategy.

8.2 Transient Period

The numerical experiments can be split up in two parts. The first part investigates in the so called transient period. This period is not indisputable a predefined time period, but differs in many papers. Some denote the transient period as the time period from the start of the simulation till the steady state is reached where others define the transient period from the start of the simulation till t is about $\frac{1}{\mu_{\min}}$,see [22]. In this report we define the transient period from the beginning of the simulation until the steady state is reached. Of course there is not one moment in time from which the steady state is reached, but this happens gradually in a time period. In this period the average reward rate does not change anymore. This quantity is very important to compare the different policies. Suppose there are two different policies A and B and the two policies are in the steady state period at a certain moment t. The reward rate of policy A is higher than for policy B. Now suppose that policy A has gained more (money) than policy B, then policy B can never catch up policy A again. But when policy B has not reached the steady state it could still catch up policy A. Moreover when in the first situation policy B has a higher reward rate, it definitely catch up with policy A. Now let's turn back to the transient period.

Like mentioned before the steady state is reached when the average reward rate or when average number of calls in the system from every class do not change in time. This moment is for every policy different. It depends on the strategy of the policy. Some strategies do not depend on the used capacity, like

the *accept all* strategy where other strategies have very different decisions when less capacity become available, like the *step ahead* policy or the *reservation* policy. The policy that does not depend on the used capacity reaches in general the steady state earlier than the strategies which does depend on the used capacity. An exception to this is the policy of *Kelly*, which strategy does not depend on the used capacity but rejects some calls when almost all capacity is still available. Of all policies that have been simulated this policy reaches the steady state as last. Therefore we use the *Kelly* policy to define the transient period. Figure 14 shows the number of classes with the system parameters of (65) for the policy of *Kelly* and *one step look ahead* policy.

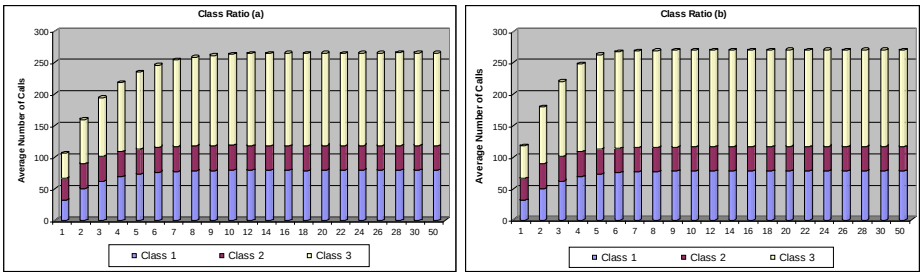

Figure 14: average number of calls over time

The number of calls in the system when the policy of *Kelly* is used do not change from the time period about $t = 12$ where the number of calls for the *one step look ahead* policy do not change from about $t = 6$. Therefore, the transient time period is at least till $t = 12$.

Another method to find the steady state period is to look at the used capacity. As mentioned before the decisions of some strategies are different when less capacity becomes available. For example the *reservation* policy could accept less profitable classes when a lot capacity is available but rejects the same type of class when little capacity is available. The period from upon little capacity is available is very interesting, because the policies now have to deal with the fact that not all calls can be accepted. From upon the moment that calls are blocked or rejected, the class ratio could change and therefore the reward rate. In order to wait till the class ratio remains the same, all calls which are accepted before the moment that little capacity is available should be out of the system. Since the service time is stochastic, we claim that all calls should be departed with high probability, lets say more then 99%

The moment when little capacity is available is defined as the moment when the first call is blocked or rejected by the system. In figure 15 the total used capacity is shown. When about 95% of the total capacity is used, the first calls are blocked or rejected. All policies use about 97% capacity.

This occupation level is reached about $t = 12$. In order to have that each call in the system has departed with at least 99% probability, we only have to

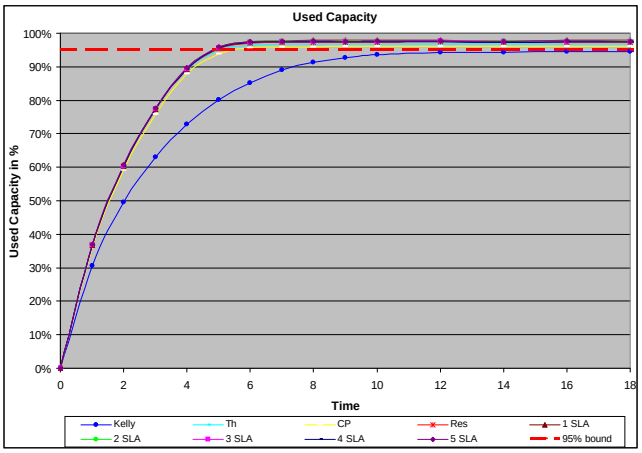

Figure 15: Used Capacity

claim the calls with highest average service duration have departed with at least 99%. The class two calls have the highest service duration, $\mu_3 = 0.3$. In order to satisfy a 99% probability departure level, equation (66) should hold.

$$1 - \exp(-\frac{t}{\mu_3}) > 0.99 \tag{66}$$

After solving equation (66) it follows that t is at least 16 units of time. Therefore we could also define the transient period as from the start of the simulation until $t = 12 + 16 = 26$. But since the above calculations are not so exact, we take a backup and say that the steady state is reached at least from $t = 30$ units of time. So the transient period is defined as the period from the beginning of the simulation till $t = 30$.

Now the transient period is intensively discussed, we show the numerical results for $\theta = \{1, 2, 4, 8, 16, 32\}$. Figure 16 shows for every policy the difference with the *accept all* policy of the total reward.

As you can see an increasing θ hardly influences the relative rewards in the transient period.Only the *Kelly* policy is significantly better for increasing θ. The rest of the policies have more or less the same total average reward. Of course every policy has gained less (money) at a moment in time than the *accept all* policy, but almost every policy catch up with the *accept all* policy within a short period of time. Only for small numbers of θ the *complete partitioning* policy and the policy of *Kelly* do not catch up with the *accept all* policy. The *reservation* policy and the *four, three* and *two steps look ahead* policies have the highest total average reward. The *one step look ahead policy* and the *threshold* policy have a little less average reward. The *complete partitioning* policy is a little worse than the *accept all* policy and the *Kelly* policy performs very badly in the transient period. Thus to come back to our problem formulation, one

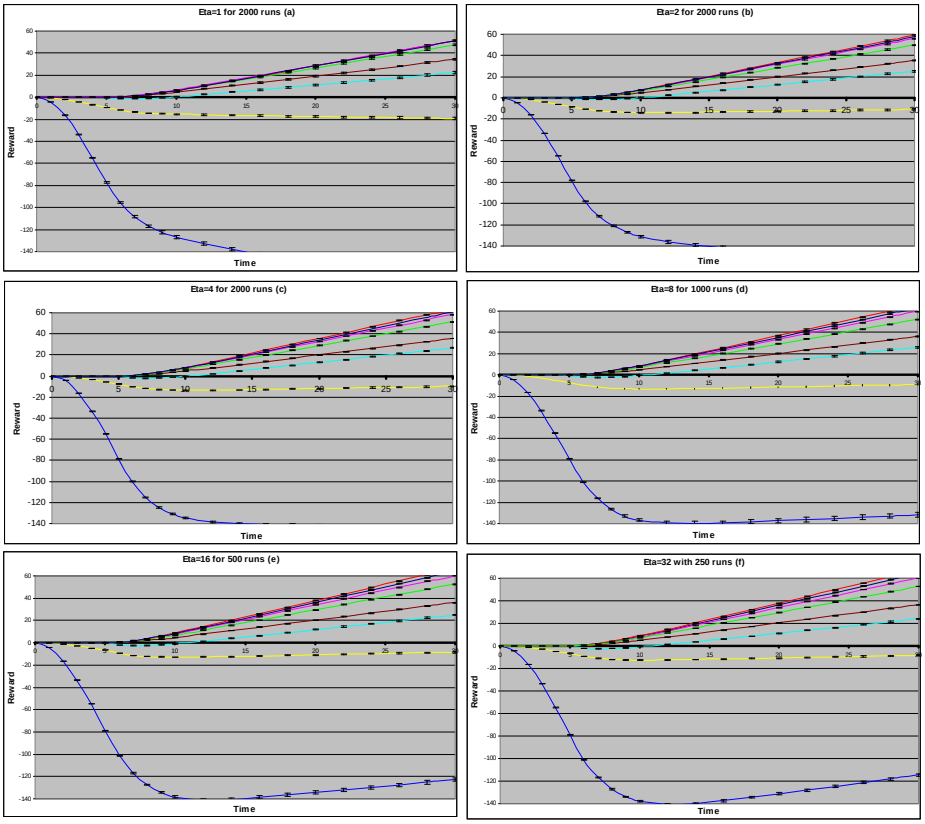

Figure 16: average difference in total reward with the *accept all* policy with a 95% confidence interval

can say that all the *look ahead* policies, but also the *reservation* and *threshold* policy perform well in the transient period.

The policy of *Iyengar & Sigman* is not shown in figure 16, because the difference with the *accept all* policy was too big. It really has bad performance measures for every θ.

8.3 Steady State Period

In the steady state period only the reward rate is of interest. To get a idea how to generate a high reward rate, the section ends with the ratio between the different kind of classes per policy or the class ratio. Figure 17 shows for $\theta = \{1, 2, 4, 8, 16, 32\}$ the reward rates. Notice that the scales on the reward rate axis differ when θ increases.

The reward rate is calculated by subtracting the total reward at $t = 30$ from

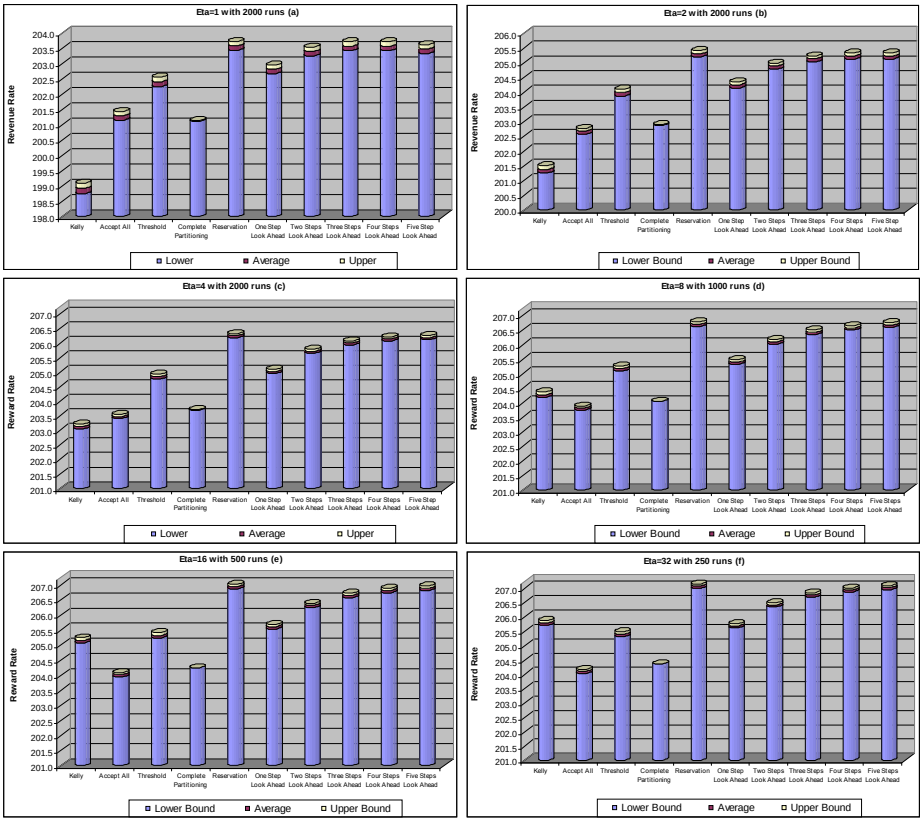

Figure 17: long run revenue rate with a 95% confidance interval

the total reward at $t = 50$ and dividing over 20, the difference in time. Some policies perform better then other policies when θ increases. Beforehand, the *Kelly* policy should outperform all policies when θ increases. The first cylinder indeed catch up with the other policies, this cylinder indicates the reward rate of the *Kelly* policy, but even when $\theta = 32$, the *Kelly* policy does not perform as well as the *reservation* policy or the *look ahead* policies. But for large θ the problem is not a stochastic problem anymore. The intensity rate tends to infinity, with other words the inter-arrival times becomes deterministic. Therefore this is not a stochastic problem, but a deterministic. Furthermore, the inter-arrival times are already very small for small θ in this system compared to applications in the telephone branch.

To most important issue that figure 17 shows us is the answer to our problem formulation, the differences in the long-run revenue rates of each policy. For small numbers of θ we see that the *reservation* policy and the look ahead policies perform well. The best look ahead policies are the *three* and *four steps look ahead*

policy for small θ. These policies have the same reward rates, but make different decisions. The *one step look ahead* policy and *two steps look ahead* policy have a little lower long-run revenue rate, but they are still good. The *five steps look ahead* policy has also a lower reward rate than the *three* and *four steps look* policy. At first this result goes against our feeling, but it is explicable. Figure 18 shows that the difference in critical levels between the *five steps look ahead* policy and the *four steps look ahead* policy. The reason why the *four steps look ahead* has a higher reward rate is due to the fact that although the *five steps look ahead* policy reserves more capacity for the more profitable classes, this is not always the best decision. It appears that the *five steps look ahead* policy reserves too much capacity for the more profitable classes. In state **n** in figure 18 the *four steps look ahead* policy accepts the arriving call, because state **n** is not a critical level. In case the system uses the *five steps look ahead* policy then the state **n** is a critical level. Since the policy does not make always the best decision, it could happen that the *four steps look ahead* policy has a higher reward rate than the *five steps look ahead* policy. Moreover, when δ (the number of steps which the policy looks ahead) increases the reward rate decreases for $\theta = 1$.

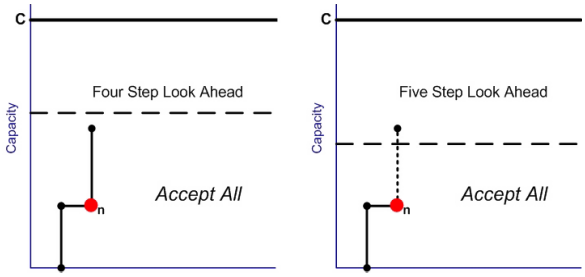

Figure 18: two sample paths where the *four step look ahead* accepts the call in state **n** and the *five step look ahead* rejects the call.

For an increasing θ the reward rates are also increasing for every policy, but the mutual differences also increase. The *one step look ahead* performs relative more worse when θ increases. For large θ the best look ahead policy is always the *five steps look ahead* policy. Likely a *more steps look ahead* policy is even better than the *five steps look ahead* policy, but this is beyond the goals of this report. Since the five steps look ahead policy has already a very good average reward rate. The *reservation* policy has for every θ the best reward rate, but does not differ much from the best look ahead policy. The increasing θ has no influence on the relative reward rates of the *complete partitioning* policy and the *threshold* policy. As expected, the *Kelly* policy has higher relative reward rate for increasing θ, but never catch up with the *reservation* policy or the best *look ahead* policy. Only theoretical or in extreme situations the policy of *Kelly* outperforms the other policies.

The policy of Iyengar & Sigman is not shown in figure 17, because this policy

has very bad results. The reward rate of Iyengar & Sigman is about 49% of the maximum reward rate for $\theta = 1$ and 93% of the maximum reward rate for $\theta = 32$, see [22] Table 1 on page 1716. For the transient state it is even worse.

8.3.1 Deeper Insight

As mentioned before we should give an idea how the strategies differ form each other and why the reward rates are different. The difference in reward rate could be caused by either the difference in ratio between the number of classes or an occupation difference of the used capacity. First the number of resources that are used by each policy are investigated. Figure 19 shows that the most policies use more or less the same amount of resources, except the *complete partitioning policy, threshold* and the *Kelly* policy.

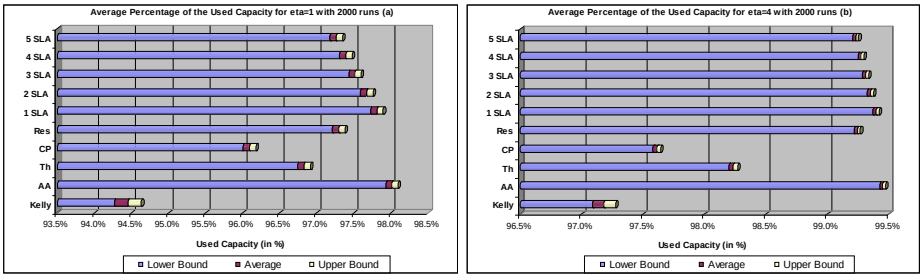

Figure 19: average percentage of the used capacity with a 95% confidance interval

These policies uses respectively $2\%, 1.5\%$ and 2.5% less capacity than the other policies and since the differences in reward rates are about the same percentages this explains why these policies have less reward rates than the other policies. The policy which uses the most capacity is the *accept all* policy, although this is certainly not the policy with highest reward. The reason why the *accept all* policy has not such a good reward rate can therefore not be the low occupation level.

A little striking is the fact that the *two steps look ahead* policy uses the most capacity among all the *look ahead* policies, but the *four steps look ahead* policy has for $\theta = 1$ the highest reward rate and the five steps look ahead policy has for $\theta = 4$ the highest reward rate. Apparently the *four* and *five steps look ahead* policies have a higher reward rate than the *two steps look ahead* policy on different reasons.

This other reason that could explain the difference in reward rate is the difference of the average class ratio. The average class ratio is ratio of the average number of calls in the system in steady state of each class. Figure 20 shows the average class ratio of each policy at $t = 50$. All policies are standardized with the *accept all* policy. The values of each class is the difference with the *accept all* policy of the average number of calls in the system. The average number of

classes is calculated over 2000 runs. The length of the 95% confidence interval is almost the same for every policy and for every type of class, except the class three of *Kelly* and the class two of the *complete partitioning* policy. The length of the 95% confidence interval of the difference in the average number of calls is for all classes at most 0.4, for the class three calls of *Kelly* it is 1.22 and for the class two calls of the *complete partitioning policy* policy it is 0.92.

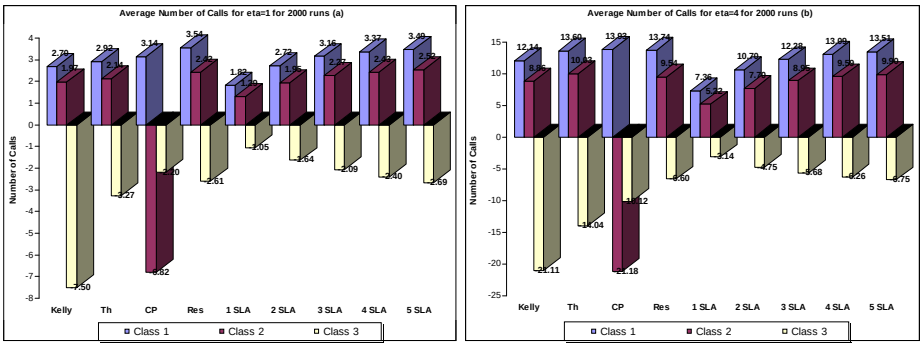

Figure 20: difference average number of calls with the *accept all* policy

Each color has its own type of class. The policy with the highest number of class one calls (the most profitable calls) is the *complete partitioning policy* policy. But this is by far not the policy with the highest reward rate, see figure 17 for $\theta = 1$ and $\theta = 4$. This is due to the fact that the *complete partitioning* policy has much less number of class two and three types. Also the *threshold* policy has a very high number of class one and class two calls, but is also not one of the best policies. This policy rejects to many class three calls. The best policies, the one with the highest reward rates according to figure 17 are the *reservation* policy and the *four steps look ahead* policy. It is clear that they have more class one and two calls and less class three calls than the *accept all* policy. In fact they have more class one and class two calls than all the other *look ahead* policies but less class three calls. It appears that they have the best ratio between the different types of classes to get the highest reward rate.

8.4 Overview

In this section all the quantities of all the simulated policies are judged and summed up in table 2. The table contains information about total reward in the transient period, the reward rate in de steady state and the computational complexity. The words "good","medium" and "bad" indicates how well the policy perform compared to the other policies. The word medium is shorted to "med".

Policy	Transient Small θ	Period Large θ	Steady Small θ	State Large θ	Computation Complexity
Kelly	*bad*	*good*	*bad*	*good*	–
AA	*med*	*med*	*med*	*bad*	–
Th	*med*	*med*	*med*	*med*	$O(C^3m^3 \log m)$
CP	*med*	*med*	*med*	*med*	$O(C^2m)$
Res	*good*	*good*	*good*	*good*	$O(C^2m^2)$
SLA	*good*	*good*	*good*	*good*	$O(m(\delta b_{\max})^{m+2})$
Optimal	–	–	*good*	*good*	$O(4^{\frac{(m+C)!}{m!C!}}C^m)$
Iye & Sig	*bad*	*bad*	*bad*	*med*	–

(2)

For the *threshold*, *complete partitioning policy*, *reservation* and the *look ahead* policy is used the best policy. For some policies the computational complexity differs when θ increases. This is due to the fact that the ratio between the call sizes and the capacity differ. In table 2 the computational complexity is measured for $\theta = 1$. For increasing θ the computational complexity also increases for the *threshold*, *complete partitioning policy*, *reservation, look ahead* and the *optimal* policy. Because when θ increases, the call sizes decreases, see (8). This is the same as increasing the total capacity C. So when for example θ doubles, the calls sizes should be divided into halves or the capacity doubles. The δ from the *look ahead* policies also depend on θ, but in a very different way. Unfortunately this is not calculated in this report. Practical the δ should not be larger than 5 with the usual parameters, see (65). But for other parameters this is not excluded.

9 Conclusions

In this report the admission control problem in loss networks is studied. The goal of this report is to develop a policy which performs well in both the transient and the steady state period and that do not face the curse of dimensionality.

We have proposed a policy that supports several classes of calls and systems with high capacity levels. First, the basic idea of the construction of the policy is investigated and what this policy should achieve. Then the policy is compared with all kinds of policies, like the optimal *threshold* policy, the optimal *complete partitioning* policy, the *reservation* policy, the policy of *Kelly* and the policy of *Iyengar & Sigman.* The policies are judged on three quantities, the average reward in the transient period, the long-run revenue rate and the computational complexity of the policy. All these quantities are measured on basis of numerical experiments. We also considered a very width range of θ, the limiting regimes.

The first quantity is the reward in the transient period. According to the numerical experiments all the policies perform well in the transient period, only the policy of *Kelly* and the policy of *Iyengar & Sigman.* These policies reject too many calls in the beginning to achieve their desired class ratio.

The second quantity is the long-run revenue rates. The policies with the best average reward rate are the *reservation* policy and the *look ahead* policies. There is not one indisputable best *look ahead* policy, but this depends on θ. For small numbers of θ (time scale) the δ (the number of steps which the policy looks ahead) should also be small and for large numbers of θ the δ should also be larger. The optimal *threshold* policy, the optimal *complete partitioning* policy and the policy of *Kelly* have less reward rates. The *Kelly* policy experiences the most influence of θ.The reward rates are relative better for large number of θ. An increasing θ has no influence on the reward rate of the *threshold* policy and the *complete partitioning* policy.

The last quantity is the computational complexity. The policy of *Kelly* and the *accept all* policy are very easy to compute and does not depend on the number of classes or the capacity. All the other policies depend on the number of classes and the total amount of capacity. The *threshold*, *complete partitioning policy*, *reservation* policy have all a polynomial complexity. The *complete partitioning* policy has the lowest computational complexity and the *threshold* policy has the highest computational complexity. The *look ahead* policy and the optimal policy have exponential complexity, but the *look ahead* policy does only in theory depends on the capacity. Practical, the number of classes depends exponential on the largest call size for the optimal *look ahead* policy. For systems with a large capacity and a small number of classes, the *look ahead* policy needs less computational time than the policies with polynomial complexity.

Overall, the optimal *reservation* and the optimal *look ahead* policy have the best performance measures and fulfill the demands of the problem formulation.

10 Recommendations

As discussed in section 5.7, all the *look ahead* policies make their decision based on the same *look ahead* period. For the *one step look ahead* policy this is very logic, but for the *more steps look ahead* policies this is a little contradictorily. The *one step look* policy tries to reserve capacity for one profitable class when this is optimal by conditioning on looking one arrival ahead. The *two steps look ahead* policy tries to reserve capacity for two profitable classes when this is optimal, but also by conditioning on looking one arrival ahead. It should be more logic when the *two steps look ahead* is conditioning on looking two arrivals ahead. The problem is that the computational complexity grows exponential with the number of arrivals looking ahead. A policy which make their decision by looking for every arrival an infinite steps ahead is the optimal decision and have according to section 5.8 a very high computational complexity.

An extension to press the computational complexity of the optimal policy is the use classifying. The problem of the model for the optimal policy in this report is the number of states, it expands very easily. By classifying the states, we reduce the state space but make an approximation mistake. The main goal here is also to develop a policy with a good balance between the approximation mistake and the computational complexity.

Since the decision of the *look ahead* policy is very complex, it is very hard to analyze. Therefore we are not be able to determine the optimal *look ahead* policy, the optimal δ. Once we know how this δ depends on the system parameters, we could give the computational complexity of the optimal *look ahead* policy in terms of the capacity C and the number of classes m.

The main idea of the look ahead policies is to predict a small period in the future and make the decision based on the reward differences for both decisions. Another possibility is to compare the reward per capacity. One could wonder if this leads to under-utilization of the system. But since the decision rule is only of importance when the system has almost used the capacity, especially for $\delta = 1$, it could be a good policy.

11 Appendix

11.1 Blocking Probabilities

To find the blocking probabilities we start with the most easy case, where only two types of classes are in the system and all sizes of each class k call are equal. In the next subsections the essay extends with more different types of classes and in the next section one can find the blocking-probabilities for unequal sizes.

11.1.1 Model for Equal Sizes

According to the introduction of this essay, the goal is to develop an admission control policy which maximizes the average reward rate based on looking one step ahead. This model consists of a telephone network where different types of

calls want to make use of the network but can be rejected on arrival, even when the capacity constraint is not violated. The system state $\mathbf{n} = (n_1, n_2, ..n_m)$ is the number of calls in the system of class i and m is number of different types of calls. The idea is that the policy rejects a call on arrival if the average reward rate, which is calculated with the blocking-probability, is higher than when the policy accepts this arriving call. So the blocking-probability of a more profitable class depends on the system state, the arrival rate of the more profitable class and the difference in reward rate. Although it is possible that more than just one call leaves the system in the critical time period, this does not matter. Because all size are equal, a more profitable class can only be rejected if it arrives before any other call leaves the system.

Two Classes Suppose the system that has only two types of calls where the stochastic variable X_i is the inter-arrival time of class i. Assume without loss of generality that a call arrives at $t = 0$, then the stochastic variable X_i denotes the next arrival time. The calls were assumed to follow a Poisson process, so the inter-arrival times are exponential and without loss of generality the rewards could be ordered, $r_1 > r_2$. Now the goal is to determine the blocking-probability of a more profitable class (a class one call) in some time period after accepting a class k call (in this section it is a class two call). This blocking can only happen within time period T when two events occur

1. a more profitable class i arrives in a time period T
2. a more profitable class i arrives before another call has left the system in a time period T

Denote for the probability of the first event $P(a_i, T) = P(X_i \leq T)$ and the second $P(X_i < \min(Y_1, Y2)) = P(X_i < Y_{\min})$. In this case $X_i = X_1$ because only a class two call has a more profitable class. A class one call has no other profitable classes.

To determine the probability that both events occur one can sustain by determining the probability density function of the second event and integrating over the correct domain 21. The probability density function of the second event can be found by first determining the probability density function of $Y_{\min}$. Because all Y_i ,$i \in \{1, 2, ..m\}$ are exponential independent stochastic variables with parameters μ_i, the stochastic variable $Y_{\min} = \min(Y_1, Y_2)$ is again an exponential random variable [55] with parameter $\mu_1 + \mu_2$.The probability density function of $X_1$is $f_{x_1}(x_1) = \lambda_1 \exp(-\lambda_1 x_1)$. The stochastic variables $X_1$and Y are also independent, thus the simultaneous density is the product of the marginal densities is

$$f_{x_1,y}(x_1, y) = f_{x_1}(x_1) f_y(y) \tag{67}$$

The probability that a class one call is blocked after accepting a class two call is the same as if a class one calls arrives before any departure.

$$P(BL_1^{\mathbf{n}},T) = P(X_1 < Y_{\min}, X_1 < T) = \int_{y=0}^{\infty} \int_{x_1=0}^{\min(y,T)} f_{x_1}(x_1) f_y(y) dx_1 dy \quad (68)$$

$$= \int_{x_1=0}^{\min(y,T)} \lambda_1 \exp(-\lambda_1 x_1)(\mu_1+\mu_2) \exp(-y(\mu_1+\mu_2)) dx_1 dy$$

$$= \frac{\lambda_1(1-\exp(-T(\lambda_1+\mu_1+\mu_2)))}{\lambda_1+\mu_1+\mu_2}$$

where the upper and lower bound of equation (68) follows from figure **??**. The elaboration is in appendix 11.3 equation (109).

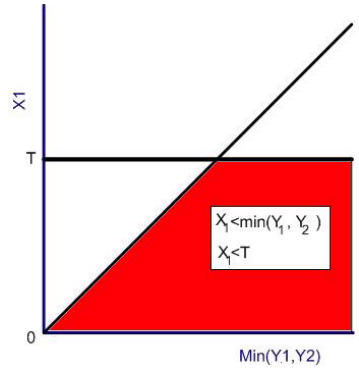

Figure 21: graph of the density function f_{x_1,y_1,y_2}

Notice that the actual departure rates also depend on the system state $\mathbf{n}$. The rate μ_1 is actually $\mu_1 n_1$, but this is done due to a shorter notation. For measuring the blocking-probability in a infinite time period the same calculations can be done

$$\lim_{T\to\infty} P(BL_{2,1},\infty) = \frac{\lambda_1}{\lambda_1+\mu_1+\mu_2} \quad (69)$$

Another way to determine $P(X_1 > Y_1 + Y_2)$ is to directly solve the simultaneous density of three of the stochastic variables.

$$P(TBL_2^{\mathbf{n}},T) = P(X_1 < Y_{\min}, X_1 < T) = \int_{y_1=0}^{\infty}\int_{y_2=0}^{\infty}\int_{x_1=0}^{\min(T,\min(y_1,y_2))} f_{x_1,y}(x_1,y_1,y_2)dx_1dy_1dy_2 \quad (70)$$

$$\lim_{T\to\infty} P(TBL_2^{\mathbf{n}},T) = P(X_1 < Y_{\min}) = \int_{y_1=0}^{\infty}\int_{y_2=0}^{\infty}\int_{x_1=0}^{\min(y_1,y_2)} f_{x_1,y_1,y_2}(x_1,y_1,y_2)dx_1dy_1dy_2$$

where the upper and lower bound from (70) follows from figure 22. For further details we refer to section 11.3 in equation (110).

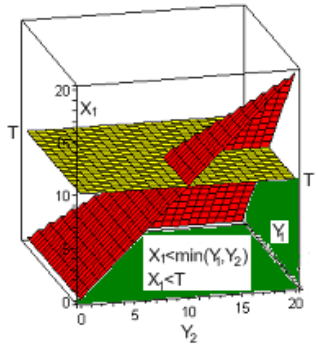

Figure 22: 3D graph if the $f_{x_1y_1y_2}$

Both calculations uses the fact the minimum of two exponential stochastic variables is again a exponential stochastic variable which parameter the sum of the parameters. When the sizes are unequal then one cannot use this. So another way to determine the same results

$$\begin{aligned} P(TBL_2^{\mathbf{n}},T) &= P(X_1 < Y_{\min}, X_1 < T) \quad (71)\\ &= \int_{y=0}^{\infty} P(X_1 < y, X_1 < T \mid Y_{\min} = y)P(Y_{\min} = y)dy \\ &= \int_{y=0}^{\infty} (1-\exp(-y(\min(\lambda_1,T))))(\mu_1+\mu_2)\exp(-y(\mu_1+\mu_2))dy \\ &= \frac{\lambda_1(1-\exp(-T(\lambda_1+\mu_1+\mu_2)))}{\lambda_1+\mu_1+\mu_2} \end{aligned}$$

Three Classes In this section there are three different types of calls. The goal is to determine the same blocking-probabilities as found in the previous section. The only difference is that sometimes there is more than just one profitable

class. If for example a class three call arrives, then the blocking-probability that decides to reject or to accept this call has to reckon with class one calls and class two calls.

First one will determine the blocking-probability when a class two arrives. The same steps are done as in the previous section. Also the same independency rules are applied here, thus the stochastic variable $Y = \min(Y_1, Y_2, Y_3)$ is again an exponential random variable with parameter $\mu_1 + \mu_2 + \mu_3$

$$\begin{aligned} P(TBL_3^{\mathbf{n}}, T) &= P(X_1 < Y_{\min}, X_1 < T) = \int_{y=0}^{\infty} \int_{x_1=0}^{\min(y,T)} f_{x_1}(x_1) f_y(y) dx_1 dy \qquad (72) \\ &= \int_{y=0}^{\infty} \int_{x_1=0}^{\min(y,T)} \lambda_1 \exp(-\lambda_1 x_1)(\mu_1+\mu_2+\mu_3) \exp(-y(\mu_1+\mu_2+\mu_3)) dx_1 dy \\ &= \frac{\lambda_1(1-\exp(-T(\lambda_1+\mu_1+\mu_2+\mu_3)))}{\lambda_1+\mu_1+\mu_2+\mu_3} \end{aligned}$$

or direct via

$$P(TBL_3^{\mathbf{n}}, T) = \int_{y_3=0}^{\infty} \int_{y_2=0}^{\infty} \int_{y_1=0}^{\infty} \int_{x_1=0}^{\min(T, y_1+y_2+y_3)} f_{y_1,y_2,y_3,x_1}(x_1, y_1, y_2, y_3) dx_1 dy_1 dy_2 dy_3 \qquad (73)$$

But for the $P(BL_3, T)$ not all the calculations are the same. Because when a class three call arrives a_3, then a first and second class call may not be rejected. The total blocking-probability is the probability that a class one is rejected or a class two call is rejected. Together the more profitable classes have a $\lambda_1 + \lambda_2$ arrival rate. So

$$\begin{aligned} P(TBL_3^{\mathbf{n}}, T) &= P(X_1 < Y_{\min}, X_1 < T \vee X_2 < Y_{\min}, X_2 < T) = P(X_{\min} < Y_{\min}, X_{\min} < T) \qquad (74) \\ &= P(\min(X_1, X_2) < \min(Y_1, Y_2, Y_3), \min(X_1, X_2) < T) \\ &= \int_{y=0}^{\infty} \int_{x=0}^{\min(y,T)} (\lambda_1+\lambda_2) \exp(-(\lambda_1+\lambda_2)x)(\mu_1+\mu_2+\mu_3) \exp(-y(\mu_1+\mu_2+\mu_3)) dx dy \\ &= \frac{(\lambda_1+\lambda_2)(1-\exp(-T(\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3)))}{\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3} \end{aligned}$$

Or

$$\begin{aligned} P(TBL_3^{\mathbf{n}}, T) &= P(X_1 < Y_{\min}, X_1 < T \vee X_2 < Y_{\min}, X_2 < T) \qquad (75) \\ &= P(X_1 < Y_{\min}, X_1 < T) + P(X_2 < Y_{\min}, X_2 < T) \\ -P(X_1 &< Y_{\min}, X_1 < T, X_2 < Y_{\min}, X_2 < T) \qquad (76) \\ &= P(BL_{3,1}, T) + P(BL_{3,2}, T) - P(X_1 < Y_{\min}, X_1 < T, X_2 < Y_{\min}, X_2 < T) \\ &= \frac{(\lambda_1+\lambda_2)(1-\exp(-T(\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3)))}{\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3} \end{aligned}$$

where the final step in equation (75) is worked out in section 11.3 equation (111). For measuring the blocking-probability in a infinite time period the same calculations can be done

$$\begin{aligned}\lim_{T\to\infty} P(TBL_3^{\mathbf{n}},T) &= P(X_1 < Y_{\min} \vee X_2 < Y_{\min}) = P(X_{\min} < Y_{\min}) \qquad (77)\\ &= \int_{y_3=0}^{\infty}\int_{y_2=0}^{\infty}\int_{y_1=0}^{\infty}\int_{x=0}^{\min(y_1+y_2+y_3)} f_{y_1,y_2,y_3,x}(x,y_1,y_2,y_3)dxdy_1dy_2dy_3\\ &= \frac{\lambda_1+\lambda_2}{\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3}\end{aligned}$$

m Classes For m classes the same results are determined as in the previous sections. The blocking-probability when a class two call arrives is

$$\begin{aligned}P(BL_2^{\mathbf{n}},T) &= P(X_1 < Y_{\min}, X_1 < T) = \int_{y=0}^{\infty}\int_{x_1=0}^{\min(y,T)} f_{x_1}(x_1)f_y(y)dx_1dy \qquad (78)\\ &\int_{y=0}^{\infty}\int_{x_1=0}^{\min(y,T)} \lambda_1 \exp(-\lambda_1 x_1)(\sum_{i=1}^{m}\mu_i)\exp(-y(\sum_{i=1}^{m}\mu_i))dx_1dy\\ &= \frac{\lambda_1(1-\exp(-T(\lambda_1+\sum_{i=1}^{m}\mu_i)))}{\lambda_1+\sum_{i=1}^{m}\mu_i}\end{aligned}$$

The blocking-probability of a more profitable class j if the policy is in state $\mathbf{n}$

$$\begin{aligned}P(BL_j^{\mathbf{n}},T) &= P(X_j < Y_{\min}, X_j < T) = \int_{y=0}^{\infty}\int_{x_j=0}^{\min(y,T)} f_{x_j}(x_j)f_y(y)dx_jdy \qquad (79)\\ &= \int_{y=0}^{\infty}\int_{x_j=0}^{\min(y,T)} \lambda_j \exp(-\lambda_j x_j)(\sum_{i=1}^{m}\mu_i)\exp(-y(\sum_{i=1}^{m}\mu_i))dx_jdy\\ &= \frac{\lambda_j(1-\exp(-T(\lambda_j+\sum_{i=1}^{m}\mu_i)))}{\lambda_j+\sum_{i=1}^{m}\mu_i}\end{aligned}$$

and the blocking-probability of any more profitable class after accepting a class k call

$$
\begin{aligned}
P(TBL_k^{\mathbf{n}}, T) &= P(X_1 < Y_{\min}, X_1 < T \vee .. \vee X_{k-1} < Y_{\min}, X_{k-1} < T) \quad (80)\\
&= P(\min_{i=1,..,k-1} X_i < Y_{\min}, \min_{i=1,..,k-1} X_i < T)\\
&= \left(\sum_{i=1}^{k-1} \lambda_i\right) \frac{(1 - \exp(-T(\sum_{i=1}^{k-1} \lambda_i + \sum_{i=1}^{m} \mu_i)))}{\sum_{i=1}^{k-1} \lambda_i + \sum_{i=1}^{m} \mu_i}
\end{aligned}
$$

and for a infinite time period

$$
\begin{aligned}
\lim_{T\to\infty} P(BL_j^{\mathbf{n}}, T) &= \frac{\lambda_j}{\lambda_j + \sum_{i=1}^{m} \mu_i} \quad (81)\\
\lim_{T\to\infty} P(TBL_k^{\mathbf{n}}, T) &= \frac{\sum_{i=1}^{k-1} \lambda_i}{\sum_{i=1}^{k-1} \lambda_i + \sum_{i=1}^{m} \mu_i}
\end{aligned}
$$

11.1.2 Model for Different Call Sizes

In the previous section every call from every class has the same size. In this section the calls have only the same size if they are from the same class. Every different class has different sizes of calls. The goal is again to maximize the expected reward rate, where also the main goal is actually to minimize the blocking probability of the more profitable classes in the system.

The big difference with the previous section where all calls have the same size is that the blocking-probability can not be determined by only looking to the first event that happened. It can happen that a call is finished before a more profitable class arrives, but can not be accepted because the capacity constraint is violated. Therefore the events that must happen for determining the blocking-probability changes.

Two Classes The goal of this model is to determine the blocking-probability of a more profitable class in a certain time period after accepting an arriving class k call. The time period could be finite or infinite. If the time period is finite then two events must happen before a more profitable class is rejected

1. A more profitable class i arrives in the time period T
2. There is not enough capacity b_i at the arrival moment of a more profitable class i

When the time period is infinite the first event always happened, because the inter-arrival time has an exponential distribution. First one determines an

expression that a more profitable class j arrives at time t.

$$f_{x_1}(x_1) = \lambda_1 \exp(-\lambda_1 x_1) \tag{82}$$

Second one determines an expression for the probability that in a time period $T \in (0, t)$ there are k units of capacity has become available. When there is only one type of class and the inter-arrival times are exponential then the number of units capacity that has become available at time t can be modelled as a Poisson Process. Assuming that the rate is constant when there are enough calls in the system

$$\begin{aligned}
P(N_1(t) = kb \qquad _1) &= \frac{(\mu_1 t)^k}{k!} \exp(-\mu_1 t) \qquad (83)\\
P(N_1(t) \leq k) &= \sum_{l_1=0}^{\lfloor k/b_1 \rfloor} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t)\\
P(N_1(t) \geq k) &= \sum_{l_1=\lceil k/b_1 \rceil}^{\infty} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t)\\
P(N_1(t) < k) = 1 - &\sum_{l_1=\lceil k/b_1 \rceil}^{\infty} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t)
\end{aligned}$$

For two type of calls

$$P(N_1(t) + N_2(t) < k) = 1 - \sum_{l_1=0}^{\infty} \sum_{l_2=\left\lceil \frac{k-b_1 l_1}{b_2} \right\rceil}^{\infty} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t) \frac{(\mu_2 t)^{l_2}}{l_2!} \exp(-\mu_2 t) \tag{84}$$

Unfortunately the events are not independent. Therefore one can use the theory [54] about the probability that two events happen.

$$P(X \in A, Y \in B) = \int_A P(Y \in B | X = x) dF_x(x) dx \tag{85}$$

Applying this theory to our problem, denote by $P(X \in A) = P(X_1 \leq T)$ and by $P(Y \in B) = P(N_1(t) + N_2(t) < b_1)$ then

$$
\begin{aligned}
P(BL_2^{\mathbf{n}},T) &= P(X_1 \leq T, N_1(t)+N_2(t) < b_1) && (86)\\
&= \int_{t=0}^{T} P(N_1(t)+N_2(t) < b_1 \mid t) f_{x_1}(t)dt && (87)\\
&= \int_{t=0}^{T} \left(1 - \sum_{l_1=0}^{\infty} \sum_{l_2=\left\lceil \frac{b_1-b_1 l_1}{b_2} \right\rceil}^{\infty} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t) \frac{(\mu_2 t)^{l_2}}{l_2!} \exp(-\mu_2 t) \right) \lambda_1 \exp(-\lambda_1 t) dt \\
&= 1\text{-}\exp(\text{-}\lambda_1 T)\text{-} \sum_{l_1=0}^{\infty} \sum_{l_2=\left\lceil \frac{b_1-b_1 l_1}{b_2} \right\rceil}^{\infty} \left(\frac{\lambda_1 (\mu_1)^{l_1} (\mu_2)^{l_2}}{l_1! l_2!} \right) \\
&\quad \left(\frac{(l_1+l_2)!\text{-} \sum_{i=0}^{l_1+l_2} \frac{(l_1+l_2)!}{i!} \exp(-T(\mu_1+\mu_2+\lambda_1))(T(\mu_1+\mu_2+\lambda_1))^i}{(\mu_1+\mu_2+\lambda_1)^{l_1+l_2+1}} \right)
\end{aligned}
$$

One can find the derivation of 86 in section 11.3 equation (112). For a infinite time period

$$
\lim_{T\to\infty} P(BL_2^{\mathbf{n}},T) = 1 - \sum_{l_1=0}^{\infty} \sum_{l_2=\left\lceil \frac{b_1-b_1 l_1}{b_2} \right\rceil}^{\infty} \frac{\lambda_1 (\mu_1)^{l_1} (\mu_2)^{l_2}}{l_1! l_2!} \frac{(l_1+l_2)!}{(\mu_1+\mu_2+\lambda_1)^{l_1+l_2+1}} \tag{88}
$$

where one can find the derivation of (88) in section 11.3 equation (**??**).

Three Classes When there are three types of calls in the system, there are some changes with the previous section. First assume that a class two call arrives. The blocking-probability can be determined as the same as in the previous section but now for three types of calls.

$$
\begin{aligned}
P(BL_2^{\mathbf{n}},T) &= P(X_1 \leq T, N_1(t)+N_2(t)+N_3(t) < b_1) = \int_{t=0}^{T} P(N_1(t)+N_2(t)+N_3(t) < b_1 \mid t) f_{x_1}(t) dt && (89)\\
&= \int_{t=0}^{T} \left(1 - \sum_{l_1=0}^{\infty} \sum_{l_2=0}^{\infty} \sum_{l_3=\left\lceil \frac{b_1-b_1 l_1-l_2 b_2}{b_3} \right\rceil}^{\infty} \frac{(\mu_1 t)^{l_1}}{l_1!} \exp(-\mu_1 t) \frac{(\mu_2 t)^{l_2}}{l_2!} \exp(-\mu_2 t) \frac{(\mu_3 t)^{l_3}}{l_3!} \exp(-\mu_3 t) \right) \lambda_1 \exp(-\lambda_1 t)\, dt \\
&\quad \left(\frac{(l_1+l_2+l_3)!\text{-} \sum_{i=0}^{l_1+l_2+l_3} \frac{(l_1+l_2+l_3)!}{i!} \exp\left(-T(\mu_1+\mu_2+\mu_3+\lambda_1)\right)(T(\mu_1+\mu_2+\mu_3+\lambda_1))^i}{(\mu_1+\mu_2+\mu_3+\lambda_1)^{l_1+l_2+l_3+1}} \right)
\end{aligned}
$$

and for a infinite time period

$$\lim_{T\to\infty} P(BL_2^{\mathbf{n}},T)=1\text{-}\sum_{l_1=0}^{\infty}\sum_{l_2=0}^{\infty}\sum_{l_3=\left\lceil\frac{b_1-b_1n_1-l_2n2}{b_3}\right\rceil}^{\infty}\frac{\lambda_1(\mu_1)^{l_1}(\mu_2)^{l_2}(\mu_3 t)^{l_3}}{l_1!l_2!l_3!}\frac{(l_1+l_2+l_3)!}{(\mu_1+\mu_2+\mu_3+\lambda_1)^{l_1+l_2+l_3+1}} \tag{90}$$

The probability that after a class three arrival a more profitable class will be rejected because there is not enough capacity left depends on the class one and class two arrivals. The blocking-probability that a class three call rejects any profitable class is the probability that a class one call or a class two call will be rejected in a certain time period.

$$\begin{aligned}
P(BL_3^{\mathbf{n}},T) &= P(X_1 \le T, N_1(t)+N_2(t)+N_3(t) < b_1 \vee X_2 \le T, N_1(t)+N_2(t)+N_3(t) < b_2) \\
&= P(X_1 \le T, N_1(t)+N_2(t)+N_3(t) < b_1) + P(X_2 \le T, N_1(t)+N_2(t)+N_3(t) < b \\
P(\max(X_1,X_2) \le T, N_1(t)+N_2(t)+N_3(t) &< \min(b_1,b_2)) \\
&= P(BL_{3,1},T) + P(BL_{3,2},T) - P(\max(X_1,X_2) \le T, N_1(t)+N_2(t)+N_3(t) < \mathrm{m} \\
&= \int_{t=0}^{\infty} P(N_1(t)+N_2(t)+N_3(t) \le b_1 \mid t) f_{x_1}(t)dt + \int_{t=0}^{\infty} P(N_1(t)+N_2(t)+N_3(t) \le
\end{aligned}$$

One can find a derivation of equation (91) in section 11.3 equation (113).

m Classes For m types the blocking-probability of a class j call in state $\mathbf{n}$ is

$$P(BL_j^{\mathbf{n}},T) = P(\sum_{i=1}^{m} N_i(t) < b_j, X_j \le T) = \int_{t=0}^{T} P(\sum_{i=1}^{m} N_i(t) < b_j \mid t) f_{x_j}(t)dt \tag{92}$$

$$=1\text{-}\exp(\text{-}\lambda_j T)\text{-}\sum_{l_1=0}^{\infty}..\sum_{l_m=\left\lceil\frac{b_j-(c-n\cdot b)-b_1 l_1-..-b_{m-1}l_{m-1}}{b_m}\right\rceil}^{\infty}\left(\frac{\lambda_j(\mu_1)^{l_1}..(\mu_m)^{l_m}}{l_1!..l_m!}\right)$$

$$\left(\frac{(l_1+..+l_m)!-\sum_{i=0}^{l_1+..+l_m}\frac{(l_1+..+l_m)!}{i!}\exp(-T(\mu_1+..+\mu_m+\lambda_j))(T(\mu_1+..+\mu_m+\lambda_j))^i}{(\mu_1+..+\mu_m+\lambda_j)^{l_1+..+l_m+1}}\right)$$

And in case of m classes of calls, we want to know the probability that none of the more profitable classes will be rejected. And in state $\mathbf{n}$ when a call arrives then the blocking-probability of a more profitable class becomes

$$\begin{aligned}
P(TBL_k^{\mathbf{n}},T) &= P(\sum_{i=1}^{m} N_i(t) < b_1, X_1 \le T \vee .. \vee \sum_{i=1}^{m} N_i(t) \le b_{k-1}, X_{k-1} \le T) \quad (93) \\
&= P(BL_1^{\mathbf{n}}) \vee P(BL_2^{\mathbf{n}}) \vee P(BL_3^{\mathbf{n}}) \vee .. \vee P(BL_k^{\mathbf{n}}) \quad (94)
\end{aligned}$$

this can be calculated with the general rule of sum. (number of terms is 2^m-1)

11.1.3 Blocking Probability with Different Call Sizes with Decreasing Rate of the Departures

The assumption that the rate is unchanged whenever a call leaves the system can only be negligible when there are enough calls in the system. But this is not always the case. Suppose after accepting a class two call more than one departures are necessary to accept a more profitable class. Suppose four departures of class three are necessary to have enough capacity available to accept this class one call. Then the rate for the first departure would be four times μ_3. But the second departure has only a rate of three times μ_3 and the last departures has a rate of one time μ_3. So the blocking-probability is higher than when assuming the rate is constant. Therefore the blocking-probability is determined with a decreasing rate of the departures.

This section starts with some definitions and notation to develop an expression for the numbers of calls that has been departed in probability in a certain time period. Finally the blocking-probability for m classes is determined.

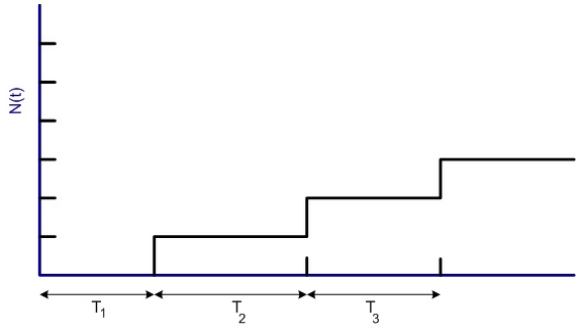

Figure 23: Example of inter-arrival times of the departures where T_i is the inter-arrival time

Denote with T_i the inter-arrival time of the i-th departure and denote with $N(t)$ the number of departures from all the classes in the system, see figure 23. The probability that three departures has occurred before time t is the same as one minus the probability that the sum of three inter-arrival times are less than t.

$$\begin{aligned} P(N(t) &\leq 2) = 1 - P(T_1 + T_2 + T_3 < t) \\ P(N(t) &\leq k) = 1 - P(T_1 + .. + T_{k-1} < t) \\ P(N(t) &< k) = 1 - P(T_1 + .. + T_k < t) \end{aligned} \tag{95}$$

this still holds when the counting process $N(t)$ has a decreasing rate. Every time when a departure occurred the rate is decreased. This is the same when T_2 has a less mean inter-arrival time as T_1. All the $T_i's$ have an exponential distribution and T_1 with mean $n_i\mu_i, T_2$ with mean $(n_i - 1)\mu_i$ and so on.

This subsection starts with only one type of class is in the system we want an expression for the probability that two departures has occurred before time t.

$$
\begin{aligned}
&P(N(t)>1) && (96)\\
&\quad = 1-P(N(t)\leq 1) = P(T_1+T_2<t) = P(T_1+T_2<t|T_1=u)P(T_1=u)\\
&\quad = \int_{u=0}^{t} P(T_2<t-u)P(T_1=u)du\\
&\quad = 1-\exp(-\mu_1(n_1-1))t + (n_1-1)\exp(-n_1\mu_1 t)\\
&P(N(t)<1) = \quad \exp(-\mu_1(n_1-1))t + (n_1-1)\exp(-n_1\mu_1 t)
\end{aligned}
$$

One can find a derivation of equation (96) in section 11.3 equation (114). The probability that three departures occurred before time t

$$
\begin{aligned}
&P(N(t)>2) && (97)\\
&\quad = 1-P(N(t)\leq 2) = P(T_1+T_2+T_3<t)\\
&\quad = P(T_1+T_2+T_3<t|T_1=u_1,T_2=u_2)P(T_1=u_1,T_2=u_2)\\
&\quad = \int_{u_1=0}^{t}\int_{u_2=0}^{t-u_1} P(T_2<t-u_1-u_2)P(T_1=u_1)P(T_2=u_2)du_1du_2\\
&\quad = 1-\frac{1}{2}n_1(n_1-1)\exp(-(n_1-2)\mu_1 t) + n_1(n_1-2)\exp(-(n_1-1)\mu_1 t)\\
&\qquad -(1+\frac{1}{2}n_1^2-\frac{3}{2}n_1)\exp(\mu_1 n_1 t)\\
&P(N(t)<2) = \quad \frac{1}{2}n_1(n_1-1)\exp(-(n_1-2)\mu_1 t) + n_1(n_1-2)\exp(-(n_1-1)\mu_1 t)\\
&\qquad -(1+\frac{1}{2}n_1^2-\frac{3}{2}n_1)\exp(\mu_1 n_1 t)
\end{aligned}
$$

the probability that k departures occurred before time t

$$
\begin{aligned}
&P(N(t)>k) && (98)\\
&\quad = 1-P(N(t)\leq k) = P(T_1+..+T_k<t)\\
&\quad = P(T_1+..+T_k<t|T_1=u_1,..,T_k=u_k)P(T_1=u_1,..,T_k=u_k)\\
&\quad = \int_{u_1=0}^{t} .. \int_{u_k=0}^{t-u_1-..-u_{k-1}} P(T_k<t-u_1-..-u_k)P(T_1=u_1)..P(T_k=u_k)du_1..du_k\\
&\quad = \int_{u_1=0}^{t} .. \int_{u_k=0}^{t-u_1-..-u_{k-1}} (1-\exp(-(n_1-k+1)(t-u_1-..-u_k))\\
&\qquad n_1\mu_1\exp(-n_1\mu_1 u_1)..(n_1-k+2)\mu_1\exp(-(n_k-k+2)\mu_1)du_1..du_k
\end{aligned}
$$

and for the final expression is not a nice closed form. In case of over 20 calls the difference between the decreasing rate and the constant rate is less than

1% for the probability that two calls has left the system. Therefore we use the assumption that the rate is constant in examples and simulations with more than 20 calls.

11.2 Policies

All the policies which could not be simulated or have too many assumptions are mentioned and sometimes discussed in this section.

11.2.1 Critical to Critical

In order to find the optimal $\alpha_{k,\mathbf{n}}$ we choose to maximize the average reward that is gained per unit time till the system state is again in a critical state. In this section one has to assume that the call sizes are all the same. A model for different sizes is too hard to calculate.

If the policy accepts the arrival then the time till the next critical state is the time till a call departures. When the call is rejected, the next event could be an arrival or a departure. The probability that the first event is an class j (with $j < k$) arrival is precisely the blocking-probability for a class j call. The critical state is reached again when a call departures. When the first event is a departure the system reaches a non critical state. Now two possible options occur. One goes back to the original definition of our policies which that in a non critical state any call will be accepted. The other option is that if the policy rejects a class k call in the critical state it also reject a class k call in the state with one unit capacity more available. Both options are considered, where the first option is called *Critical to Critical Original* and second *Critical to Critical New*. If in this state also an arrival or a departure could happen as second event. When the second event is an arrival the system reaches a critical state again and the time period is over. When another call departures the policy gains nothing and the time period is over.

Figure 24 shows the possible transitions, with only two types of class, equal unit sizes and a maximum capacity of three units. The most profitable class is class one. So a class one call will always be accepted if the capacity constraint is not violated. The class two calls are always accepted if the system is not in a critical state. The critical states are $\Omega^* = \{(2,0),(1,1),(0,2)\}$ Suppose the system is in the critical state $(2,0)$ and a class two arrives. When the policy accepts this arriving class the system state reaches $(2,1)$ and could go to $(1,1)$. When the policy reject this class two call then a the more profitable class could arrives a the system goes to state $(3,0)$. When a departure occurs the system is again in $(2,0)$ and the time period is over. When the policy rejects this call and the first event is a departure, the system state goes to $(1,0)$. After an arrival the system goes again to a critical state, suppose $(1,1)$. But when the system is in $(1,0)$ and again a departure occurs, the system is goes to the state $(0,0)$ and we consider that the time period is over and no reward has been gained. These are all the possibilities that can occur.

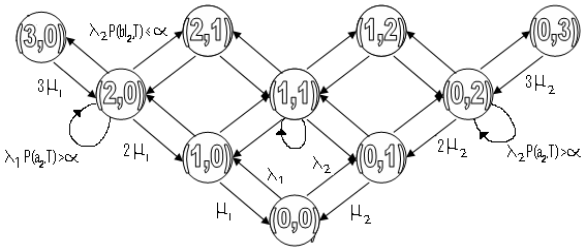

Figure 24: Example of a Markov Chain with type of classes

This policy still has the same problems as in the previous policies. If the policy rejects a class k call, does it also rejects a class $k-1$ call and so on. Hence we have to maximize over all the possibilities whether to reject or accept all the class $k-1$ calls.

When a class k call arrives and the policy C2CO or C2CN accepts then it gains per unit time

$$\frac{r_k}{\mathbf{n}\cdot\boldsymbol{\mu}} \tag{99}$$

and when the policy C2CO rejects this class k call it gains per unit time

$$\max_{j\in\{1,2,..k-1\}}\left\{\frac{\sum_{i=1}^{j}P(BL_{j,i}^{\mathbf{n}})r_i+P(1-BL_j^{\mathbf{n}})\sum_{p=1}^{m}\frac{n_p\mu_p}{\mathbf{n}\cdot\boldsymbol{\mu}}\sum_{i=1}^{m}P(BL_{j,i}^{\mathbf{n}-\mathbf{e}_p})r_i}{E\left(A^jB^m\right)_{\min}^{\mathbf{n}}+\sum_{i=1}^{j}P(BL_{j,i}^{\mathbf{n}})EB_i+P(1-BL_j^{\mathbf{n}})\sum_{p=1}^{m}\frac{n_p\mu_p}{\mathbf{n}\cdot\boldsymbol{\mu}}E\left(A^mB^m\right)_{\min}^{\mathbf{n}-\mathbf{e}_p}}\right\} \tag{100}$$

with j the number of classes which are accepted on arrival. The expected service time of a class k call $EB_i=\frac{1}{n_i\mu_i}$ and $E\left(A^jB^m\right)_{\min}^{\mathbf{n}}=\frac{1}{\lambda_1+..+\lambda_j+n_1\mu_1+..n_m\mu_m}$, EA_i $=\frac{1}{\lambda_i}$is the expected arrival time.

For the policy C2CN

$$\max_{j\in\{1,2,..k-1\}}\left\{\frac{\sum_{i=1}^{j}\left(P(BL_{j,i}^{\mathbf{n}})r_i+P(1-BL_j^{\mathbf{n}})\sum_{p=1}^{m}\frac{n_p\mu_p}{\mathbf{n}\cdot\boldsymbol{\mu}}P(BL_{j,i}^{\mathbf{n}-\mathbf{e}_p})r_i\right)}{E\left(A^jB^m\right)_{\min}^{\mathbf{n}}+\sum_{i=1}^{j}P(BL_{j,i}^{\mathbf{n}})EB_i+P(1-BL_j^{\mathbf{n}})\sum_{p=1}^{m}\frac{n_p\mu_p}{\mathbf{n}\cdot\boldsymbol{\mu}}E\left(A^jB^m\right)_{\min}^{\mathbf{n}-\mathbf{e}_p}}\right\} \tag{101}$$

For unequal size is to hard, because the available capacity is not always enough after a departure for a more profitable class with different sizes. If the policy accepts a class k arrival then the time till the next critical state is when b_k units of size departures. When the call is rejected, the next critical state is when a more profitable class j arrives before b_j units of size of other call departures and after b_j units of size departures again. But the possibilities are way too many for a fast calculation.

11.2.2 Threshold Dependency Policy

The threshold α is assumed to be dependent on the class call arrival and the system state. One could also assume that the threshold does not depend on the state of the system. In this case there is only one threshold for each different class. This means that α_k could be between zero and one. In this case the policy accepts an arriving class k call if the blocking-probability of any more profitable class is below the threshold α_k. In this manner the policy is minimizing the blocking-probability of the most profitable classes. This approach founds its applications where a considerably penalty must be paid when the policy rejects one the most profitable classes. In the next sections we consider only the cases that the thresholds does depend on the system state, $\alpha_{k,\mathbf{n}}$.

In the introduction section is considered that with the blocking-probability the average reward rate can be determined. The time period in which we determine the reward rate is based on looking one step ahead. The time period for the calculation of the blocking-probability should be logically infinite, because an arrival or departure could be in any point in time. But since the average reward rate is determined based on looking one step ahead one could also assume to calculate the blocking-probability based on an time period of looking one step ahead. Another motivation to choose a finite time period was that the calculations should become more easy and that the assumption made about the rate of departures is unchanged becomes more negligible. But when this certain time period is infinite, the calculations do not change, instead sometimes they become more easier.

The probability of blocking tends to a fixed value when the time period tends to infinity, because the probability density function of the exponential distribution is decreasing in time and the available amount of capacity is increasing in time.

In order to compare the decisions fairly, one need a good time period in which the rewards are gained. The most fair time period for comparison is the time after the decision is made till the system state returns to the same state for both decisions. This means that when a decision is made the system state could go from $\mathbf{n} \rightarrow \mathbf{n} + \mathbf{e}_i$ or to stay in $\mathbf{n}$ and when the system is in the same state again the time period is over. But to determine the average reward that is gained in such a time period is very hard to calculate. Therefore we choose some other time periods in which the policy gains the rewards. The start time of the period is always the moment at the decision epoch.

In order to maximize the total reward gained from the accepted calls one can apply the policy that is achieved in the previous sections.

$$d_k(\mathbf{n}) = 1 \Leftrightarrow \ (\mathbf{n} + \mathbf{e}_k) \cdot \mathbf{b} \leq \min_{\{i=1,2,..,k-1)} C_i \ \cap P(BL_k, T) \leq \alpha_k \qquad (102)$$

11.2.3 Look Ahead Policy (Equal Size)

In order to make the optimal decision, we choose to maximize the average reward that is gained per unit time. The average reward rate can be gained by the first two arrivals only, which would be accepted if the capacity constraint is not violated. The time period in which the reward can be gained starts at the decision epoch till the accepted arrivals have been departed.

If the policy rejects the arriving call then the next arrival could always be accepted. The problem here is that the policy does not know if classes $2, 3, .., k+1$ must be rejected too. Therefore one has to split this problem in $k-1$ smaller problems. First we pretend that we *accept all* classes up to class $k-1$ and determine the average reward rate. Then one set class $k-1$ to always reject and determine again the average reward rate. In this way one can find the maximum average reward rate after rejecting the arriving class k call.

Still the policy can always accepts all the classes in a non critical state, case 2.a or rejects all the classes $j, ..m$ in a non critical state, case 2.b. In case 2.a all the sizes are equal.

So the profit that is gained can be determined for each policy in $m-k+k-1$ steps. Thus if a class k call arrives in critical state $\mathbf{n}$ then the optimal average reward rate when policy 2.a accepts is with equal sizes is the nominator

$$r_k+(1-P(BL_j^{\mathbf{n}+\mathbf{e}_k},\infty))\left(\sum_{p=1}^{m}\frac{(n+e_k)_p\mu_p}{(n+e_k)\cdot\mu}\sum_{i=1}^{j}P(BL_{j,i}^{\mathbf{n}+\mathbf{e}_k-\mathbf{e}_p},\infty)r_i+\left(\sum_{p=1}^{m}\frac{(n+e_k)_p\mu_p}{(n+e_k)\cdot\mu}(1-P(BL_j^{\mathbf{n}+\mathbf{e}_k-\mathbf{e}_p},\infty)\right.\right. \tag{103}$$

and the denominator

$$E\max\left\{B_k,\left(A^jB^m\right)_{\min}^{\mathbf{n}+\mathbf{e}_k}+(1-P(BL_j^{\mathbf{n}+\mathbf{e}_k},\infty))\left(\begin{array}{r}\sum_{p=1}^{m}\frac{(n+e_k)_p\mu_p}{(n+e_k)\cdot\mu}\left(\left(A^jB^m\right)_{\min}^{\mathbf{n}+\mathbf{e}_k-\mathbf{e}_p}+\sum_{i=}\right.\right.\\ +\left(\sum_{p=1}^{m}\frac{(n+e_k)_p\mu_p}{(n+e_k)\cdot\mu}(1-P(BL_j^{\mathbf{n}+\mathbf{e}_k-\mathbf{e}_p},\infty))\right)\sum_{q=1}^{m}\sum_{p=1}^{m}\frac{(n+e_k-e\end{array}\right. \tag{104}$$

where we have to take $\max_{j\in\{m,m-1,..,k\}}$of both parts. The $P(BL_j^{\mathbf{n}},\infty))$ is probability that the first arrival is blocked in a infinite time period because the capacity constraint is violated. One minus $P(BL_j^{\mathbf{n}},\infty))$ is the probability that the first arrival is not blocked because of the capacity constraint or the probability that first event is a departure. The term $E\left(A^jB^m\right)_{\min}^{\mathbf{n}}$ $=\frac{1}{\lambda_1+..+\lambda_j+\mathbf{n}\cdot\boldsymbol{\mu}}$, which means the average minimum time before any arrival of the classes $1, .., j$ or any departure for the classes $1, .., m$ occurs. And where $P(BL_{j,i}^{\mathbf{n}+\mathbf{e}_k-\mathbf{e}_p},\infty)=P(BL_j^{\mathbf{n}+\mathbf{e}_k},\infty)\frac{\lambda_i}{\lambda_1+..+\lambda_j}$

Nominator:

1. Gains r_k units because of the acceptance of the arrival
2. The policy gains $r_1, ..r_j$ if the first event is a departure (need some capacity) then the arrival of the accepting arrivals takes place

3. The policy gains $r_1, ..r_m$ if the first two events are departures (not in critical state anymore) then any arrival will be accepted

Denominator:

1. If the first arrival has the longest service time needed then this is the cycle length
2. if the second arrival has the longest service time plus time till arrival is longer then the first arrival then this is the cycle length
3. if (2) then the system has to wait till the first event (arrival of the classes $k+1, .., m$ are not events, these are not seen by the system)
4. If first event is arrival then if blocked cycle length is over if accepted cycle length is then till arrival departures
5. if first event is departure then the system has to wait till next event (arrival of the classes $k+1, .., m$ are not events, these are not seen by the system)
6. if (5) and second event is arrival then the system has to wait till its departure and cycle length is over
7. if (5) and second event is departure then the system has to wait till any arrival and its departure and cycle length is over

The cycle length can be calculated with the results from 105

$$E[\max(X,Y)] = E[X|X>Y]P(X>Y) + E[Y|Y>X]P(Y>X) \quad (105)$$

where the stochastic variable X denotes the service time of the first accepted arrival and Y of the second one. Equation (105) is worked out in section 11.3 equation (**??**). Note that Y is a mix of an Erlang distribution and a hyper exponential distribution. Denote by $\left(A^j B^m\right)_{\min}^{\mathbf{n}+\mathbf{e}_k} = Z$, $(1-P(BL_j^{\mathbf{n}+\mathbf{e}_k}, \infty) = p$, then Y can be expressed in terms of p, Z and B_i. But when $\mathbf{n}$ is large enough, then one departure does not change the rate of the other departures, hence it can be neglected.

and when the policy rejects a class k call it does the same as case 1.

and when policy 1.a equal size rejects a class k call

$$\max_{j\in\{k-1,k-2,..,1\}} \left\{ \frac{(1-P(BL_k^{\mathbf{n}},\infty)) \sum_{i=1}^{j} \frac{\lambda_i}{\lambda_1+..+\lambda_j} r_i + P(BL_k^{\mathbf{n}},\infty) \sum_{i=1}^{m} \frac{\lambda_i}{\lambda_1+..+\lambda_}}{E\left(A^j B^m\right)_{\min} + \frac{\lambda_1+..+\lambda_j}{\lambda_1+..+\lambda_j+\mu_1+..+\mu_m} \sum_{i=1}^{j} \frac{\lambda_i}{\lambda_1+..+\lambda_j} EB_i + \frac{\mu_1+..+\mu_m}{\lambda_1+..+\lambda_j+\mu_1+..+\mu_m} \left((EA^m_{\min} | EA_1 > EB^m_{\mathrm{m}} \right.} \right. \quad (106)$$

where the nominator denotes the average reward and the denominator denotes the average time period. This result needs some clarification. First the nominator denotes the average reward that is gained by the policy if it rejects all classes $j \in \{k-1, k-2, .., 1\}$.

For unequal sizes the cycle length must end when the first call arrives (after rejection).

$$\frac{\sum_{i=1}^{m} \frac{\lambda_i}{\lambda_1+..\lambda_m} \left(1 - P(BL_i^{\mathbf{n}})\right) r_i}{EA_{\min}^{\mathbf{n}} + \sum_{i=1}^{m} \frac{\lambda_i}{\lambda_1+..\lambda_m} \left(1 - P(BL_i^{\mathbf{n}})\right) EB_i} \tag{107}$$

$$\begin{aligned} E\left[\min(X_1, X_2, X_3) | X_1 > Y, X_2 > Y\right] &= \qquad (108) \\ E\left[X_1 | X_1 > Y, X_2 > Y, X_1 < \min(X_2, X_3)\right] P(X_1 &< \min(X_2, X_3) | X_1 > Y, X_2 > Y) \\ + E\left[X_2 | X_1 > Y, X_2 > Y, X_2 < \min(X_1, X_3)\right] P(X_2 &< \min(X_1, X_3) | X_1 > Y, X_2 > Y) \\ + E\left[X_3 | X_1 > Y, X_2 > Y, X_3 < \min(X_1, X_2)\right] P(X_3 &< \min(X_1, X_2) | X_1 > Y, X_2 > Y) \end{aligned}$$

11.3 Elaborations

The elaboration of equation (**??**)

$$\begin{aligned} &= \int_{y=0}^{\infty} \int_{x_1=0}^{\min(y,T)} \lambda_1 \exp\left(-\lambda_1 x_1\right)(\mu_1+\mu_2) \exp\left(-y(\mu_1+\mu_2)\right) dx_1 dy \qquad (109) \\ &= \int_{y=0}^{\infty} \left(-\exp\left((-\lambda_1 \min\left(y, T\right)\right) + 1\right)(\mu_1+\mu_2) \exp\left(-y(\mu_1+\mu_2)\right) dy \\ &= \int_{y=0}^{T} (\mu_1+\mu_2) \exp\left(-y(\mu_1+\mu_2) - (\mu_1+\mu_2) \exp\left(-y(\mu_1+\mu_2+\lambda_1)\right) dy \\ &\quad + \int_{y=T}^{\infty} \left(-\exp\left(-\lambda_1 T\right) + 1\right)(\mu_1+\mu_2) \exp\left(-y(\mu_1+\mu_2)\right) dy \\ &= -\exp\left(-T(\mu_1+\mu_2) + \frac{(\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)} \exp\left(-T(\mu_1+\mu_2+\lambda_1)\right) + 1 - \frac{(\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)}\right. \\ &\quad \left. + \exp\left(-\lambda_1 T\right)\right) \exp\left(-T(\mu_1+\mu_2)\right) + \exp\left(-T(\mu_1+\mu_2)\right) \\ &= \exp\left(-T(\mu_1+\mu_2+\lambda_1)\right) \left[\frac{(\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)} + 1\right] + \frac{(\lambda_1+\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)} - \frac{(\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)} \\ &= \frac{\lambda_1(1 - \exp(-T((\lambda_1+\mu_1+\mu_2)}{(\lambda_1+\mu_1+\mu_2)} \end{aligned}$$

The elaboration of equation (**??**)

$$\int_{y_1=0}^{\infty}\int_{y_2=0}^{\infty}\int_{x_1=0}^{\min(T,\min(y_1,y_2))} f_{x_1,y}(x_1,y_1,y_2)dx_1dy_1dy_2 \tag{110}$$

$$= \int_{y_1=0}^{\infty}\int_{y_2=0}^{\infty}\int_{x_1=0}^{\min(T,\min(y_1,y_2))} \lambda_1\exp(-\lambda_1 x_1)\mu_1\exp(-y_1\mu_1)\mu_2\exp(-y_2\mu_2)dx_1dy_1dy_2$$

$$= \int_{y_1=0}^{\infty}\int_{y_2=0}^{\infty} [-\exp(-\lambda_1\min(T,y_1,y_2)+1]\,\mu_1\exp(-y_1\mu_1)\mu_2\exp(-y_2\mu_2)dy_1dy_2$$

$$= \int_{y_1=0}^{T}\int_{y_2=0}^{y_1} (1-\exp(-\lambda_1 y_2)\mu_1\exp(-y_1\mu_1)\mu_2\exp(-y_2\mu_2)dy_1dy_2$$

$$+ \int_{y_2=0}^{T}\int_{y_1=0}^{y_2} (1-\exp(-\lambda_1 y_1)\mu_1\exp(-y_1\mu_1)\mu_2\exp(-y_2\mu_2)dy_1dy_2$$

$$+ \int_{y_2=T}^{\infty}\int_{y_1=T}^{\infty} (1-\exp(-\lambda_1 T)\mu_1\exp(-y_1\mu_1)\mu_2\exp(-y_2\mu_2)dy_1dy_2$$

The elaboration of 75

$$= P(X_1<Y_{\min}, X_1<T) + P(X_2<Y_{\min}, X_2<T) - P(X_1<Y_{\min}, X_1<T, X_2<Y_{\min}, X_2<T)$$

$$= \int_{y=0}^{\infty} P(X_1<y, X_1<T \mid Y_{\min}=y)dy + \int_{y=0}^{\infty} P(X_2<y, X_2<T \mid Y_{\min}=y)dy - \int_{y=0}^{\infty} P(\max(X_1,X_2)<\min(y,T$$

$$= \int_{y=0}^{\infty} (1-\exp(-y(\min(\lambda_1,T))))(\mu_1+\mu_2)\exp(-y(\mu_1+\mu_2))dy + \int_{y=0}^{\infty} (1-\exp(-y(\min(\lambda_2,T))))(\mu_1+\mu_2)\exp($$

$$- \int_{y=0}^{\infty} (1-\exp(-y(\min(\lambda_1,T))))(1-\exp(-y(\min(\lambda_2,T))))(\mu_1+\mu_2)\exp(-y(\mu_1+\mu_2))dy$$

$$= \frac{(\lambda_1+\lambda_2)(1-\exp(-T(\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3)))}{\lambda_1+\lambda_2+\mu_1+\mu_2+\mu_3}$$

The elaboration of equation (86)

$$\int_{t=0}^{T}\left(1-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{k-b_1l_1}{b_2}\right\rceil}^{\infty}\frac{(\mu_1 t)^{l_1}}{l_1!}\exp(-\mu_1 t)\frac{(\mu_2 t)^{l_2}}{l_2!}\exp(-\mu_2 t)\right)\lambda_1\exp(-\lambda_1 t)dt \tag{112}$$

$$= \int_{t=0}^{T}(\lambda_1\exp(-\lambda_1 t))\,dt-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{k-b_1l_1}{b_2}\right\rceil}^{\infty}\frac{\lambda_1(\mu_1)^{l_1}(\mu_2)^{l_2}}{l_1!l_2!}\int_{t=0}^{T}\left(t^{l_1+l_2}\exp(-t(\mu_1+\mu_2+\lambda_1))\right)dt$$

$$= 1-\exp(-\lambda_1 T)-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{k-b_1l_1}{b_2}\right\rceil}^{\infty}\left(\frac{\lambda_1(\mu_1)^{l_1}(\mu_2)^{l_2}}{l_1!l_2!}\right)$$
$$\left(\frac{(l_1+l_2)!-\sum_{i=0}^{l_1+l_2}\frac{(l_1+l_2)!}{i!}\exp(-T(\mu_1+\mu_2+\lambda_1))(T(\mu_1+\mu_2+\lambda_1))^i}{(\mu_1+\mu_2+\lambda_1)^{l_1+l_2+1}}\right)$$

And for infinite time

$$= \int_{t=0}^{\infty}(\lambda_1\exp(-\lambda_1 t))\,dt-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{b_1-b_1l_1}{b_2}\right\rceil}^{\infty}\frac{\lambda_1(\mu_1)^{l_1}(\mu_2)^{l_2}}{l_1!l_2!}\int_{t=0}^{\infty}\left(t^{l_1+l_2}\exp(-t(\mu_1+\mu_2+\lambda_1))\right)dt$$

$$= 1-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{b_1-b_1l_1}{b_2}\right\rceil}^{\infty}\frac{\lambda_1(\mu_1)^{l_1}(\mu_2)^{l_2}}{l_1!l_2!}\frac{(l_1+l_2)!}{(\mu_1+\mu_2+\lambda_1)^{l_1+l_2+1}}$$

The elaboration of equation (91)

$$\begin{aligned}P(X_1 &\leq T, N_1(t)+N_2(t)+N_3(t)<b_1)+P(X_2\leq T, N_1(t)+N_2(t)+N_3(t)<b_2)-P(X_1\leq T, N_1(t)+N\\ &= P(X_1\leq T, N_1(t)+N_2(t)+N_3(t)<b_1)+P(X_2\leq T, N_1(t)+N_2(t)+N_3(t)<b_2)-P(\max(X_1,\\ &= \int_{t=0}^{T}P(N_1(t)+N_2(t)+N_3(t)<b_1)\mid t)f_{x_1}(t)dt+\int_{t=0}^{T}P(N_1(t)+N_2(t)+N_3(t)<b_2\mid t)f(t)dt-\int_{t=0}^{T}\\ &\quad -\int_{t=0}^{T}\left(1-\sum_{l_1=0}^{\infty}\sum_{l_2=\left\lceil\frac{\min(b_1,b_2)-b_1l_1}{b_2}\right\rceil}^{\infty}\frac{(\mu_1 t)^{l_1}}{l_1!}\exp(-\mu_1 t)\frac{(\mu_2 t)^{l_2}}{l_2!}\exp(-\mu_2 t)\right)\lambda_1\exp(-\lambda_1 t)(1-\exp(-\lambda_2\end{aligned}$$

The elaboration of a λ_i is the arrival rate of the more profitable class given that it has arrived before any other departure (j classes) is

$$= \int_{y=0}^{\infty} \left(\sum_{j=1}^{m} \mu_j \exp(-\sum_{j=1}^{m} \mu_j y) \right) \left[-\frac{(1+\lambda_i x_i)\exp(-\lambda_i x_i)}{\lambda_i} \right]_0^y dy$$

$$= \int_{y=0}^{\infty} \left(\sum_{j=1}^{m} \mu_j \exp(-\sum_{j=1}^{m} \mu_j y) \right) \left[\frac{1-\exp(-\lambda_i y) - \lambda_i y \exp(-\lambda_i y)}{\lambda_i} \right] dy$$

$$= \int_{y=0}^{\infty} \frac{\sum_{j=1}^{m} \mu_j}{\lambda_i} \left(\exp(-\sum_{j=1}^{m} \mu_j y) - \exp(-\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right) y) - \lambda_i y \exp(-\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right) y) \right) dy$$

$$= \frac{\sum_{j=1}^{m} \mu_j}{\lambda_i} \left[-\frac{\exp(-\sum_{j=1}^{m} \mu_j y)}{\sum_{j=1}^{m} \mu_j} + \frac{\exp(-\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right) y)}{\lambda_i + \sum_{j=1}^{m} \mu_j} + \frac{\lambda_i(\sum_{j=1}^{m} \mu_j y + 1 + \lambda_i y)\exp(-\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right) y)}{\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2} \right]^{y=}$$

$$= \frac{\sum_{j=1}^{m} \mu_j}{\lambda_i} \left[-0 + \frac{1}{\sum_{j=1}^{m} \mu_j} - \frac{1}{\lambda_i + \sum_{j=1}^{m} \mu_j} - \frac{\lambda_i}{\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2} \right] = \frac{1}{\lambda_i} - \frac{\sum_{j=1}^{m} \mu_j}{(\lambda_i + \sum_{j=1}^{m} \mu_j)\lambda_i} - \frac{\lambda_i \sum_{j=1}^{m} \mu_j}{\lambda_i \left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2}$$

$$= \frac{\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2 - \sum_{j=1}^{m} \mu_j \left(\lambda_i + \sum_{j=1}^{m} \mu_j\right) - \lambda_i \sum_{j=1}^{m} \mu_j}{\lambda_i \left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2}$$

$$= \frac{\lambda_i^2 + 2\lambda_i \sum_{j=1}^{m} \mu_j + \left(\sum_{j=1}^{m} \mu_j\right)^2 - \lambda_i \sum_{j=1}^{m} \mu_j - \left(\sum_{j=1}^{m} \mu_j\right)^2 - \lambda_i \sum_{j=1}^{m} \mu_j}{\lambda_i \left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2}$$

$$= \frac{\lambda_i^2}{\lambda_i \left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2} = \frac{\lambda_i}{\left(\lambda_i + \sum_{j=1}^{m} \mu_j\right)^2}$$

elaboration of

$$
\begin{aligned}
P(T_1 + T_2 \quad &< \quad t) = P(T_1 + T_2 < t | T_1 = s) P(T_1 = s) \qquad (114)\\
&= \int_{s=0}^{t} P(T_2 < t - s) P(T_1 = s) ds \\
&= \int_{s=0}^{t} (1 - \exp(-(n_1 - 1)\mu_1)(t - s)) n_1 \mu_1 \exp((-n_1\mu_1)s) ds \\
&= \int_{s=0}^{t} (1 - \exp(-(n_1 - 1)\mu_1 t) \exp((n_1 - 1)\mu_1 s))) n_1 \mu_1 \exp(-n_1\mu_1 s) ds \\
&= \int_{s=0}^{t} n_1\mu_1 \exp(-n_1\mu_1 s) - \exp(-(n_1 - 1)\mu_1 t) \exp((n_1 - 1)\mu_1 - (n_1\mu_1))s) ds \\
&= \int_{s=0}^{t} n_1\mu_1 \exp(-n_1\mu_1 s) - \exp(-(n_1 - 1)\mu_1 t) \exp(-\mu_1 s) ds \\
&= 1 - \exp(-\mu_1(n_1 - 1))t + (n_1 - 1)\exp(-n_1\mu_1 t)
\end{aligned}
$$

elaboration of (105)

$$
\begin{aligned}
E \max\{X, Y\} \quad &= \qquad (115)\\
&\int_{y=0}^{\infty} \int_{x=0}^{\infty} \max(x, y) f_x(x) f_y(y) dx dy \\
&= \int_{y=0}^{\infty} \int_{x=0}^{y} y\lambda \exp(-\lambda x)\mu \exp(-\mu y) dx dy + \int_{y=0}^{\infty} \int_{y=x}^{\infty} x\lambda \exp(-\lambda x)\mu \exp(-\mu y) dx dy \\
&= \int_{y=0}^{\infty} \left[y \left(1 - \exp(-\lambda x)\right) \mu \exp(-\mu y) \right]_{x=0}^{x=y} dy + \int_{y=0}^{\infty} \left[x \left(1 - \exp(-\lambda x)\right) \mu \exp(-\mu y) \right]_{x=y}^{x=\infty} dy \\
&= \int_{y=0}^{\infty} \left[y \left(1 - \exp(-\lambda y)\right) \mu \exp(-\mu y) \right] dy + \int_{y=0}^{\infty} - \left[y \left(1 - \exp(-\lambda y)\right) \mu \exp(-\mu y) \right] dy \\
&= \frac{\lambda^2 + \mu^2 + \lambda\mu}{\lambda\mu(\lambda + \mu)}
\end{aligned}
$$

11.4 Codes

11.4.1 Maple Code

11.4.2 C++ code

11.4.3 Excel Sheets

12 References

References

[1] control for broadband services: Stochastic knapsack with advance information (1996) *European Journal of Operational Research, 89 (1-2), pp. 127-134*

[2] Chlebus, E., A. Coyle, W. Henderson Mean-value Analysis for Examining Call Admission Control Thresholds in Multi-service Networks (1994) *Conference/Journal: 14th International Teletraffic Congress, Antibes Juan Les Pins, France*

[3] Choi, Jihyuk, Kwon, Taekyoung, Choi, YangheeAltman, E., Jiménez, T., Koole, G. On optimal call admission control in a resource-sharing system (2001) *IEEE Transactions on Communications 49 (9), pp. 1659-1668*

[4] Beigy, H., Meybodi, M.R. A general call admission policy for next generation wireless networks (2005) *Computer Communications 28 (16), pp. 1798-1813*

[5] Beigy, H., Meybodi, M.R A two-threshold guard channel scheme for minimizing blocking probability in communication networks (2004) *International Journal of Engineering, Transactions B: Applications 17 (3), pp. 245-262*

[6] Beigy, H., Meybodi, M.R. An adaptive call admission algorithm for cellular networks (2005) *Computers and Electrical Engineering 31 (2), pp. 132-151*

[7] Beigy, H., Meybodi, M.R. Multi-threshold Guard Channel Policy for Next Generation Wireless Networks (2003) *Lecture Notes in Computer Science, 2869, pp. 755-762.*

[8] Bertsimas, D., Chryssikou, T. Bounds and policies for dynamic routing in loss networks (1999) *Operations Research, 47 (3), pp. 379-394.*

[9] Chen, Ing-Ray, Chen, Chi-Ming Threshold-Based Admission Control Policies for Multimedia Servers (1996) *Conference/Journal: Computer Journal 39*

[10] Chiu S.Y., Lu L., Cox Jr. L.A. Optimal access , Naghshineh, Mahmoud Call admission control for multimedia services in mobile cellular networks: A Markov decision approach (2000) *IEEE Symposium on Computers and Communications - Proceedings, pp. 594-599*

[11] Cosyn, J., Sigman, K. Stochastic Networks: Admission and Routing Using Penalty Functions (2004) *Queueing Systems 48 (3-4), pp. 237-262*

[12] Das, S., Ghosh, D. Binary knapsack problems with random budgets (2003) *Journal of the Operational Research Society 54 (9): 970-983*

[13] Dean, B.C., Goemans, M.X., Vondrák, J. Approximating the stochastic knapsack problem: The benefit of adaptivity (2004) *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS, pp. 208-217.*

[14] Dziong Zbigniew, Mason Lorne An analysis of near optimal call admission and routing model for multi-service loss networks (1992) *Proceedings - IEEE INFOCOM, 1, pp. 141-152*

[15] Elhedhli, Samir Exact Solution of a class of nonlinear knapsack problems (2005) *Operation Research Letters 33 pp. 316-324*

[16] Ezziane Z Solving the 0/1 knapsack problem using an adaptive genetic algorithm (2002) *AI Edam-Artificial Intelligence for Engineering Design Analysis and Manufacturing 16 (1): 23-30*

[17] Gavious, Arieh, Rosberg, Zvi Restricted complete sharing policy for a stochastic knapsack problem in B-ISDN (1994) *IEEE Transactions on Communications, 42 (7), pp. 2375-2379.*

[18] Gibbens, Richard J., Kelly, Frank P., Key, Peter B. A Decision-Theoretic Approach to Call Admission Control in ATM Networls (1995) *IEEE Journal on Selected areaur in Communications, Volume 13, No. 6, pp. 1101-1114*

[19] Grimaldi, Ralph P. Discrete and combinatoral mathematics 5th ed. (1999) *Pearson Addison Wesley, pp. 27*

[20] Hunt, P.J., Laws, C.N. Asymptotically optimal loss network control (1993) *Mathematical Operations Research, 18, pp. 880-900.*

[21] Hunt, P.J., Laws, C.N. Optimization via trunk reservation in single resource loss systems under heavy traffic (1997) *Annals of Applied Probability, 7 (4), pp. 1058-1079.*

[22] Iyengar, G., Sigman, K. Exponential penalty function control of loss networks (2004) *The Annals of Applied Probability, 14 (4), pp. 1698-1740*

[23] Karmankar, Uday S., Yoo, Jinsung Stochastic dynamic product cycling problem (1994) *European Journal of Operation Research Volume 73, Issue 2, 1994, pp. 360-373*

[24] Kelly, F.P. Loss Network (1991) *The Annals of Applied Probability, Volume 1, No.3.,pp 319-378*

[25] Kleywegt, A.J., Papastavrou, J.D. The Dynamic and Stochastic Knapsack Problem (1998) *Operations Research, 46 (1), pp. 17-35.*

[26] Kleywegt, A.J., Papastavrou, J.D. The dynamic and stochastic knapsack problem with random sized items (2001) *Operations Research, 49 (1), pp. 26-41*

[27] Knightly, Edward W., Shroff, Ness B. Admission Control for Statistical QoS: Theory and Practice (1999) *Volume 13, Issue 2, Pages 20-29*

[28] Koole, Ger Stochastic scheduling with event-based dynamic programming (2000) *Mathematical Methods Operation Research 51 pp. 249-261*

[29] Koole, Ger Structural results for the control of queueing systems using event-based dynamic programming (1998) *Queueing Systems 30 pp. 323-339*

[30] Ku, C.-Y., Yen, D.C., Chang, I.-C., Huang, S.-M., Jordan, S. Near-optimal control policy for loss networks (2006) *Omega 34 (4), pp. 406-416*

[31] Ku, C.-Y., Jordan, S. Near optimal admission control for multi server loss queues in series (2003) *European Journal of Operations Research 144 (1), pp. 166-178*

[32] Lee, Tae-Eog, Geun Tae Oh The asymptotic value-to-capacity ratio for the multi-class stochastic knapsack problem (1997) *European Journal of Operations Research 103 pp. 584-594*

[33] Lin K.Y., Ross S.M. Admission control with incomplete information of a queueing system (2003) *Operations Research, 51 (4), pp. 645-654*

[34] Lin K.Y., Ross S.M. Optimal admission control for a single-server loss queue (2004) *Journal of Applied Probability, 41 (2), pp. 535-546*

[35] Lippman, Steven A. Applying a New Device in the Optimization of Exponential Queueing Systems (1975) *Operations Research, Vol.23, No.4.,pp.687-710*

[36] Marchetti-Spaccamela, A., Vercellis, C. Stochastic on-line knapsack problems (1995) *Conference/Journal: Mathematical Programming: Series A and B*

[37] Nasser, N. Stochastic decision-based analysis of admission control policy in multimedia wireless networks (2005) *Lecture Notes in Computer Science 3462: 1281-1296*

[38] Nawijn, M. Wim Look Ahead Policies for Admission to a Single Server Loss System (1990) *Operations Research, Vol. 38, No. 5, pp. 854-862*

[39] Ni, J., Tsang, D.H.K., Tatikonda, S., Bensaou, B. Threshold and reservation based call admission control policies for multiservice resource-sharing systems (2006) *Operations Research 54 (1), pp. 11-25*

[40] Ohmikawa M, Takagi H, Kim SY Optimal call admission control for voice traffic in cellular mobile communication networks (2005) *IEICE Transactions on Fundamentals of Electronics Communication and Computer Sciences E88A (7), pp. 1809-1815*

[41] Örmeci, E.L., Van Der Wal, J. Admission policies for a two class loss system with general inter-arrival times (2006) *Stochastic Models 22 (1), pp. 37-53*

[42] Örmeci, E.L., Burnetas, A. Dynamic admission control for loss systems with batch arrivals (2004) *Advances in Applied Probability 37 (4), pp. 915-937*

[43] Örmeci, E. Lerzan; Burnetas, Apostolos N.; Emmons, Hamilton Dynamic policies of admission to a two-class system based on customer offers (2002) *IEEE Transactions on Communications, 34 (9) pp. 813-822 (10)*

[44] Ramjee R., Towsley D., Nagarajan R. On optimal call admission control in cellular networks (1997) *Wireless Networks, 3 (1), pp. 29-41*

[45] Tsang, Danny H.K. Ross, Keith.W. Algorithms to determine exact blocking probabilities for multiratetree networks (1990) *IEEE Transactions on Communications, 38 (8) pp. 1266-1271*

[46] Ross, Keith W., Yao, David D. Monotonicity properties for the stochastic knapsack (1990) *IEEE Transactions on Information Theory, 36 (5), pp. 1173-1179.*

[47] Ross, Keith W. Multiserice Loss Models for Broadband Telecommunication Networks (1995) *Springer, New York*

[48] Ross, Keith W., Tsang, Danny H.K. Optimal circuit access policies in an ISDN environment: A Markov decision approach (1989) *IEEE Transactions on Communications, 37 (9), pp. 934-939.*

[49] Ross, Keith W., Tsang, Danny H.K. Stochastic knapsack problem (1989) *IEEE Transactions on Communications, 37 (7), pp. 740-747.*

[50] Saquib, Mohammad, Yates, Roy Optimal call admission to a mobile cellular network (1995) *IEEE Vehicular Technology Conference, 1, pp. 190-194*

[51] Taylor Howard, M., Karlin, Samual Introduction to Stochatic Modeling (1998) *Acadamic Press*

[52] Yin, Z., Xie, J. Admission Control scheme for multi-class services in QoS-based mobile cellular networks (2004) *School of Electronics and Information Technology, Shang hai Jiaotong University, Shanghai 200030, China*

[53] Ziedins I. Optimal admission controls for Erlang's loss system with phase-type arrivals (1996) *Probability in the Engineering and Informational Sciences, 10 (3), pp. 397-414*

[54] Dictaat Kansrekening pp. 71

[55] Dictaat Queueing Theory

[56] http://mathworld.wolfram.com/IncompleteGammaFunction.html