UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Distributed Scheduling with Active Idleness

A Key to Multiclass Queuing Networks Stabilisation

José António Alves Moreira
(Licenciado em Eng. Electrotécnica e de Computadores)

Dissertação para a obtenção do Grau de Mestre em Eng. Electrotécnica e de Computadores

DOCUMENTO PROVISÓRIO

Resumo

Esta tese apresenta um mecanismo de supervisão que é capaz de estabilizar qualquer rede de filas de espera, não acíclica, com múltiplas classes e com uma política de sequenciamento distribuída. É necessário que a *Condição de Intensidade de Tráfego* seja respeitada assim como algumas restrições menores nas distribuições de serviço e de chegada de clientes. Este mecanismo é baseado no conceito de *Inactividade Activa*. Este conceito representa a capacidade do sequenciador manter o servidor inactivo mesmo na presença de clientes à espera de serem processados. Esta capacidade pode parecer um desperdício de recursos, mas o importante é olhar para a rede de filas de espera como um todo. O que parece ser uma perda de recursos para um servidor, pode de facto ser extremamente benéfico para o desempenho de toda a rede de filas de espera.

A tese apresenta uma possível implementação deste conceito denominada *Controlador de Janela Temporal* e ilustra não só a sua capacidade para estabilizar, mas também para melhorar o desempenho de algumas redes de filas de espera, estáveis, não acíclicas e com múltiplas classes, independentemente da medida de desempenho em consideração.

Palavras-chave: Redes de Filas de Espera, Sequenciamento Distribuído, Estabilidade, Políticas "Idling", Inactividade Activa.

Abstract

This thesis presents a supervisory mechanism that is able to stabilize any non-acyclic, stochastic, multiclass queuing network with a distributed scheduling policy. The *Traffic Intensity Condition* has to hold as well as some mild assumptions on the service and arrival distributions.

This mechanism is based on the *Active Idleness* concept. By*Active Idleness* it is meant the scheduler's ability to force inactivity on the server, even in the presence of work. Although this ability appears as leading to a waste of the available resources, the important factor is to look at the queuing network as a whole. What might appear as a loss of resources when looking to an individual server, could in fact be highly beneficial for the performance of the entire queuing network.

The thesis presents one possible implementation of the concept, termed *Time Window Controller*, and illustrates its ability to, not only stabilise, but also improve the performance of some stable non-acyclic, stochastic, multiclass queuing networks, independently of the performance measure under consideration.

Key-words: Queuing Networks, Distributed Scheduling, Stability, Idling Policies, Active Idleness.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of this thesis is to present a different perspective on the stability of multiclass queuing networks. The need of this new perspective comes from a series of remarkable results, published in the last decade, regarding the stability of multiclass, non-acyclic, queuing networks.

It was once thought that if a queuing network had enough resources to process the rate of customers arriving to it, then the number of customers inside the network would stay bounded, meaning that it would be stable. The order by which the customers are processed in each server was considered irrelevant to the stability of the network. The only requirement seemed to be that each server does not stop processing as long as there are customers waiting. The question of the order by which customers are processed was only considered when dealing with the performance of the network.

This conjecture was questioned by the discovery of several examples of queuing networks that although possessing enough resources to cope with the arriving rate of customers, presented an unstable behaviour due to their scheduling policy. Meaning that the order by which customers are served has an important role in the stability of some queuing networks.

With these new results, the main path followed by the queuing networks research community was to develop new methodologies that take into account the scheduling

policy at each server to determine the queuing network's stability.

This thesis sustains that the stability of a queuing network only depends on its topology. The topology of a network includes the number of servers, how they are connected, how do the customers flow through it, and the characteristics of the arrival process and service times. The main point is that an unstable queuing network that possesses the necessary resources can be made stable by slightly changing the working of its scheduling policy. This approach is oriented with a Systems Theory and Control view, since the objective is not to determine if a given queuing network is stable, but if that queuing network can be stabilized, with an appropriate choice of policy.

The main contribution of this thesis concerns the development of a mechanism that performs that slight change in the scheduling policies, stabilizing in that form the queuing network.

To achieve this objective, a new concept, denominated *Active Idleness*, is presented in this thesis. This concept represents the ability of forcing a server to stay idle even in the presence of customers waiting to be served. Although this concept may initially appear contradictory due to the link that is usually made between idleness and waste of resources, this thesis will demonstrate that this concept is a key to the stabilization of multiclass, non-acyclic, queuing networks.

To implement this concept in a real setting, a supervisory mechanism denominated *Time Window Controller* or *TW Controller* is presented. This mechanism represents an instance of the *Active Idleness* concept. This mechanism consist on assigning to each class in the queuing network a fraction of the available resources. If a given class uses more than its share of resources, it is blocked from being processed by a certain amount of time.

With the *TW Controller*, it is possible to stabilized any non-acyclic, multiclass, queuing network that is unstable due to its scheduling policy. This thesis presents

a series of experimental results obtained with different networks to corroborate this statement. Those experiments include the demonstration of the stabilization of the following systems:

- A two server system under *FIFO*, [Dai, 1995].
- A two server system under a buffer priority scheduling policy, [Lu and Kumar, 1991].
- A four server system under *FIFO*, [Seidman, 1994].

It will also be demonstrated that the *Active Idleness* concept in the form of the *TW Controller* is not only able to stabilized, but is also able to improve the performance of a queuing network. Even when the queuing network is originally stable.

The relevant contributions of this thesis are:

- Stability should be considered a property of the queuing network which does not depend on its control policy.
- *Active Idleness* is a key to the stabilization of multiclass, non-acyclic, queuing networks.
- A simple and effective implementation of the *Active Idleness* concept is presented in the form of the *Time Window Controller*
- A formal demonstration and several experimental results demonstrate the ability of the *Time Window Controller* to stabilized any unstable pair *network/policy* that respects some minor technical assumptions.
- The *Time Window Controller* is not only able to stabilized, but is also able to improve the performance of some networks, even when the pair *network/policy* is stable.

1.1 Thesis Outline

This thesis is divided in five chapters and includes two appendices. This chapter presents an introduction to the work presented in the thesis and highlights its main contributions. It also presents a brief review of the contents of each chapter and appendices.

The second chapter, entitled *Literature Review*, presents a review of the relevant literature for this thesis and introduces some important concepts needed in the following chapters. It starts by a brief review of some queuing network concepts, followed by a review of the stability problem for queuing networks. The following sections present some relevant work in the areas of manufacturing systems and data networks, ending with a presentation of the purpose of this thesis taking into account the work described previously.

The third chapter introduces the main concepts and theoretical results of the thesis. The first section presents the concept of *Active Idleness* and the reasons why this concept is a key to the stabilization of multiclass, non-acyclic, queuing networks. The following section rigorously presents an instance of that concept named *Time Window Controller*. The third section presents a series of results regarding the use of the *Time Window Controller*, specially a demonstration of its ability to stabilize multiclass, non-acyclic, queuing networks, under some minor technical assumptions.

The fourth chapter presents a series of experimental results that corroborate the theory presented in Chapter 3. It starts by a brief overview of the software developed to obtain the experimental results. It then follows with the presentation of the used network topology to obtain the experimental results. The experimental results for the *TW Controller* are divided in three parts. The first deals with the stabilization properties, the second with the sensitivity to its parameters, and the last part deals with the performance improvement properties.

The last chapter presents the main conclusions and contributions of this thesis

and presents some paths for future work.

The thesis also includes two appendices presenting a series of experimental results. These results concern the application of the *TW Controller* to two different queuing networks. They serve as a complement to corroborate the analysis presented in Chapter 4.

Chapter 2

Literature Review

This chapter presents a review of the relevant literature for this thesis and introduces some concepts and definitions necessary in the following chapters. Section 1 presents a review of the main concepts and literature regarding queuing networks with emphasis on those relevant to this thesis. Section 2 discusses the queuing network stability problem and the relevant literature associated to it. Section 3 addresses the main results obtained by the application of the queuing network methodology to manufacturing systems, specially to semiconductor manufacturing systems. Section 4 presents some relevant research developed by the data networks community, and the last section uses the previous sections to place this thesis in context.

2.1 Queuing Networks

Queuing networks is one of the most used tools to model *Discrete Event Dynamic System (DEDS)* [Cassandras and Lafortune, 1999]. Figure 2.1 presents an example of a single server queuing network.

Customers arrive to the network entering into the queue where they wait their turn to be processed. The server chooses customers from the queue to process. After being processed, the customers leave the queuing network. This simple example presents the three main entities in a queuing network:

Figure 2.1: Examples of a single server queuing network.

- **Customers**, also known as parts (manufacturing systems) or packages (data networks).
- **Servers**, also referred sometimes as service stations.
- **Queues**, also referred sometimes as buffers.

The number of customers waiting in a queue can alternatively be designated as the queue's inventory.

A queuing network is constituted by two parts: the *queuing network topology* and a *control policy*. The *queuing network topology* contains the layout of the network in the form of the routing each customer must follow. The *topology* also includes the description for the customer arrival and processing time distributions.

The *control policy* performs two functions: controlling the admission of new customers into the queuing network, which is usually referred as the *admission policy* and deciding how each server processes customers, which is known as the *scheduling policy*. Although all queuing networks must have a scheduling policy, it is not mandatory to possess an *admission policy*.

Queuing networks are divided in two major classes: *open queuing networks* and *closed queuing networks*. The difference is that while in *open queuing networks* customers arrive and leave the queuing network at a variable rate, in *closed queuing networks* the number of customers is fixed, or the entry of new customers into the

system is controlled by an admission policy.

The queuing network topology can be classified in two groups. The first group named *acyclic queuing networks*, is characterized by the non existence of cycles in the flow of customers. The second group named *non-acyclic queuing networks* does not have any restrictions on the routing of customers. Figure 2.2 presents examples of these two groups.

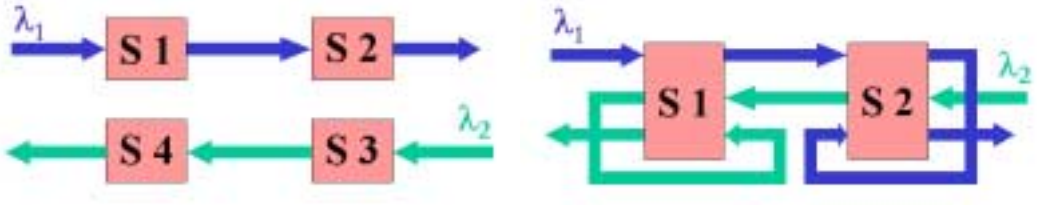

Figure 2.2: Examples of queuing network topologies: acyclic (left) and non-acyclic (right).

The models for the arrival and processing of customers can be classified into *deterministic* and *stochastic* models. A typical example of a stochastic model for the arrival of customers to the queuing network is the *Poisson process* [Ross, 1996]. For the processing time of customers, a typical model is the *exponential distribution* [Ross, 1987].

The *scheduling policy* of a queuing network is classified by two main properties: *locality* and *idleness*. The *locality* of a scheduling policy refers to the amount of information that the scheduling policy needs to perform its scheduling decisions. A scheduling policy can be classified concerning its *locality* as a *distributed (local)* or *non-local* scheduling policy.

- In a *distributed (local)* scheduling policy, each server performs its scheduling decisions using information that is local to the server or its respective customers.

- In a *non-local* scheduling policy, each server uses some or all the information contained in the entire queuing network to perform its scheduling decisions.

The *idleness* of a scheduling policy refers to the ability of a server to stay idle even in the presence of customers waiting to be processed.

- In a *idling* scheduling policy, each server has the possibility of staying idle even when there exist customers to be processed.

- In a *non-idling* scheduling policy, each server will keep processing customers as long as there exist customers waiting to be served.

There exists a large number of scheduling policies for queuing networks. In a deterministic model, it is possible to compute the entire scheduling of customers, since the future in perfectly known [French, 1982, Conway et al., 1967]. In stochastic models the future of the queuing network is unknown. Several scheduling policies have been developed to deal with this problem. [Graves, 1981, Panwalkar and Iskander, 1977] present a review of several distributed scheduling policies for queuing networks.

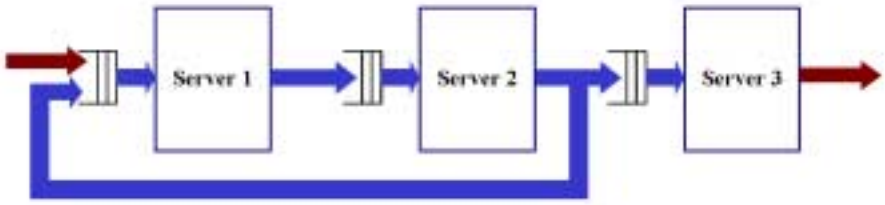

Figure 2.3: Example of a *single class* queuing network.

A queuing network is classified concerning the existence of multiple classes as a *single class* or a *multiclass* queuing network. In a *single class* queuing network, all customers arriving to a given server are indistinguishable. That is, independently of the their past history, they are processed in the same manner. Figure 2.3

presents an example of a *single class* queuing network. [Jackson, 1957] presented a detailed description of the modeling and analysis of some types of *single class* queuing networks, also known as Jackson networks.

In a *multiclass* queuing network, there exists the possibility of a server differentiating customers by their past history and in consequence process those customers in different manners. Note, for example that a Jackson network is not a *multiclass* queuing network because for each server all customers are indistinguishable. Figure 2.4 presents an example of a *multiclass* queuing network.

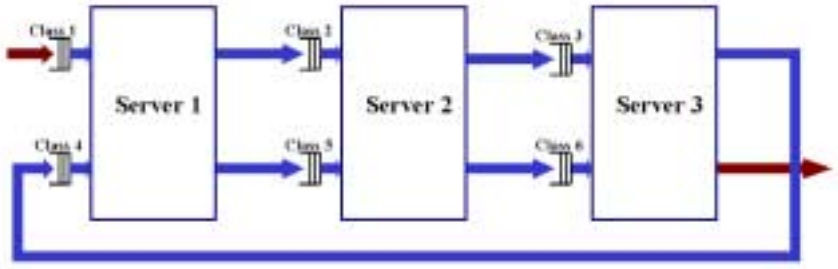

Figure 2.4: Example of a *multiclass* queuing network.

The figure presents a queuing network topology constituted by three servers, where each processes two classes of customers. Note that although in the figure each class has an associated queue, that is not actually required in a physical implementation, since usually there is a single queue for each server. The server recognizes the customer class by a tag attached to it.

In the remainder of this thesis three distributed scheduling policies will be referred in several occasions. For the convenience of some readers, those policies are presented in the following definitions.

Definition 2.1 (First In First Out (FIFO)). In the *First In First Out* scheduling policy, each server chooses for processing the customer that arrived first to the server, independently of its class.

Definition 2.2 (First Buffer First Served (FBFS)). In the *First Buffer First Served* scheduling policy, each server chooses to process the customer belonging to the lower class available at the server. Customers of the same class are served by their order of arrival. Note that for this policy it is necessary that each class number corresponds to the processing stage of the customer, that is, class k corresponds to the k^{th} processing operation that the customer received since entering the network.

Definition 2.3 (Last Buffer First Served (LBFS)). The *Last Buffer First Served* scheduling policy is similar to the *FBFS*, with the exception that priority is given by each server to the customers belonging to the higher class. That is, serve first the class closer to exiting the system.

2.2 Stability of Queuing Networks

Determining the stability of a queuing network is of crucial importance since it determines the ability of the queuing network to process all the customers arriving to the system within reasonable bounds in terms of the flow time.

The classical method to determine the stability of a queuing network is to compute the *invariant probability distribution* for the number of customers in the queuing network [Kleinrock, 1975, Kelly, 1979, Walrand, 1988]. The stability of the queuing network is then obtained by computing the average value of the invariant distribution, which if finite implies the stability of the queuing network.

The problem with this method is that outside the narrow class of queuing networks possessing a *product form* solution, that is, single class queuing networks using the *FIFO* scheduling policy, it is very rare to obtain an explicit solution for the computation of the invariant distribution [Kelly, 1979, Baskett et al., 1975].

Independently of those difficulties, the queuing networks community conjectured

that the stability problem was linked to the question of whether each server in the queuing network had enough resources to process all customers that arrive to the queuing network. This idea was materialized in a stability condition known as the *Traffic Intensity Condition*, which is a necessary stability condition but was also conjectured to be a sufficient stability condition. Before presenting the *Traffic Intensity Condition*, it is first necessary to present the definition of *Traffic Intensity*.

Definition 2.4 (Traffic Intensity). Consider a multiclass queuing network composed of I single server stations. μ_i^k is the mean processing time of class k at server i and λ_i^k the mean arrival rate of class k at server i. N_i is the number of classes processed by server i.

The *Traffic Intensity* at server i is computed by the following expression:

$$\rho_i = \sum_{k=0}^{N_i} \mu_i^k . \lambda_i^k \tag{2.1}$$

Note that class k of server i is associated with a virtual buffer in the service station i that corresponds to a specific processing stage of the customer.

The *Traffic Intensity Condition* requires for all servers $i = 1, ..., I$ that $\rho_i < 1$. This condition is necessary but was also conjectured to be sufficient to guarantee the stability of a queuing network.

This conjecture only looked to one part of the queuing network, that is, it disregarded the scheduling policy implemented at each server. Ignoring the scheduling policy when looking to the stability of the queuing network appeared as an acceptable decision, since as long as the scheduling policy does not imply a reduction of capacity at each server in such a way that the *Traffic Intensity Condition* is not violated, then the network should be stable independently of the scheduling policy in use. The only constraint on the scheduling policy the above argument requires, is for it to be non-idling or work-conserving.

This conjecture was questioned with the discovery of several examples of queuing networks that, although respecting the *Traffic Intensity Condition*, presented an unstable behaviour. [Lu and Kumar, 1991] presented an example of a queuing network topology with two servers and four classes that in conjunction with a non-idling buffer priority scheduling policy resulted in an unstable queuing network. [Rybko and Stolyar, 1992] also presented an example of an unstable queuing network under a buffer priority scheduling policy. [Seidman, 1994] presented a queuing network topology with four servers and twelve classes that is unstable under the First In First Out scheduling policy. This result showed that even this simple scheduling policy presented stability problems. Bramson in [Bramson, 1994a, Bramson, 1994b] presented a rigorous demonstration of the instability of the *FIFO* scheduling policy for some queuing network topologies.

The problem was that although the traffic intensity condition appeared to work for several queuing networks (e.g. Jackson Networks), the above authors were able to present examples of queuing networks where the scheduling policy gained a more relevant role due to the inclusion of non-acyclic flows. These non-acyclic flows in conjunction with the scheduling policy created a starvation phenomena between the service stations, invalidating the *Traffic Intensity Condition* conjecture.

After these results, the queuing networks research community started to develop new methodologies to deal with the stability problem. The goal was to develop a mathematical framework that when applied to a given queuing network, would be able to provide sufficient conditions for its stability. [Chen and Zhang, 2000] present in their introduction a review of some of those methodologies.

The foremost methodology has been to transform the queuing network in an equivalent fluid model and to use the latter one for studying the stability of the queuing network [Dai, 1995, Dai and Meyn, 1995, Chen, 1995].

This approach is based on a result first established in [Rybko and Stolyar, 1992]

and then generalized in [Dai, 1995] that a queuing network is stable if its corresponding fluid network is stable. [Bramson, 1999] presents an example that demonstrates that the reverse is not true, that is, an unstable fluid model does not imply that the original queuing network is unstable.

An effective method for studying the stability of fluid network is to identify a Lyapunov function of a fluid network. [Chen, 1995] and [Kumar and Meyn, 1995] used quadratic Lyapunov functions.

[Bertsimas et al., 1996] proposed a linear program that could also establish a necessary and sufficient condition for the global stability of a two station queuing network. [Dai and Weiss, 1996] used linear piecewise Lyapunov functions to establish the stability of several queing networks. Another important result was provided in [Dai et al., 1999], where an example of a three station multiclass queuing network is presented, for which the the stability region is not monotone in terms of the processing time of each customer class.

The difficulties with these methodologies, is their dependence on sophisticated analytical tools that require several conditions to be met by the queuing network. The results obtained are sometimes partial in the sense of only providing the stability condition for a subset of the possible parameter space.

There are several scheduling policies which are known to be stable for any queuing network topology. One of those is the *Head-of-the-Line Proportional Processor Sharing (HLPPS)* scheduling policy. In this scheduling policy, the total capacity at each service station is shared between the classes waiting to be served, proportionally to their processing times. [Bramson, 1998] demonstrated that this scheduling policy is stable for any queuing network topology as long as the *Traffic Intensity Condition* is true. This demonstration is only applicable to continuous systems, where all classes are served simultaneously at a rate dependent of the capacity allocated to it. The transposition of this scheduling policy to discrete time systems

was performed by the data networks community in the form of the *Packet-by-Packet General Processor Sharing (PGPS)* [Parekh and Gallager, 1993] and the *Weighted Fair Queuing (WFQ)* [Demers et al., 1990] scheduling policies. The ability of this scheduling policy to be stable for any queuing network topology comes from the fact that it separates the queuing network in to a set of tandem queues for which the *traffic intensity condition* is still a valid conjecture to be a sufficient stability condition (in the case of a Markovian system the conjecture has already been demonstrated to be true).

Another point that is important to this discussion on the stability of queuing networks regards the properties of idling scheduling policies. Currently, the methods developed to deal with the stability of queuing networks require the scheduling policy of the network to be non-idle. This requirement is due to the extra complexity that idling policies bring in the form of an extra degree of freedom, since each server not only has to choose the next customer to serve, but it also has the possibility of staying idle. This degree of freedom destroys the analytical properties that non-idling policies have and that are essential for those methods. The requirement of only using non-idling scheduling policies might initially appear as a small drawback, since idling policies appear at a first glance as a waste of capacity.

The notion of idling policies being looked as a waste of capacity is a very common conception. The truth is that idling policies present very interesting properties regarding the stability of queuing networks. [Perkins and Kumar, 1989] were able to guarantee the stability of their scheduling policy for non-acyclic queuing network by the inclusion of an idling behaviour to the scheduling policy. Another way where the idling behaviour has been used to solve stability problems is through the use of regulators. The notion of regulator presented in [Anantharam and Konstantopoulos, 1994] was used in [Humes Jr., 1994] and [Winograd and Kumar, 1996] to present some solution to the stability question. Regulators have also been studied regarding

their use for solving the problem of the quality of service in a data network [Turner, 1986].

2.3 Manufacturing Systems

The queuing networks methodology has for long been extensively applied to the modeling and analysis of manufacturing systems. The more complex manufacturing system to which the queuing network methodology has been applied is Semiconductor Manufacturing systems. These systems are especially complex due to the non-acyclic topology they possess and for that reason are especially relevant to this thesis. Figure 2.5 presents an example of a model for a semiconductor manufacturing line.

[Kumar, 1993] presents a review of results regarding *re-entrant lines*. A *re-entrant line* is a non-acyclic queuing network with only one entry and exit point, which is a very common type of topology in semiconductor manufacturing facilities. [Wein, 1988] studied the impact of scheduling in semiconductor wafer fabrication facilities, comparing a variety of admission and scheduling rules. He concluded that a larger performance improvement could be obtained by an appropriate choice of the admission policy than from the scheduling policy. This meant that the admission policy should be looked as the important component of the system control.

[Lu et al., 1994] presented a new class of scheduling policies, called *Fluctuation Smoothing*, and in the same line of the previous work by Wein, presented a detailed statistical analysis of several admission and scheduling policies for a semiconductor wafer fabrication facility. Contrary to the conclusions of Wein, this article presented a new family of scheduling policies named *Fluctuation Smoothing (FS)*, that is able to obtain a significant performance improvement, even using the best admission policy of Wein. This results clearly showed that contrary to the results of Wein, the scheduling policy is as important as the admission policy.

The previous article also demonstrated that the *Fluctuation Smoothing (FS)* policies are able to obtain a better performance than all other policies presented in the article. The problem is that *FS* policies are non-local and of complex implementation, as discussed in [Nakata et al., 1999].

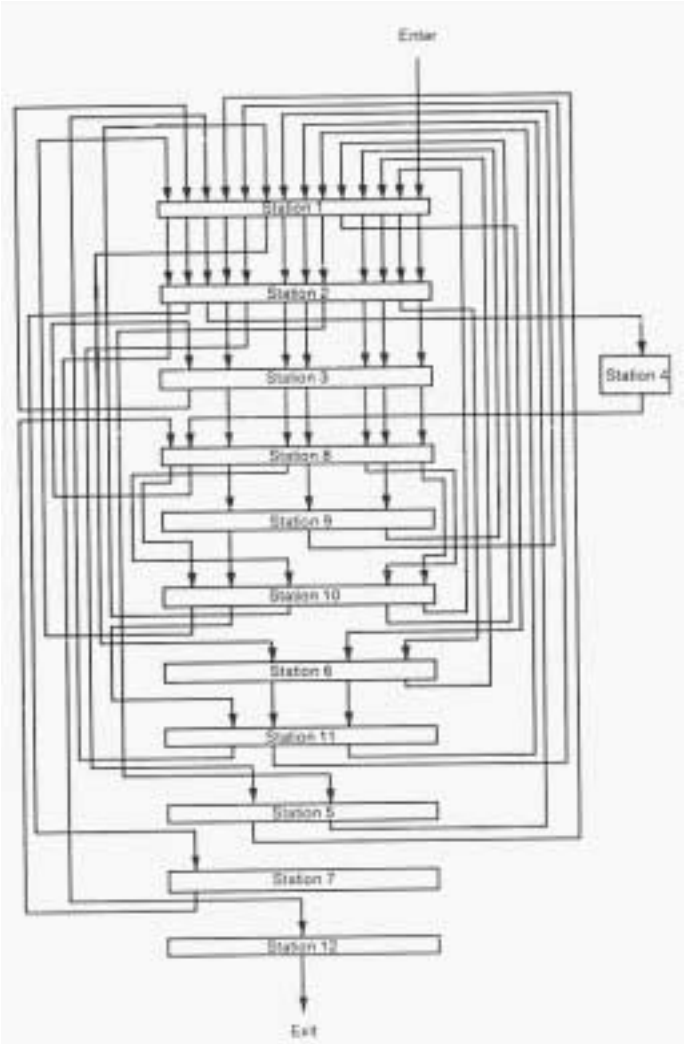

Figure 2.5: Model of a semiconductor manufacturing line (from [Lu et al., 1994]).

2.4 Data Networks

The field of data networks [Tanenbaum, 1996, Keshav, 1997, Bertsekas and Gallager, 1987], has evolved at a stunning speed over the last decades. The "information age" requires the development of increasingly complex data network systems [Comeford, 2000]. To deal with the increasing complexity of those systems, it is necessary to employ complex modelling techniques for their study. Queuing networks is one of the techniques that has been applied to the modelling of data networks [Bertsekas and Gallager, 1987, Cassandras and Lafortune, 1999].

The question of queuing network stability is not a primary concern in this field, since the existence of non-acyclic queuing networks like those presented in Figure 2.5 is very rare or even non-existent.

The research area on data networks that is of interest to this thesis is the area that deals with the *Quality of Service (QoS)* guarantees for a data network. This area deals with the problem of providing performance guarantees for the customers of a data network. One example of this problem is the current effort to provide multimedia applications (ex. audio and video conference, multimedia information retrieval, etc.) over the Internet. This requires that the network provides guarantees with respect to bandwidth, packet delay, delay jitter, and loss. To enable the network to provide this type of guarantees, the customers must specify their traffic characteristic so that the network is able to provide a quality of service guarantee by reserving and scheduling network resources in accordance with those specifications.

Although this problem does not appear directly linked to the question of the stability of a queuing network, it is obvious that requiring that a queuing network provides certain *QoS* guarantees is a much stronger requirement than requiring that the queuing network is stable.

The objective is to look at the methodologies used to solve this problem for some insight into possible solutions to the the question of stability. This idea is not new

since in [Bramson, 1998] it was stated that the manufacturing systems community should look into the work being developed by the data networks community.

Given the traffic specification of the customers, there are several scheduling policies that are able to provide *QoS* guarantees. [Zhang, 1995] provides a review of some of those scheduling policies. They are divided into work-conserving (non-idling) and non-work-conserving (idling) scheduling policies. Table 2.1 presents some of the work-conserving scheduling policies and table 2.2 presents some of the non-work-conserving scheduling policies.

Table 2.1: Work conserving scheduling policies for *QoS* guarantees.

Scheduling Policy	References
Virtual Clock	[Zhang, 1991]
PGPS (packet-by-packet GPS)	[Parekh and Gallager, 1993]
	[Parekh and Gallager, 1994]
WFQ (weighted fair queuing)	[Demers et al., 1990]
SCFQ (self-clocked fair queuing)	[Golestani, 1994]
WF^2Q (worst-case fair WFQ)	[Bennet and Zhang, 1996]
Delay-EDD (delay-earliest-due-date)	[Ferrari and Verma, 1990]
Frame-based fair queuing	[Stiliadis and Varma, 1996]

Table 2.2: Non-work conserving scheduling policies for *QoS* guarantees.

Scheduling Policy	References
JITTER EDD (jitter earliest-due-date)	[Verma et al., 1991]
RCSP (rate-controlled static priority)	[Zhang, 1995]
Hierarchical roud robin	[Zhang and Ferrari, 1993]
Stop-and-go queuing	[Golestani, 1991]

Some authors were able to develop a joint approach to this problem by investigating specific combinations of traffic conditions and scheduling policies [Cruz, 1991a, Cruz, 1991b, Parekh and Gallager, 1993, Parekh and Gallager, 1994].

Of special interest are scheduling policies like the *WFQ* or *PGPS*. These policies are based on the *General Processor Sharing (GPS)* scheduling policy, which con-

sist on splitting the resources at the server between the several customers that are sending traffic to it. This ensures that none of the customers is allowed to use the resources at the expense of denying a share of resources to the traffic sent by the other customers.

The scheduling policy is not able to guarantee the *QoS* of a queuing network if some of the customers are unable to respect the agreed traffic specifications. To solve this problem the concept of regulator was developed. One possible implementation is the *Leaky Bucket Regulator* presented in [Anantharam and Konstantopoulos, 1994] and graphically described in Figure 2.6.

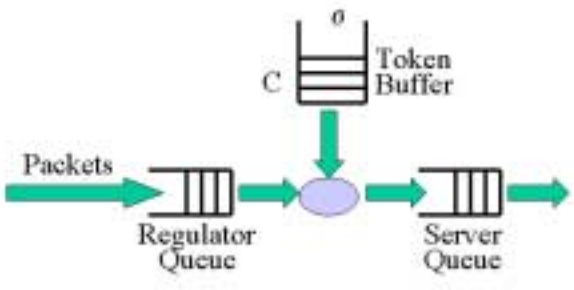

Figure 2.6: Example of the *Leaky Bucket Regulator*.

The *Leaky Bucket Regulator* works as follows. There exists a *Token Buffer* with a finite size C that receives *Tokens* at a constant rate σ. Packets arriving to the regulator are sent to the server queue, consuming in the process a certain number of *Tokens* proportional to the packet size. If there are not enough *Tokens* in the *Token Buffer*, then the packet will stay in the regulator buffer until enough *Tokens* arrive to allow the packet to be able to pass to the server queue.

Adding a regulator to the traffic sources in a queuing network, guarantees that they will comply with the agreed traffic specification since any deviation to that specification will be filtered by the regulators in the form of an increase of the inventory at the regulator queue. Figure 2.7 presents an example of a server with

two regulators. When analyzing the behaviour of the system that results from the reunion of the regulators plus the server, it is possible to conclude that it can in some situations present an idling behaviour, since there is the possibility of existing inventory at the entrance of the regulators but none at the server entrance due to token exhaustion. This implies that the server will stay idle even with packets trapped behind the regulators waiting to be processed.

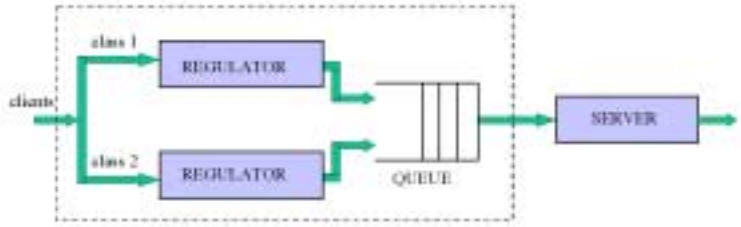

Figure 2.7: Graphic description of the use of regulators for a two class server.

A related area of interest is the question of *Service Differentiation* in a data network. This problem regards the control of a queuing network were there is a hierarchy of classes to which the customers belong. A possible example of this problem is an Internet Service Provider with different services. It is in the interest of the service provider to always serve at a faster rate the services that require short response time (ex. IP telephony) than those that allow longer response times (ex. FTP session).

This problem can be addressed by several methodologies. But in [Dovrolis and Ramanathan, 1999] it was shown that by using an appropriate scheduling algorithm which shares the resources proportionally to the class level provides better results than other methods based on class priority or on price schemes.

2.5 A Different Perspective

Taking into account the discussion presented in the previous sections, there exists a dichotomy regarding queuing networks stability.

- It is always possible to define a scheduling policy that renders any queuing network unstable – for instance, adding too much idle time.
- If a network topology respects the *Traffic Intensity Condition*, then there are several scheduling policies that render the queuing network stable – for instance, *Least Slack* rule, [Lu et al., 1994].

This dichotomy leads to conclusions like either the same queuing network is stable or unstable depending on its control policy. Any of the conclusions depends on the scheduling policy being used. This thesis looks at stability in a different perspective.

When analyzing a queuing network, the first and foremost question, is if the network is stable. Due to the above dichotomy, a negative answer does not imply that a change in the control policy might make the network stable. It seems then more appropriate to regard stability as a question of whether a given queuing network might be stabilized or not. With this change of perspective, the stability question is only linked to the network topology and not to a specific scheduling policy, becoming an intrinsic property of the queuing network. This way, a negative answer implies that there exists no control policy that renders the queuing network stable.

Another question this thesis addresses, which results from the discussion in the previous sections, regards the separation between admission policies and scheduling policies. It will be shown in the next chapters that the proposed solution to the stability question also brings the unification of the admission policy and scheduling policy into a single framework.

Chapter 3

The Time Window Controller

This chapter presents the concept of *Active Idleness* and an implementation of that concept named *Time Window Controller*. The first section introduces the reasons behind the concept of *Active Idleness* and the implementation of the *Time Window Controller*. The second section presents a rigorous description of the *Time Window Controller*. The last section presents the properties of the *Time Window Controller*, specially a demonstration of its ability to stabilize unstable queuing networks.

3.1 Introduction

From the discussion in Chapter 2 there are two important conclusions. The first is that it appears that the *Traffic Intensity Condition* is not a sufficient stability condition for non-acyclic queuing networks. The second conclusion is that idling policies present some interesting features in what concerns their potential to stabilize queuing networks, which requires further study.

The difficulty concerning stability in non-acyclic, multiclass, queuing networks resides on the strong coupling between different classes served by any given server and the impact this coupling may have on subsequent servers and classes. The majority of the authors tries to look at the stability problem as that of the pair *topology/scheduling policy*. This thesis claims that stability is an intrinsic property

of the topology. The perspective of this thesis is that a queuing network is stable if there is one scheduling policy that renders the system stable. One can think of many policies that make one system unstable. One instance is a policy that shuts down the server whenever there is work above some bound. Clearly, if the probability of reaching that bound is non zero, then the queue will grow indefinitely with probability one after some time. On the other hand, there are scheduling policies, like the *Least Slack Rule*, that guarantee that a queuing network is stable as long as it respects the *Traffic Intensity Condition*. With this perspective, the *Traffic Intensity Condition* can again be considered a sufficient condition.

The objective of this thesis is not to simple say that if a queuing network under a given scheduling policy is unstable, although respecting the *Traffic Intensity Condition*, the solution is to simply change the scheduling policy to one that guarantees the stability of the queuing network, like the *Least Slack Rule*. The objective is to eliminate the phenomena that brings unstable behaviour to multiclass, non-acyclic, queuing networks.

The discussion will concentrate on non-idling distributed scheduling policies, which correspond to all the examples presented in the previous chapter. The reason why networks can in some instances present an unstable behaviour (considering stability for the pair *topology/scheduling policy*) is that each server does not perceive the influence of its actions on the performance of the entire network.

In a non-idling scheduling policy, each server will keep processing customers as long as there are customers waiting to be served. This behaviour seems appropriate, since staying idle when there exist customers waiting to be served seems a waste of resources, contributing to a degradation of the performance. But this is a line of thought that only looks to the performance of each server individually. It does not take into account how these decisions can affect the performance of the network as a whole, specially in non-acyclic queuing networks.

Another question regards the fairness of treatment between classes in multiclass queuing networks. Scheduling policies like *FIFO* do not take into account how the resources of each server are being divided between classes, and given they are local scheduling policies, they are not able to foresee how their choice of class to process affects the performance of the entire network.

It is the combination of these two effects, non-idleness and myopic behaviour regarding classes, that may lead a multiclass, non-acyclic, queuing network to instability. Examples of this have been published and Chapter 4 will present experimental evidence of this assertion.

The solution proposed in this thesis is to provide the scheduling policy with the minimum amount of non-local and static information that guarantees the stability of the queuing network. The reason of keeping the information to a minimum and static is to keep the scheduling policy distributed and simple to compute. This way each server continues to take its decisions locally, with a small set of parameters that are static in time and contain some kind of information about the entire network. The advantage of this framework is that there is no need to construct a central controller that gathers information from the entire network.

The question resides on how should this non-local information be introduced into the workings of the scheduling policy. This thesis presents for that end the concept of *Active Idleness*. The *Active Idleness* concept represents the ability to force a server to stop processing a given class and in some occasions even stay idle in the presence of customers to be processed. Note that what is meant with this concept is that each server in multiclass queuing networks, with a non-idling scheduling policy, will in certain occasions be prohibited of processing certain classes of customers, or even processing any customer, thus transforming the original policy into an idling policy. The implementation of this concept resides on using the non-local information to insert in each server the appropriate amount of *active idle* time.

Why should this approach succeed in stabilizing a queuing network that is unstable, due to its non-idling and distributed scheduling policy? The reason resides in the fact that an appropriate use of *Active Idleness* can in fact insert the change in the myopic behaviour of the scheduling policy at each server, so that it takes into account the needs of the entire queuing network.

The main question resides in the development of an application of this concept that is able to fulfill the expectations described above. This thesis presents a supervisory mechanism named *Time Window Controller*, or *TW Controller*, that is added to the scheduling policy at each server. The main objective of the *TW Controller* is to decouple the queuing network through *Active Idleness* into a series of tandem queues by splitting the resources at each server among its respective classes. It is this splitting that guarantees no server in the network to become starved of a certain class due to the behaviour of another server.

For the *TW Controller* to be able to decouple the queuing network through splitting the resources at each server among its classes, it must first measure the amount of resources used by each class. For that, the *TW Controller* will use the processing history of each class to compute the amount of server resources used by that class. The objective is not to compute the long term server resource usage by each class, but to use a short/medium term measure, so that it corresponds to the current status of the network. With this measure, the *TW Controller* can restrict the amount of resources used by each class to a maximum value. With this scheme it is easy to decouple the queuing network. For that, it is only necessary to divide the resources available at the server between the classes it serves. This split requires that the sum of the maximum amount of resources that are used by each class is less or equal to the total available at the server. Figure 3.1 presents an example of this idea. The *TW Controller* is set to ensure that class 1 only gets 30% of the available resources, while class 2 gets 20%, and class 3 gets 50%. As long as the

short term occupation of the server by each class obeys those fractions, the single server behaves as if there would be three servers, each serving a single class.

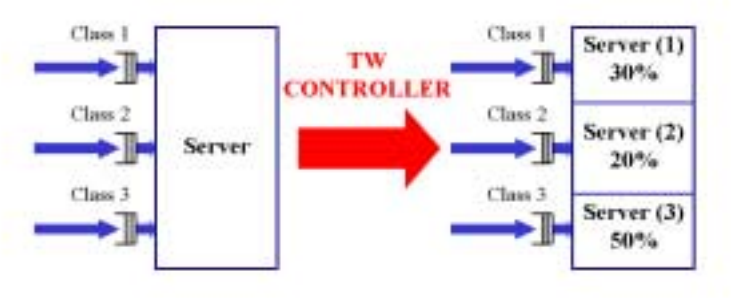

Figure 3.1: Example of the resource splitting approach of the *TW Controller*.

Note that the *TW Controller* decouples the queuing network through the scheduling policy, that is, if a class exceeds its maximum resource level, the *TW Controller* will not allow the server to process customers of that class. At first, this seems similar to the *Generalized Processor Sharing* approach. The next section will show it is not so.

3.2 The Time Window Controller

Consider an open, multiclass, queuing network composed of I single server stations. Customers inside the queuing network are members of classes (or buffers) which correspond to virtual buffers linked to a specific operation in a given server. The network is populated by K classes, where customers of class k arrive to the network via an exogenous arbitrary arrival process with independent and identically distributed interarrival times $\{\epsilon_k(n), n \leq 1\}$. It is allowed for some k that $\epsilon_k(n) = \infty$ for all n, in which case the external arrival process to class k is null. A customer of class k, after being served at a unique server j, written $j = s(k)$, or conversely,

$k \in c(j)$, becomes a customer of class $R(k)$, where R is a bijective function that represents the routing map for the queuing network.

Given an arbitrary probability distribution for the processing time of each customer, denote by μ_k the mean processing time for customers of class k. Customers of different classes are not allowed to merge into a single class or to split into different classes. For each class k define $F(k)$ as:

$$F(k) = \begin{cases} k & \text{if class k has an non-null exogenous arrival rate.} \\ j & \text{if } k \text{ has a null exogenous arrival rate, where } j \text{ is} \\ & \text{the class with a non-null exogenous arrival rate that} \\ & \text{feeds customers to class } k. \end{cases} \tag{3.1}$$

Note that, since there are no splitting or merging of classes, $F(k)$ is an injective function. For each class k let $\lambda_k = \frac{1}{E[\epsilon_{f(k)}(1)]}$. One interprets λ_k as the effective mean arrival rate of class k.

The class of queuing networks described above requires the use of a deterministic routing for the customers and does not allow assembly or disassembly operations in the form of merging or splitting customers of different classes. In the remainder of this thesis, when referring to a multiclass queuing network, it corresponds to a queuing network which respects the previous conditions.

The *TW Controller* will be presented in a constructive way, where first a series of definitions corresponding to its building blocks are introduced. Then, this section culminates with the definition of the *Time Window Controller*. The first of those is the definition of the *Time Window* associated to a class.

Definition 3.1 (Time Window). Consider a class k of a multiclass, non-acyclic, queuing network. The *Time Window* associated to that class is defined as the finite time interval that starts at the current system time t_c and extends T_k time units into the past.

The *Time Window* of a class represents the amount of the class history that will be needed by the *TW Controller*. The use of a finite size window is not only due to memory and computational requirements, but is also essential to achieve the short/medium term objectives set to the *TW Controller*, as will be shown in the next section.

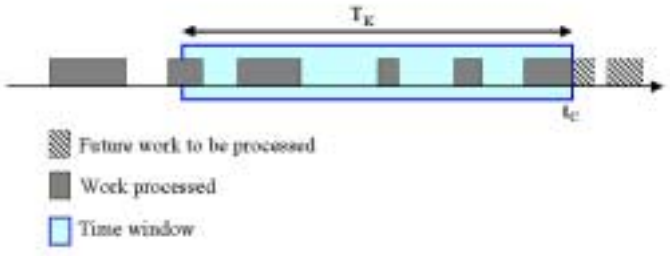

Figure 3.2: Example of the *Time Window* for a class k.

Figure 3.2 represents a time diagram of the processing times for several customers of a given class. Each filled rectangle represents a customer. The *Time Window* for this class is represented in the figure by the large rectangle that starts at the current system time t_c and includes all the customers processed in the previous T_k time units. As time progresses, the *Time Window* slides to keep up with the system's evolution. The next definition is that of the *Processing History* associated to a class.

Definition 3.2 (Processing History). For each customer i of class k, define $t_{k,i}^{start}$ and $t_{k,i}^{end}$ as the start and finish time instants for the processing of that customer. The *Processing History* of class k is defined as a function $H_k(t)$ given by:

$$\underset{0\leq t\leq t_c}{H_k(t)} = \begin{cases} 1 & \text{if } t_{(k,i)}^{start} \leq t \leq t_{(k,i)}^{end} \ \forall i \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The *Processing History* associated to a class represents a function that describes the amount of time used by the server to process customers of that class. The

objective is to obtain a chronological description of the processing time used by each class from the corresponding server.

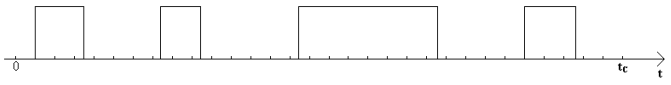

Figure 3.3: Example of $H_k(t)$.

Figure 3.3 presents an example of a possible *Processing History* for class k. The next definition presents the concept of the *Time Fraction* of a class k at a given time t:

Definition 3.3 (Time Fraction). The *Time Fraction* of class k with a *Time Window* of size T_k at time t is defined as $f_k(t)$ and is computed by the following expression:

$$f_k(t) = \beta_k . \frac{1}{T_k} \int_{t-T_k}^{t} e^{\alpha_k . (\tau - t)} . H_k(\tau) d\tau \tag{3.3}$$

where $\alpha_k \in [0, \infty[$, is the Smoothing Parameter and β_k is a normalization parameter, given by:

$$\beta_k = \frac{\alpha_k}{1 - e^{-\alpha_k . T_k}} \tag{3.4}$$

The *Time Fraction* of a class represents the fraction of the total time contained in its *Time Window* during which the server was processing customers of that class. It clearly represents a measure of the amount of server resources assigned to customers of that class. If $\alpha_k = 0$, it measures the exact time fraction allocated to class k over the *Time Window* span.

The need to use an exponential function to smooth the class *Processing History* is to eliminate sudden changes on the *Time Fraction* value of a class. The objective

is that when computing the *Time Fraction*, the more recent processing history has a larger weight to the computation than the processing history near the end of the *Time Window*. Figure 3.4 presents an example of the type of behaviour that could arise without a smoothing function.

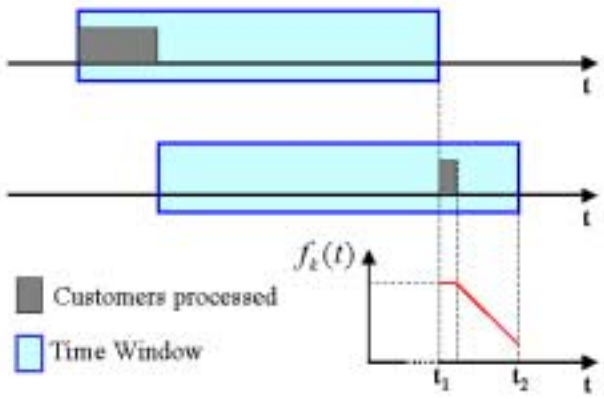

Figure 3.4: Example of the behaviour of the class *Time Fraction* without a smoothing function.

The figure represents the evolution of class k *Time Fraction* between time instants t_1 and t_2. At time instant t_1 the server starts processing a customer of class k. At time instant t_2, one previous processed customer leaves the scope of the *Time Window*. The problem is the influence of that previous processed customer on the class *Time Fraction*. Since that customer was processed in a distant past relative to the *Time Window*, it should not present such a large influence in comparison to customers that were processed more recently. This behaviour would be reduced with the use of an appropriate exponential smoothing function.

Since the *Time Window* width is finite, it was necessary to include the normalization factor β_k. This guarantees that, if during the entire *Time Window* the server is always processing customers of a given class, the computed value for the *Time*

Fraction of that class will be 1.

Figure 3.5 presents an example of the computation of the *Time Fraction* for a class. The graphic contains the exponential smoothing function and the starting and ending instants of several customers of this class that were served during the present *Time Window*. The *Time Fraction* is the sum of the total area of the graphic. To compute the *Time Fraction* of a given class at time t it is only necessary to have the *Processing History* contained in the class *Time Window*.

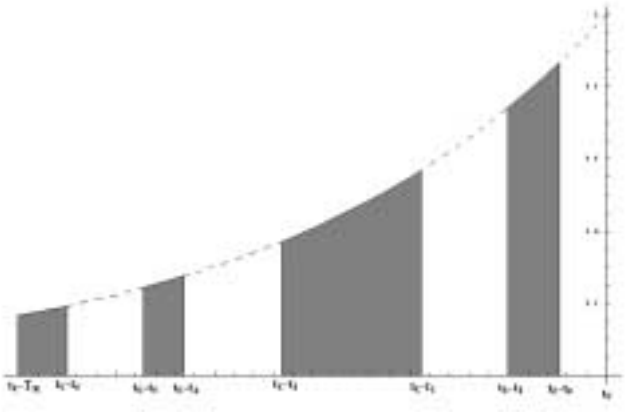

Figure 3.5: Computation of the *Time Fraction*.

The last concept necessary for the presentation of the *TW Controller* is the concept of *Blocked* class, which has the following definition:

Definition 3.4 (Blocked class). A class k is said to be *Blocked* at time t with parameter f_k^{max} if $f_k(t) > f_k^{max}$, where f_k^{max} is the *Maximum Time Fraction* allowed for class k.

A *Blocked* class is simply a class that has exceeded the *Maximum Time Fraction* f_k^{max} that was awarded to it. Since the *Time Fraction* $f_k(t)$ is a measure of the server resources used by class k in its *Time Window*, the *Maximum Time Fraction* f_k^{max} represents the maximum level of resources that class k can use in its *Time Window* without becoming *Blocked*.

Finally, by using the previous definitions it is now possible to present a definition of the *Time Window Controller*.

Definition 3.5 (Time Window Controller). Let ω be a multiclass, non-acyclic, queuing network, where each service station is controlled by the non-idling scheduling policy Λ. The *Time Window Controller* for this queuing network consists on assigning to each class k a *Maximum Time Fraction* f_k^{max} and a *Time Window* with parameter T_k for which it is possible to compute the *Processing History* $H_k(t)$ with a *Smoothing Parameter* α_k. Each service station performs its scheduling decisions using policy Λ with the exception that, when performing a scheduling decision at time t, all classes that are *Blocked* should be considered empty of customers.

The definition states that the *TW Controller* is described by a set of parameters $(\alpha_k, T_k, f_k^{max})$ with $k = 1...K$. The functioning of the *TW Controller* is very simple. Each time a server has to make a scheduling decision, the *TW Controller* calculates the *Time Fraction* of all classes in that server. If any class has a *Time Fraction* higher that its *Maximum Time Fraction*, then the *TW Controller* blocks that class from the set of classes from which the server can remove customers to process. This procedure restricts the server's scheduling options, since it can only use a subset of the entire set of available customers.

The only interference of the *TW controller* in the original scheduling policy resumes to the possibility of not allowing the scheduling policy to use a given class because it is *Blocked*. Thus, there is the possibility that at a certain time the scheduling policy is not able to choose a customer to be processed because all customers are in classes that are *blocked*. In this case the server becomes idle, not because the server is empty of customers, but because the *TW Controller* forbids the scheduling policy of using the available customers. For this reason this type of idleness is termed as *Active Idleness* since it is imposed by the *TW Controller* on the scheduling policy

and not a consequence of actual absence of customers. Naturally, idleness incurred for actual lack of customers, would be considered as *Passive Idleness*. Figure 3.6 presents a graphic example of this concept.

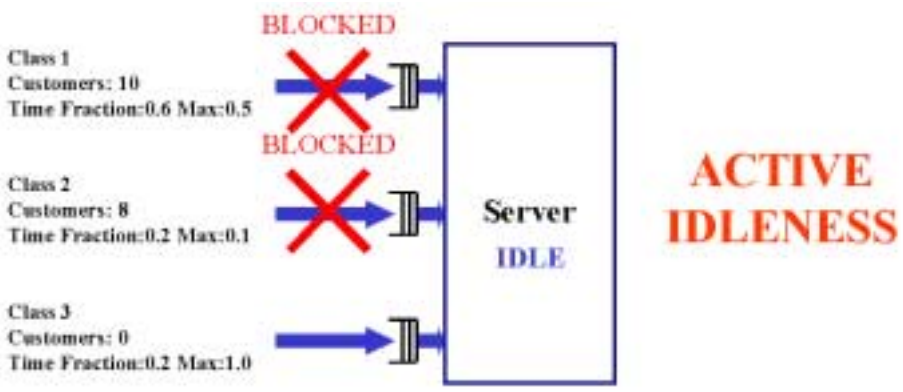

Figure 3.6: Example of *Active Idleness*.

The figure illustrates a server that processes customers of three different classes. There exist 10 customers of class 1, 8 customers of class 2 and no customers of class 3. In the situation presented, class 1 and 2 have their *Time Fractions* larger than their respective *Maximum Time Fractions*, which means that those classes are *Blocked*. Class 3 has a *Time Fraction* lower than its *Maximum Time Fraction* but, since there are no customers of that class, the server is unable to process any customers, although there are customers of class 1 and 2. This situation clearly represents an instance where the server becomes idle due to an active imposition by the *TW Controller* which corresponds to the concept of *Active Idleness*.

During the queuing network evolution, when a class becomes *blocked* the server will no longer be able to process any of its customers, which implies that its corresponding *Time Fraction* will decrease with time, guaranteeing that in some point in the future it will cease to be *Blocked*. Note that adding the *TW Controller* to the queuing network does not imply that the scheduling policy is no longer distributed. Each server, in essence, has a *TW Controller* with the $(\alpha_k, T_k, f_k^{max})$ parameters

corresponding to the classes it processes.

3.3 Properties of the Time Window Controller

Given the description of the *TW Controller* presented in the previous section, the first property that is possible to deduce is presented in the following theorem.

Theorem 3.1. There is a choice of $(\alpha_k, T_k, f_k^{max})$ such that the performance of a multiclass, non-acyclic, stochastic queuing network with the *TW Controller* can be at least as good as that of the original queuing network, independently of the performance measure under consideration.

Proof: The proof is straightforward. If the *Maximum Time Fraction* of each class, f_k^{max}, is assigned to a unitary value, then for any value of T_k and α_k none of the classes will ever be *Blocked*, which implies that the original scheduling policy will never be interrupted by the *TW Controller*. ■

Remark 3.1. Given that the choice of $(\alpha_k, T_k, f_k^{max})$ in Theorem 3.1 is a feasible instance of the *TW controller*, it constitutes an upper bound on the queuing network's performance.

Theorem 3.1 guarantees that there is an instance of the *TW Controller* that in essence makes it non existent by not allowing it to interfere with the original scheduling policy. Remark 3.1 takes that property of the *TW Controller* into account to stress that there may exist other instances where the performance of the queuing network is improved in regard to the one obtained with the original policy. Note that this is only a possibility. If the original scheduling policy is the optimal for the network, then there is no instance of the *TW Controller* that improves its performance.

The second property of the *TW Controller* represents the main contribution of this thesis. It concerns the stabilization properties of the *TW Controller* and will demonstrate its ability to stabilize a large class of queuing networks. It requires only that the queuing network respects the *Traffic Intensity Condition* and some mild assumptions on the service and arrival distributions. First it is necessary to present the following definition:

Definition 3.6. Let Ω_T be the set of all multiclass queuing networks with a single class per service station, a single entry point for new customers, and only one exit point. The set Ω_{TS} is defined as the subset of Ω_T for which the *Traffic Intensity Condition* is a sufficient stability condition.

The objective of this definition is to present the set of tandem queuing networks for which the *Traffic Intensity Condition* is a sufficient stability condition. This definition is needed because there are queuing networks that belong to the set Ω_T but are not members of the set Ω_{TS}. For example, by the *Pollaczek-Khinchin* formula [Cassandras and Lafortune, 1999], a GI/GI/1 queue may have an unbounded queue length when the arrival and processing times are modeled by heavy tail distributions [Adler, 1998]. Note that Ω_{TS} contains a large number of queuing networks including, for example, all Markovian tandem queuing networks.

Finally, the following theorem presents the stabilization property of the *TW Controller*.

Theorem 3.2. Consider a queuing network ω with I service stations and K classes. Each service station is controlled by the non-idling scheduling policy Λ and the queuing network respects the *Traffic Intensity Condition*. Each customer of class k has a maximum processing time ψ_k. There is a set of parameters $(\overline{\alpha}_k, \overline{T}_k, \overline{f}_k^{max})$ such that the *TW Controller* with those parameters is able to:

a) Decouple the queuing network ω into M networks, ω_m, with $m = 1, ...M$, for some $M \leq K$. Each $\omega_m \in \Omega_T$.

Let Ω_{dec} be the set of all ω_m.

b) If Ω_{dec} is contained in Ω_{TS}, then the *TW Controller* with parameters $(\overline{\alpha}_k, \overline{T}_k, \overline{f}_k^{max})$ is a stable policy for the original queuing network.

Proof: The proof will be presented in two parts. First, a method to compute an instance $(\overline{\alpha}_k, \overline{T}_k, \overline{f}_k^{max})$ of the *TW Controller* is presented. Next, it will be demonstrated that with these parameters, the *TW Controller* is able to accomplish the two properties described in the theorem.

Consider a multiclass queuing network ω. Since the queuing network respects the *Traffic Intensity Condition*, the following expression is true for each server i of the queuing network ω.

$$\sum_{k=1}^{K} \lambda_k.\mu_k.\sigma(i,k) + \varepsilon_i = 1 \quad \text{for } i = 1, ..., I \text{ and } \varepsilon_i > 0, \tag{3.5}$$

$$\text{with } \sigma(i,k) = \begin{cases} 1 & \text{if } k \in c(i), \\ 0 & \text{otherwise.} \end{cases}$$

Equation 3.5 presents a different version of the *Traffic Intensity Condition* as originally presented in Chapter 2. This difference is due to the manner in which the classes in the queuing network are named. In definition 2.1 each class was associated to the respective server while in equation 3.5 each class has a unique value. The value ε_i represents the slack available at server i. This slack guarantees that the resources at the server are above those required by the customer arrival rate.

Choose for $\overline{\alpha}_k$ and $\overline{f}_k^{max}$ the following values:

$$\begin{aligned} \overline{\alpha}_k &= 0 \text{ for } k = 1, ..., K \\ \overline{f}_k^{max} &= \mu_k.\lambda_k + \frac{\varepsilon_i}{2 \times N_i}, \end{aligned}$$

where N_i is the number of classes at server i. Note that the slack, ε_i at each server is divided in two, where half is included in each class' *Maximum Time Fraction*, f_k^{max}. The other half will be used later in the demonstration.

Before choosing the value for $\overline{T}_k$, it is first necessary to present the following situation that represents the worst possibility of a class exceeding its *Maximum Time Fraction*.

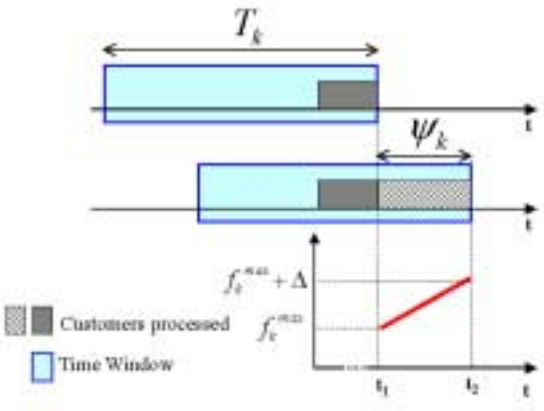

Figure 3.7: Worst situation for exceeding f_k^{max}.

Consider the situation illustrated in figure 3.7. It represents the *Time Window* of a class k in two time instants t_1 and t_2. At time instant t_1 the server ends processing a customer of class k and the *Time Fraction*, $f_k(t)$, of that class equals its *Maximum Time Fraction*, f_k^{max}. Since class k is still not *Blocked*, the server may decide to process another customer of class k. If that service happens to be the maximum processing time for class ψ_k. The question is the following: *what should*

be the size of the Time Window, T_k, such that the excess in the Time Fraction of class k is lower than $\frac{\varepsilon_k}{2\times N_i}$?

By observing figure 3.7, the value of T_k should be:

$$\Delta = \frac{\psi_k}{T_k}$$
$$\frac{\psi_k}{T_k} < \frac{\varepsilon_k}{2 \times N_i} \Leftrightarrow T_k = \frac{\psi_k}{\frac{\varepsilon_k}{2\times N_i}}$$

This result guarantees that if for each class k, $\overline{T}_k$ is assigned the value $\frac{\psi_k}{\frac{\varepsilon_k}{2\times N_i}}$, then at any given time instant, all classes will at most exceed their *Maximum Time Fractions* by $\frac{\varepsilon_k}{2\times N_i}$.

With this choice of parameters, the system is now decoupled in the form that in any given time instant, each class k is guaranteed to have an allocated time fraction of value $f_k^{max} + \frac{\varepsilon_k}{N_i}$. This value is guaranteed independently of the behaviour of the other classes.

The second assertion should now be obvious, since the instance provided ensures that each tandem queue respects the *Traffic Intensity Condition*.

■

An immediate consequence of Theorem 3.2 is the following result.

Corollary 3.1. Under the assumptions of Theorem 3.2, the *Traffic Intensity Condition* is a sufficient stability condition.

This corollary emphasizes the perspective on stability sustained by this dissertation. The stability of a queuing network is only dependent on the existence of a scheduling policy that renders the queuing network stable. The TW Controller is a mechanism that guarantees that a queuing network satisfying the *Traffic Intensity Condition* will be stabilized, even under a non-idling control policy that leads that network to instability.

This theorem, although demonstrating the stabilization properties of the *TW Controller*, requires the existence of an upper bound on the processing time of the customers. This requirement is needed to guarantee the decoupling of the queuing network. It is the firm believe of the author that this theorem is also valid without the existence of that upper bound. However, it was not possible to provide a technically sound proof of that stronger result. Therefore, the above theorem without the assumption of an upper bound on the processing time of each class should simply be stated as a conjecture. Naturally, if the conjecture turns out to be true, Corollary 3.1 will have a wider scope of applicability.

The belief that this conjecture is true is based on the extensive experimental studies conducted, where no such bound was ever imposed nor needed to stabilize all systems considered. Chapter 4 and Appendices A and B present some of those studies.

Chapter 4

Experimental Results

This chapter illustrates, by means of several computer simulations, the properties of the *Time Window Controller*. The first section describes the simulation software developed to obtain the experimental results presented in this chapter. The second section presents the queuing network topology that will serve as a test bed for the study. The third section shows how the *TW Controller* is able to stabilize a queuing network that is unstable under the *First In First Out* scheduling policy. The fourth section presents the results regarding the influence of the parameters (α_k, T_k) of the *TW Controller* on the queuing network's performance. The last section of this chapter demonstrates how the *TW Controller* not only is able to stabilize an unstable network but it is also able to improve the performance of a stable queuing network.

Although all the results presented in this chapter were obtained using a specific queuing network, appendix A and B present similar experimental studies to different network topologies and scheduling policies.

4.1 The Queuing Network Simulator

For the experimental study presented in this thesis it was necessary to develop a generic queuing network simulator. The reason for developing a new simulator instead of using an available software package was due to the necessity of implementing

idling scheduling policies in multiclass queuing networks. This task can become very complex to implement in a standard software package. Another reason was the computation of several statistics that were specific to this thesis. Due to the previous reasons, a queuing network simulator was developed in *C++* [Stroustrup, 1997] for a *Linux* workstation.

The simulation of stochastic systems is a well established field [Law and Kelton, 1991, Rubinstein and Melamed, 1998]. In the present case, the queuing network simulator was designed using an object oriented methodology [Meyer, 1997].

The queuing network simulator is composed of four objects: simulator, scheduling policy, system dynamics and data processing. The system dynamics object is designed by the user and contains the queuing network topology including the distributions for the customer arrival process and for the processing times. The scheduling policy object is also designed by the user and contains the scheduling policy that is used by each server in the queuing network. The data processing object is responsible for collecting all the relevant data of the simulation. Finally the simulator object is responsible for controlling the evolution of the simulation by interacting with the other objects. Figure 4.1 presents a diagram of the interactions between the objects that constitute the queuing network simulator.

The use of this methodology allows a large degree of flexibility on implementing complex queuing networks and scheduling policies. The program receives an input file containing the simulation parameters and creates several output files containing the simulation results. The output files can then be processed by the *Matlab* program to generate a graphic representation of the simulation results.

4.2 A Non-Acyclic Queuing Network Topology

To present an experimental study of the *Time Window Controller* properties, it is first necessary to choose a queuing network to serve as a test bed. The choice

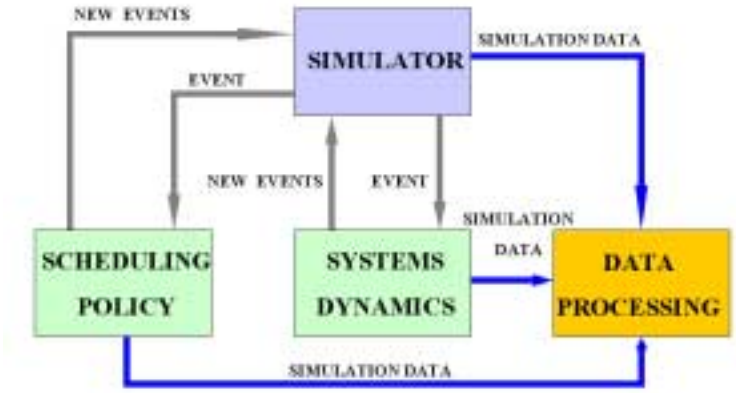

Figure 4.1: Object interaction diagram.

was a queuing network presented in [Dai, 1995] which is constituted by two service stations with six classes. Variants of this topology were studied in [Kumar, 1993] and [Whitt, 1993]. Figure 4.2 presents a diagram of the queuing network topology.

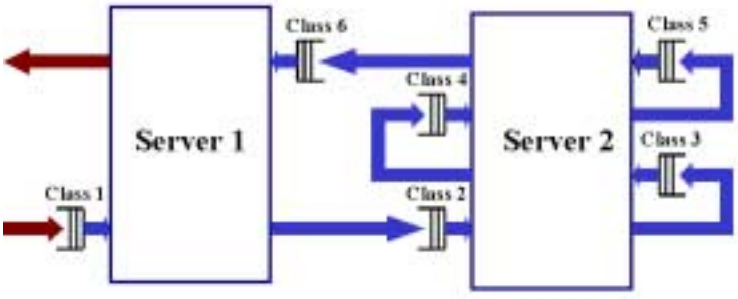

Figure 4.2: Dai's queuing network topology.

[Dai, 1995] presents a simulation of this queuing network using a deterministic setting for the processing times and for the time interval between customer arrivals to the system. He also used the *First In First Out (FIFO)* scheduling policy at each server. The parameters used in his simulation are presented in table 4.1.

The following expression demonstrates that with those parameters the *Traffic Intensity Condition* is verified for this queuing network.

Table 4.1: Queuing network parameters.

Parameter	Value
customer mean arrival rate to the system (λ)	1.000
class 1 mean processing time (μ_1)	0.001
class 2 mean processing time (μ_2)	0.897
class 3 mean processing time (μ_3)	0.001
class 4 mean processing time (μ_4)	0.001
class 5 mean processing time (μ_5)	0.001
class 6 mean processing time (μ_6)	0.899

Traffic Intensity Condition

server 1: $\lambda \times \mu_1 + \lambda \times \mu_6 = 1.0 \times 0.001 + 1.0 \times 0.899 = 0.9 < 1$

server 2: $\lambda \times \mu_2 + \lambda \times \mu_3 + \lambda \times \mu_4 + \lambda \times \mu_5 =$

$= 1.0 \times 0.897 + 1.0 \times 0.0001 + 1.0 \times 0.001 + 1.0 \times 0.001 = 0.9 < 1$

The simulation results obtained by Dai demonstrated that this queuing network topology under the *FIFO* scheduling policy presents an unstable behaviour even in a deterministic setting. These results clearly show that for this queuing network the *Traffic Intensity Condition* is not a sufficient stability condition, understanding here that the queuing network is composed by the topology and the scheduling policy. To replicate that result, the queuing network was simulated in a stochastic setting where the arrival of customers to the system is modeled by a Poisson process and the processing times at the service stations are modeled by an exponential distribution. Figures 4.3 and 4.4 present the simulation results in the form of the server and class inventory or queue length evolution for the application of the *FIFO* scheduling policy to this queuing network topology with the parameters presented in table 4.1.

The unstable behaviour of this queuing network can be observed by the evolution of the server's inventory. It is characterized by an oscillatory behaviour with the oscillation amplitude growing linearly with time. Note that the oscillations in the servers are out of phase, which means that when the inventory on one of the servers

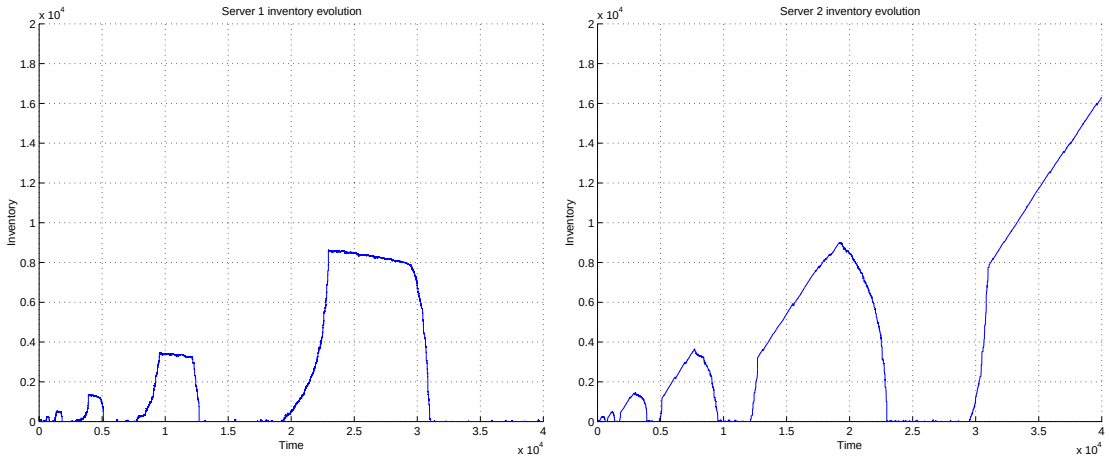

Figure 4.3: Server inventory evolution for the *FIFO* scheduling policy.

is high, the inventory on the other server is very low or even non existent.

This behaviour coincides with the one observed by Dai with the exception that the format of the inventory evolution curves is slightly different, since his simulation corresponds to a deterministic queuing network while the results presented here correspond to a stochastic queuing network.

Another way to view the inherent unstable behaviour of this queuing network that is not mentioned by Dai is to compute the long term *Time Fraction* used by each class during the simulation run.

Table 4.2 presents those results. The *long term Time Fraction* for class k, f_k^{∞}, is computed by dividing the time the server was processing customers of class k by the total simulation time. The *Min. processing rate* for class k corresponds to the minimum processing rate of customers of class i necessary to guarantee the *Traffic Intensity Condition*.

It is clear by the results, that all classes except the first one are not using enough of the available resources to be able to process enough customers to keep the system stable. Clearly the problem is with the *FIFO* scheduling policy that is not able to properly use the available resources, creating starvation cycles between the service

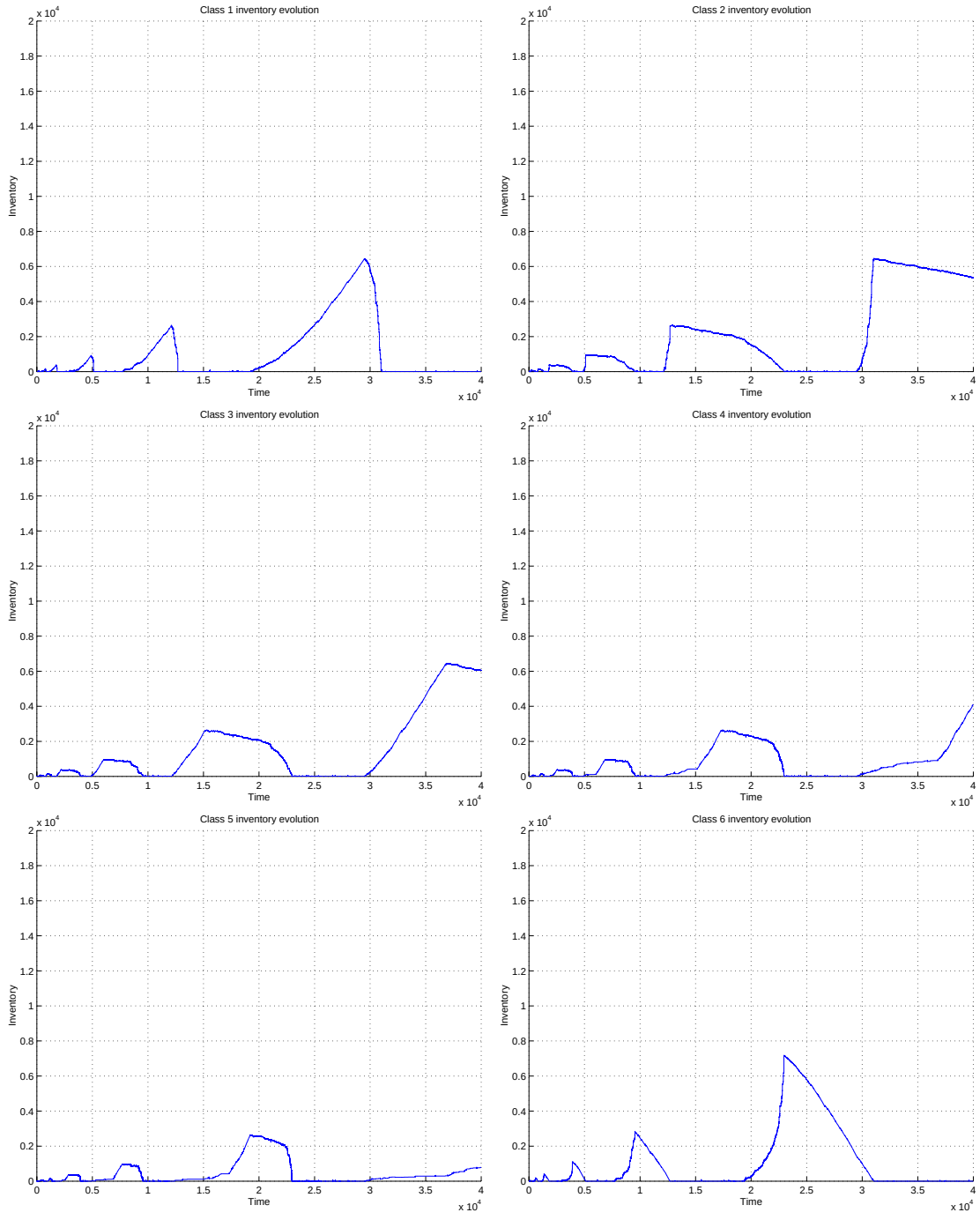

Figure 4.4: Class inventory evolution for the *FIFO* scheduling policy.

Table 4.2: Long term time fraction, f_k^∞, used by each class/server.

Class	f_k^∞	Min. processing rate
1	0.0010	0.001
2	0.7849	0.897
3	0.0007	0.001
4	0.0006	0.001
5	0.0006	0.001
6	0.5352	0.899
Server		
1 (class 1+6)	0.5362	0.9
2 (class 2+3+4+5)	0.7868	0.9

stations that effectively reduce the resource availability for each class. Seidman had already demonstrated in [Seidman, 1994] how the *FIFO* scheduling policy creates starvation cycles in a non-acyclic queuing network.

Another important property of this queuing network is that it is stable for the *First Buffer First Served (FBFS)* and *Last Buffer First Served (LBFS)* scheduling policies. This property was demonstrated in [Lu and Kumar, 1991] for all non-acyclic, stochastic, multiclass, queuing networks that possess a single entry point and only one exit point. This result will be necessary to demonstrate the performance enhancement properties of the *TW Controller* in Section 4.4.

Finally, it is necessary to define a cost function for this queuing network to be used in the following sections. From all the possible cost functions for this system, equation 4.1 presents one possible instance that correspond to a linear combination of the average inventory of customers of a given class. $\overline{I}_{Ci}$ designates the average inventory of class i customers. The objective is to penalize more the inventory that accumulates in the first classes which correspond to the initial processing stages. In this way, the system that has most customers in last stages of production will have a lower cost which is logical since we are dealing with a *push* system. Since there is no control on the arrival rate of customers, the only way to reduce the queuing

network's inventory is to push the customers towards the exit point.

$$J(\overline{I}_{C_1}, ..., \overline{I}_{C_6}) = 6 \times \overline{I}_{C1} + 5 \times \overline{I}_{C2} + 4 \times \overline{I}_{C3} + 3 \times \overline{I}_{C4} + 2 \times \overline{I}_{C5} + \overline{I}_{C6} \quad (4.1)$$

4.3 Stabilization Properties of the TW Controller

This section presents a demonstration of the stabilisation properties of the *Time Window Controller*. The objective is to use the *TW Controller* to stabilise the queuing network presented in the previous section.

The choice of parameters for the *TW Controller* will be made using the procedure presented in the proof of Theorem 3.2. The *Maximum Time Fraction*, f_k^{max}, of each class will be equal to the product of the mean effective arrival rate with the mean processing time plus a portion of the slack processing rate available at the corresponding server.

The values of α_k and T_k were chosen in a way that guarantees that the *Time Window* possesses enough memory, but their choice does not follow the procedure used in the proof of Theorem 3.2. The objective of this choice is to present the robustness of the *TW Controller* when the parameters α_k and T_k are chosen using a heuristic approach.

Table 4.3 presents the choice of parameters for the *TW Controller* using the queuing network parameters presented in table 4.1.

Figures 4.5 and 4.6 present a comparison of the server and class inventory evolution for the network with and without the *TW Controller*.

The simulation results clearly show that the *TW controller* was able to effectively stabilize the queuing network. Table 4.4 presents a comparison of some statistics obtained from the simulations.

The results in table 4.4 show that with the *TW Controller*, the servers in the queuing network are able to process enough customers of each class to keep the

Table 4.3: *TW Controller* parameters.

Parameter	Value
T_k	100
α_k	0.1
f_1^{max}	$0.001 + 0.050 = 0.051$
f_2^{max}	$0.897 + 0.025 = 0.922$
f_3^{max}	$0.001 + 0.025 = 0.026$
f_4^{max}	$0.001 + 0.025 = 0.026$
f_5^{max}	$0.001 + 0.025 = 0.026$
f_6^{max}	$0.899 + 0.050 = 0.949$

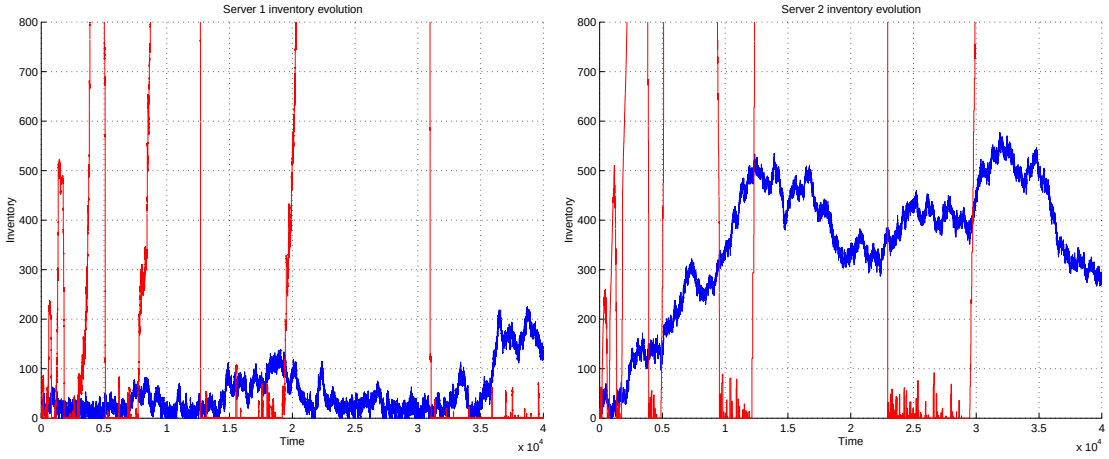

Figure 4.5: Comparison of the server inventory evolution for the Dai queuing network topology with the *FIFO* (red) and *FIFO + TW Controller* (blue) scheduling policies.

system stable. Note that the statistics presented in the table do not have any meaning for an unstable system. Since in that case there is no steady state value which implies that those statistics are only valid in the finite time interval of the simulation.

Another important statistic to notice is the *Active Idle* time at each server presented in table 4.5. This value clearly demonstrates that adding *Active Idle* time to the system is not a waste of resources but an effective way to stabilize the queuing

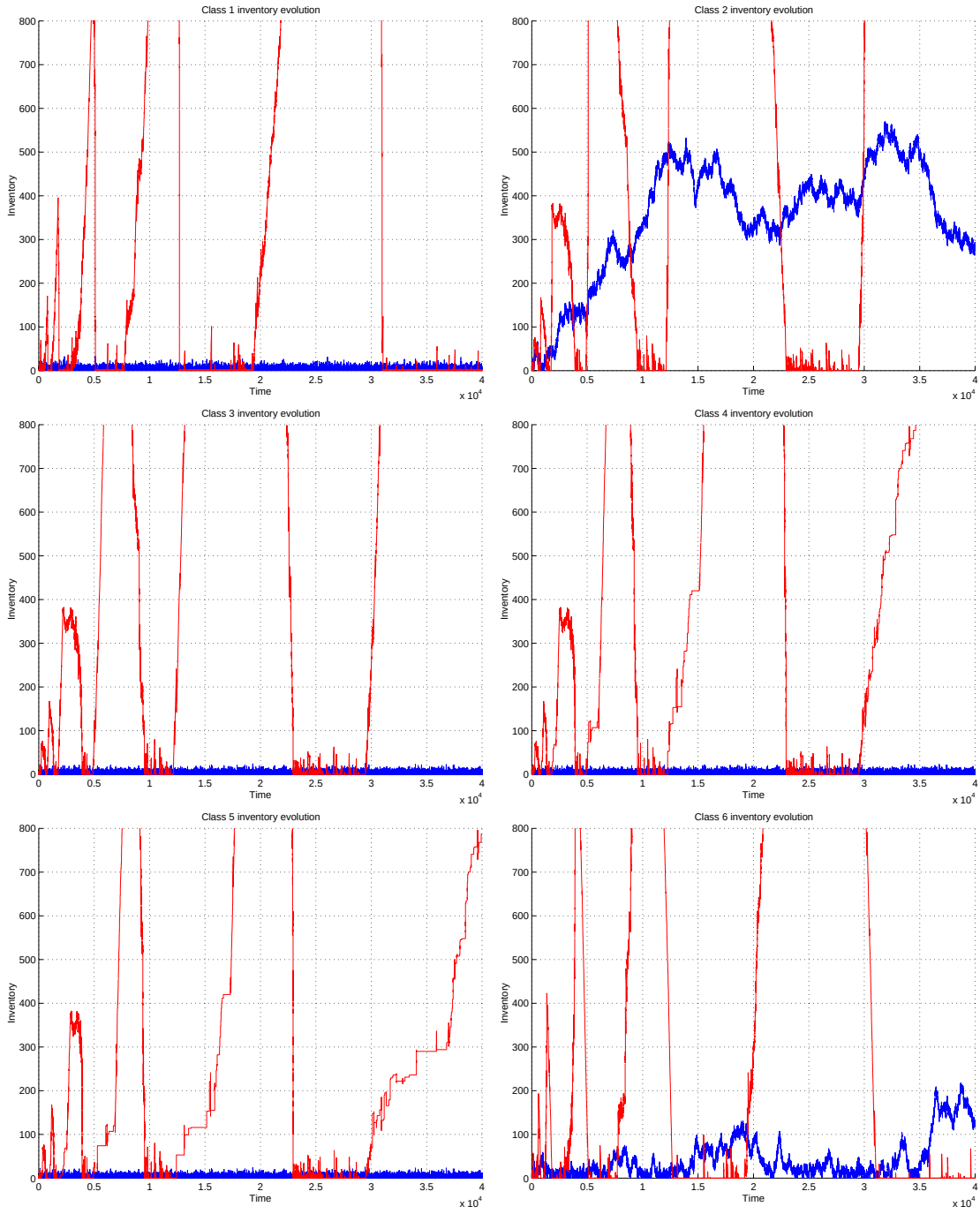

Figure 4.6: Comparison of the class inventory evolution for the *FIFO* (red) and *FIFO + TW Controller* (blue) scheduling policies.

Table 4.4: Simulation statistics for the Dai queuing network topology.

Statistic	Original	*TW Controller*
$\overline{I}_{S_1}$	2102	48
$\overline{I}_{S_2}$	4824	355
f_1^∞	0.001	0.001
f_2^∞	0.7879	0.8964
f_3^∞	0.0007	0.001
f_4^∞	0.0006	0.001
f_5^∞	0.0006	0.001
f_6^∞	0.5352	0.8945
Cost	—	1841

Table 4.5: Comparison of the *Active Idle Time* in each server.

Statistic	Original	*TW Controller*
Active Idle time at server 1	0	0.0638
Active Idle time at server 2	0	0.0994

network. Defining effective load of a server as the original load plus the active idle time, note that server 2 has an effective load of 99.94%, whereas server 1 has an effective load of 96.38 %. This explains why server 2 has a higher average inventory, since it is closer to the stability bound.

The question that arises regarding the stabilization properties of the *TW Controller* is the following: *is it necessary to perform a complete decoupling of the queuing network to stabilize an unstable queuing network?*

This question is very important, since although theorem 3.2 guarantees the stabilization property of the *TW Controller* through the complete decoupling of the queuing network, this decoupling implies a loss of flexibility by the queuing network. If the *TW Controller* is able to stabilize the queuing network without requiring a complete decoupling of the queuing network, it is likely that the sharing of resources resulting from allowing some degree of coupling would imply an improvement of performance.

To answer this question, a trial and error procedure was performed to discover a region of the *TW Controller* parameters were this property could be observed. The parameters presented in table 4.6 were chosen for the *TW Controller*. Note that with the exception of parameter f_5^{max} which is not set, all other parameters are set to a unitary value, meaning that the *TW Controller* will not have any influence on customers of those classes.

Table 4.6: *TW Controller* parameters.

Parameter	Value
T_k	100
α_k	0.01
f_1^{max}	1.0
f_2^{max}	1.0
f_3^{max}	1.0
f_4^{max}	1.0
f_5^{max}	—
f_6^{max}	1.0

The objective is to change the f_5^{max} parameter from a unitary value that correspond to the original unstable queuing network, to a value value lower than one that correspond to a queuing network where the *TW Controller* influences the processing of customers of class 5.

To demonstrate that an instance of the *TW Controller* is able to stabilize the queuing network, two simulations of different time lengths are performed for the same instance of the *TW Controller*. If the cost is approximately the same for both simulations, then that instance of the *TW Controller* is able to stabilize the queuing network. If there is an increase on the cost, that instance is not able to stabilize the queuing network.

Figure 4.7 presents the evolution of the cost for several simulations of the network with different instances of the *TW Controller*. Each instance was simulated twice with a simulation length of 20000 and 40000 time units.

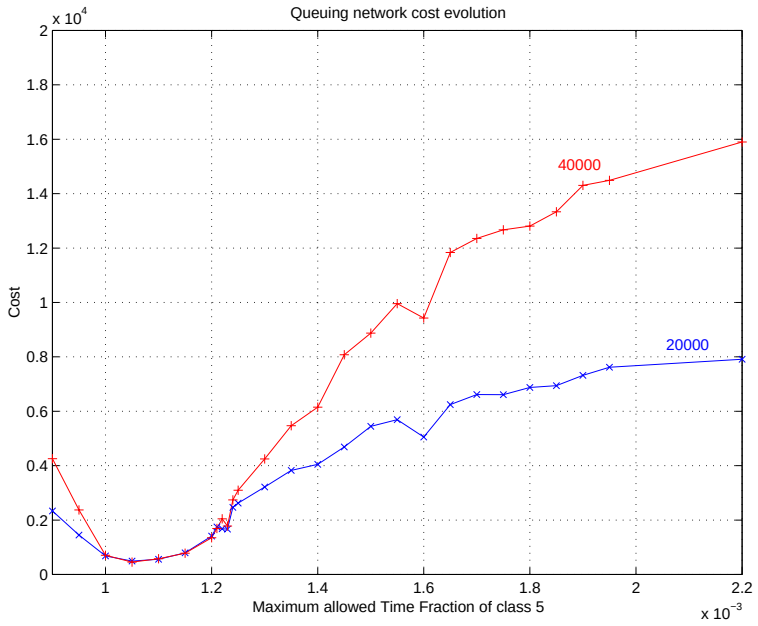

Figure 4.7: Comparison of the evolution of the queuing network cost for two simulation runs of 20000 and 40000 time units.

The results show that, although the queuing network is never completely decoupled, there is a region where the *TW Controller* is able to stabilize the queuing network. It is also possible to verify that in the stable region there are instances of the *TW Controller* with a better performance than others.

This result shows that a possible course of action to stabilize a queuing network is to first guarantee its stability through its complete decoupling. After obtaining an instance of the *TW Controller* that stabilizes the queuing network it is then possible to search the parameter's space to obtain other instances which are able to improve its performance.

4.4 The TW Controller Parameters

The objective of this section is to use the queuing network presented in Section 4.2 for an experimental study of the *TW Controller's* sensitivity relatively to parameters

α_k and T_k.

Table 4.7 presents a set of parameters for the *TW Controller* that correspond to an instance where it is able to stabilize Dai's network.

Table 4.7: *TW Controller* parameters.

Parameter	Value
T_k	—
α_k	—
f_1^{max}	0.0011
f_2^{max}	0.9967
f_3^{max}	0.0011
f_4^{max}	0.0011
f_5^{max}	0.0011
f_6^{max}	0.9989

The first experiment consists on observing the influence of T_k on the performance of the queuing network. For this end, a set of simulations was performed using the parameters in table 4.7 with different values for T_k and α_k. Figure 4.8 presents the results of those simulations in the form of the evolution of the cost with parameter T_k.

It is possible to observe that the performance of the system as a function of T_k depends significantly on the choice of α_k. For a given α_k, performance improves as T_k grows. This behaviour should not be too surprising, given that small values of T_k imply a shorter memory. One should expect degradation or even instability as T_k approaches zero.

The smaller the value of T_k, more heavily does performance depend on α_k. Curiously, performance is worse for α_k close to zero, for any value of T_k. This behaviour is in line with the discussion made in Section 3.2 (Figure 3.4).

Finally, for a fixed α_k, as T_k grows there is a point after which the added memory through the increase in T_k does not translate into any performance gain. This is due the fact that a non-zero α_k reduces the impact of customers processed in a relatively

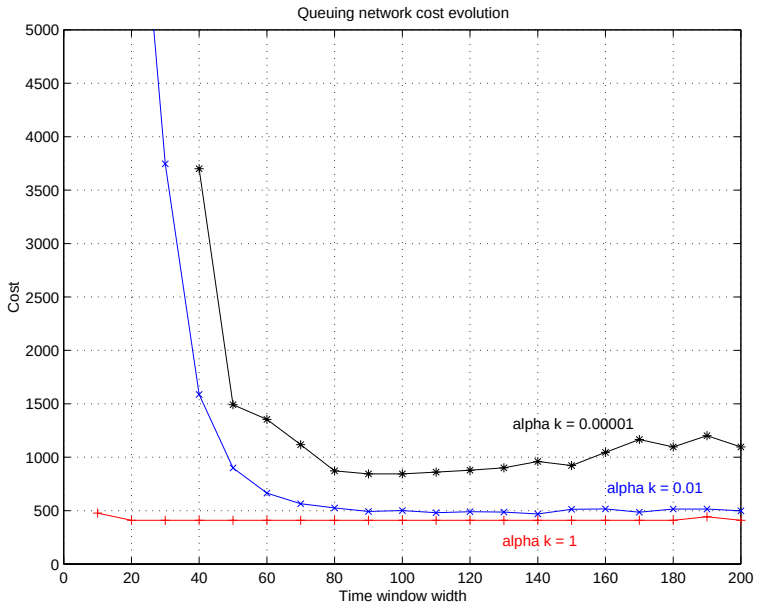

Figure 4.8: Evolution of the cost with T_k.

distant past.

The second experiment consists on performing the same procedure for α_k, where figure 4.9 presents the corresponding cost evolution

The simulation results show that the influence of α_k is not as obvious as could be expected. For $T_k = 10$, a small size window, although performance degrades as α_k approaches zero, the apparently odd behaviour is the fact that it improves so much as α_k approaches 1. Whereas the first situation is again in line with the discussion of Section 3.2, it could be surprising to observe that when $\alpha_k = 1$, a window of size 10 leads to a performance so close to the one achieved with a window of size 100 or 1000. Note that, for $\alpha_k = 1$, $e^{-10\alpha_k}$ is already a very small number, which means that for wider windows, like 100 or 1000, the extra memory is almost insignificant. This explains why performance is so close for the three sizes tested when α_k is 1.

As to the other two values of T_k, the optimal choice of α_k lies somewhere between 0.1 and 0.3, meaning that there is a need for some significant memory in order to

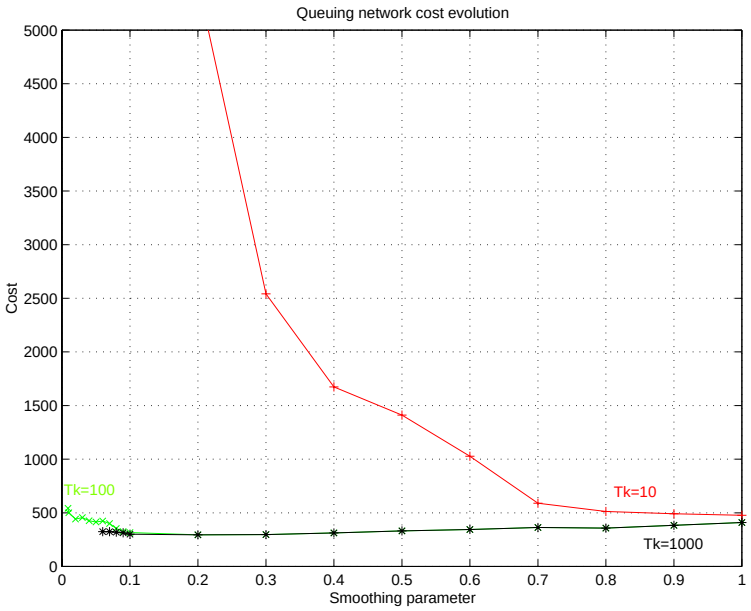

Figure 4.9: Evolution of the cost with α_k.

achieve the best results.

This shows that an appropriate choice of α_k and T_k is not mutually independent or simple. Their value is crucial for the performance of the controller. However when stability is the issue, it is not complicated to find an appropriate set of values.

4.5 Performance Properties of the TW Controller

Having presented the stabilization properties of the *TW Controller* in Section 4.3, the question that this section addresses is if the *TW Controller* presents any advantages when dealing with stable queuing networks. Remark 3.1 clearly opens the possibility of improving the performance of a queuing network through the TW Controller.

It was mentioned in Section 4.2 that Dai's network is stable under the *Last Buffer First Served (LBFS)* scheduling policy. It is also shown in [Lu et al., 1994] that the *LBFS* scheduling policy is able to achieve a good performance in comparison with

other distributed scheduling policies. Taking into account these results, a possible demonstration of the performance enhancement properties of the *TW Controller* would be to use Dai's network with the *LBFS* scheduling policy as a test bed to demonstrate the performance improvement capabilities of the *TW Controller*.

The problem resides on how to choose the *TW Controller* parameters. Given a choice of values for α_k and T_k, there are still for this network six f_k^{max} parameters that need to be set.

Instead of starting to explore the parameter space in a blind manner, a better approach would be to observe the dynamics of this queuing network. In the topology presented in figure 4.2, customers of class 6 due to the *LBFS* scheduling policy will always have priority over customers of class 1. The parameters in table 4.1 imply that customers of class 6 have a larger mean processing time than customers of class 1. It is then obvious that server 1 will spend most of its time processing customers of class 6.

Using this line of thought, it is also possible to speculate that, in some occasions server 2 will be starved of customers, due to server 1 being processing customers of class 6 instead of customers of class 1, which are needed to feed server 2.

Taking these factors into account, a possible way to use the *TW Controller* to improve the performance of this network is to restrict the myopic behaviour of the *LBFS* scheduling policy at server 1 by reducing the *Maximum Time Fraction* of class 6, f_6^{max}, to a value lower that one.

To test this hypothesis, the *TW Controller* was set with the parameters presented in table 4.8, where the value of the f_6^{max} parameter will be changed from the unitary value corresponding to the *LBFS* scheduling policy to values lower than one.

Figure 4.10 presents the results of several simulations performed with different values of f_6^{max} in the form of the evolution of the cost and average *Active Idle* time at server 1 with the f_6^{max} parameter.

Table 4.8: *TW Controller* parameters.

Parameter	Value
T_k	100
α_k	0.01
f_1^{max}	1.0
f_2^{max}	1.0
f_3^{max}	1.0
f_4^{max}	1.0
f_5^{max}	1.0
f_6^{max}	—

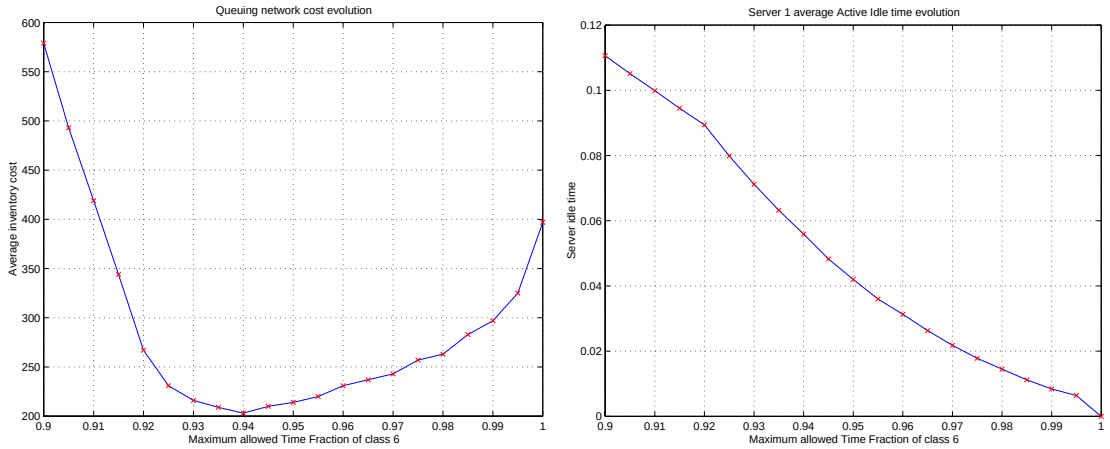

Figure 4.10: Evolution of the queuing network cost (left) and average *Active Idle* time of server 1 (right) for the *LBFS + TW Controller* scheduling policy with the f_6^{max} parameter.

The results clearly show that the *TW Controller* is able to improve the performance of the queuing network. The improvement in some instances is close to 50%. Connected with this improvement is the inclusion of *Active Idleness* to the behaviour of server 1, since all classes in server 2 have a unitary value for their *Maximum Time Fraction*. Figure 4.11 presents the evolution of the average inventory in the system. It is also possible to observe a significant improvement for some instances, which implies a reduction in the customers lead time due to Little's Law. Note that for

this alternative cost function, the optimal value of f_6^{max} is between 0.95 and 0.97, and the cost reduction relatively to $f_6^{max} = 1$ is a little over 25%. The average inventory for $f_6^{max} = 1$ is 80 and the average inventory for $f_6^{max} = 0.955$ is 57.

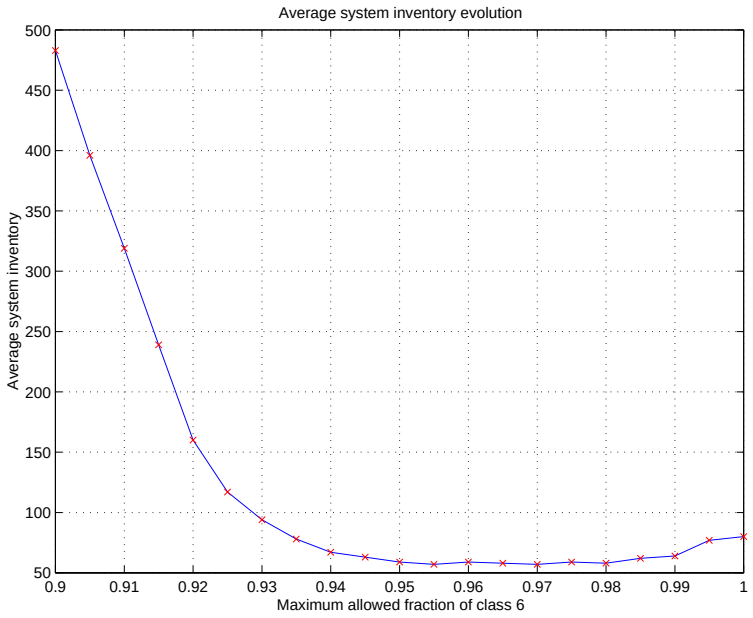

Figure 4.11: Evolution of the average system inventory with the f_6^{max} parameter.

To further show the performance improvement, Figure 4.12 presents a comparison of the server inventory evolution of the original queuing network with the *LBFS* scheduling policy and the same queuing network with the *TW Controller*, where the f_6^{max} parameter is set to 0.94.

By observing Figure 4.12 it is easy to note that, during the simulation, the *TW Controller* is able to maintain a lower inventory value for server 2. It is also possible to notice that the inventory evolution in server 1 with the *TW Controller* has a lower chattering than the inventory evolution with the original *LBFS* scheduling policy.

Another way to present the performance improvement obtained by the *TW Controller*, is to compare the cumulative probability distribution obtained from the two simulations discussed above. Figure 4.13 and 4.14 present the cumulative probability

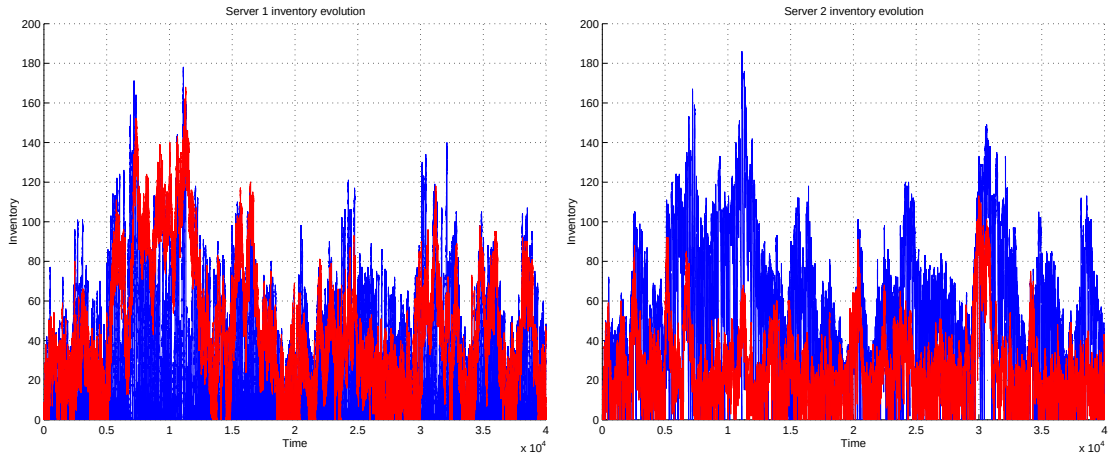

Figure 4.12: Comparison of the server inventory evolution for the *LBFS* (blue) and *LBFS + TW Controller* (red) scheduling policies.

distribution for the server and class inventory.

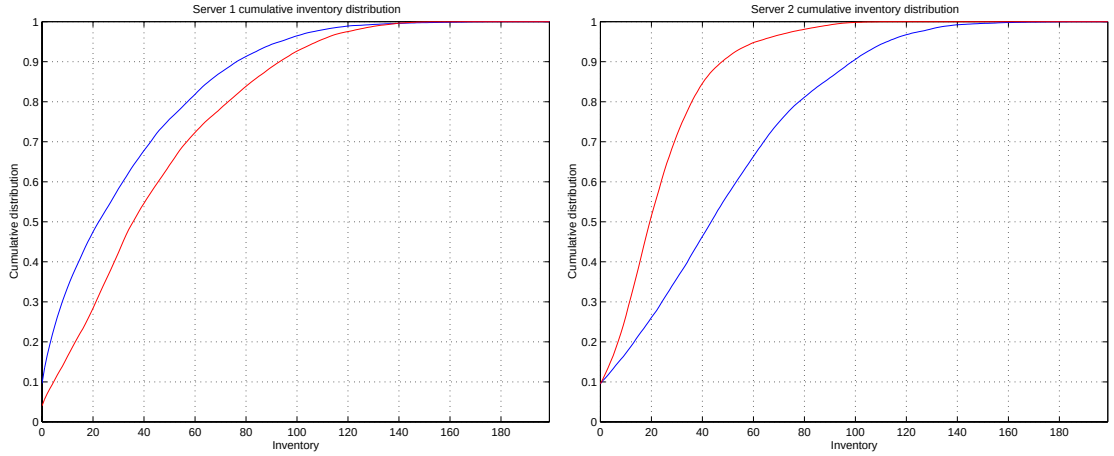

Figure 4.13: Comparison of the server cumulative probability distribution for the *LBFS* (blue) and *LBFS + TW Controller* (red) scheduling policies.

Figure 4.13 shows that the probability of having a larger inventory value in server 2 is much lower with the *TW Controller*. This result is obtained at the cost of a slight increase on the probability of having a larger inventory in server 1 with the *TW Controller*, but it is this slight increase in server 1 that allows the substantial

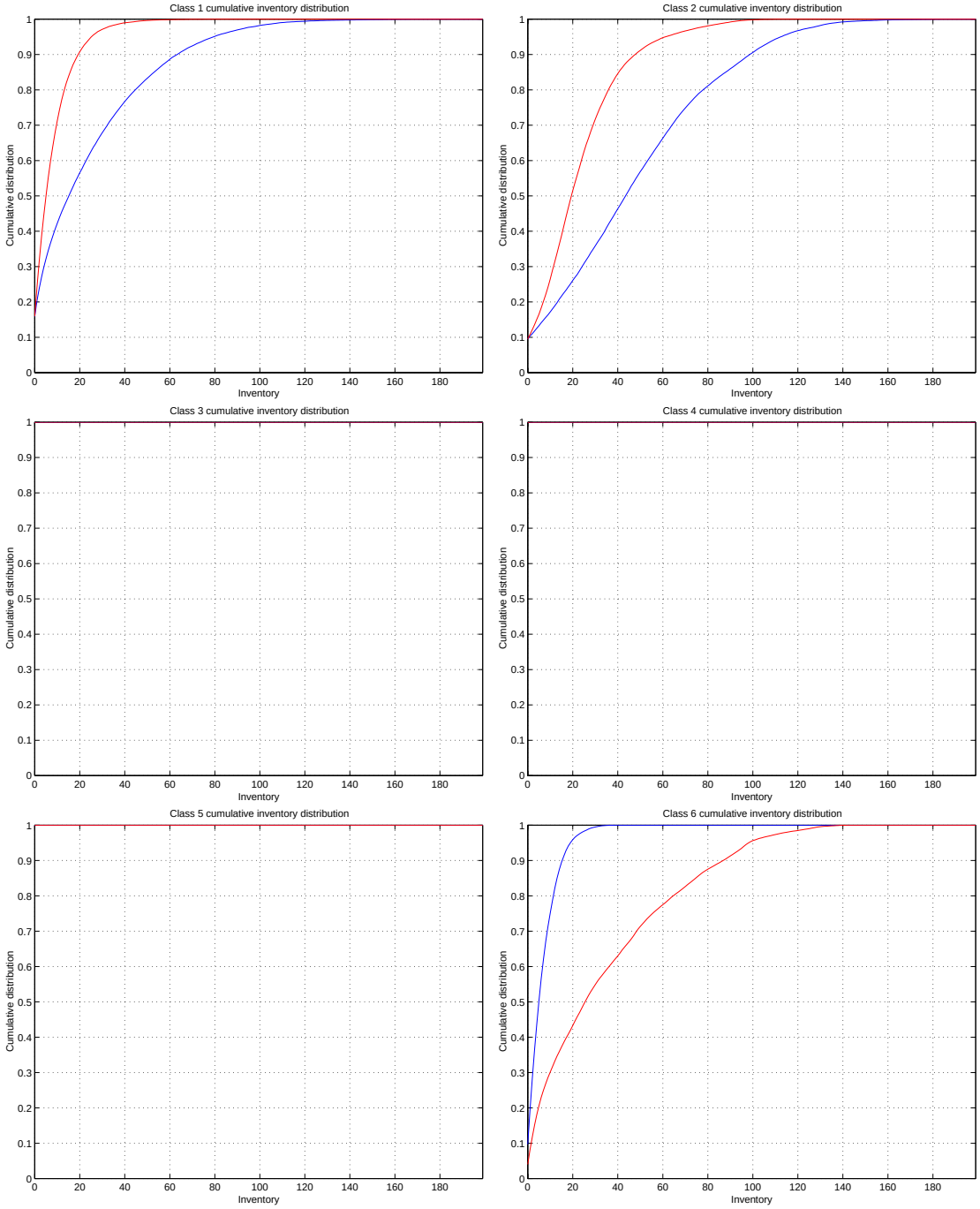

Figure 4.14: Comparison of the class cumulative probability distribution for the *LBFS* (blue) and *LBFS + TW Controller* (red) scheduling policies.

decrease in the average inventory at server 2. This behaviour clearly shows that, although the *TW Controller* only affects the behaviour of class 6 in server 1, the consequences are felt by the entire queuing network in the form of an improvement of performance.

As a final note, it should be stressed that the *Time Window* size is much smaller than the size implied by the proof of Theorem 3.2. Also, no upper bound was assumed for the processing time of each class. This fact reinforces the belief that there is no need to impose upper bounds on the processing times for Theorem 3.2 to hold.

Chapter 5

Conclusions

The main conclusions from this thesis can be resumed in the following three items:

- *Active Idleness* is not a waste of resources but an effective tool to stabilize non-acyclic, multiclass, queuing networks.
- The *Time Window Controller* is an effective and simple implementation of the *Active Idleness* concept.
- Even when stability is not in question, *Active Idleness* can be used as a performance improvement tool.

The first item represents the main contribution of this thesis. Although other authors came across the advantages of using idling policies to solve the stability problem, the author does not know of any other work that looks at idle behaviour as the key to the stabilization of multiclass, non-acyclic, queuing networks. Moreover, not only idleness is presented as a stabilization key, but also a systematic procedure based on idleness is proposed to stabilize a given network. This statement is clearly substantiated by the results presented in Chapter 4 and appendices. It is expected that this concept of *Active Idleness* is received with skepticism by most readers. The reason for that is the link that is made between idleness and waste of resources. This is a notion that has its roots in traditional queuing network theory which deals mainly with Jackson-type networks.

It must be emphasized that the concept of *Active Idleness* has its scope of use in the class on non-acyclic, multiclass, queuing networks. For this type of networks it is not correct to use the same line of thought applied to simpler systems like Jackson's networks. The results presented in this thesis firmly corroborate this.

The second conclusion summarizes the results obtained for the *Time Window Controller*. The *TW Controller* is a possible implementation of the *Active Idleness* concept. It is able to fulfill all of its requirements. The *TW Controller* is of very simple implementation and most importantly, keeps the scheduling policy distributed, although using some non-local static information. The experimental results presented in Chapter 4 and appendices, in conjunction with Theorem 3.2, show that the *TW Controller* is able to stabilize queuing networks with different topologies and scheduling policies.

The last conclusion refers to the remarkable experimental results presented. *Active Idleness* is not only a concept for stabilizing unstable queuing networks, but also an effective way to improve the performance of stable networks. This thesis presents several instances of such improvements, where the cost could be reduced by 50% in comparison to relatively good scheduling policies, as is the case of the *Last Buffer First Served*.

The main contribution of this thesis is the demonstration that *Active Idleness* is a key to stabilizing multiclass, non-acyclic, queuing networks. It is important to stress that *Active Idleness* is not a new scheduling policy, but a mechanism to insert the necessary amount of non-local information into the queuing network to render it stable. The influence on the scheduling policy resumes to blocking some customers of a given class of being processed during a given time interval. It does not change the way in which the original scheduling policy chooses the customers to be processed. In essence it implements a global feedback procedure that is static in time but feeds each server with enough information to guarantee the stability of

the entire system. It also brings together into the same framework the scheduling policy and the admission policy of a queuing network.

This is due to the double role that the *TW Controller* performs when it blocks a customer of a given class from being processed. In one hand, the controller blocks that class to ensure that other classes have access to the server resources. This ensures the queuing network stability. On the other hand, if the class corresponds to customers arriving to the queuing network, the *TW Controller* is in fact implementing an admission policy, since it restricts in some form, the admission of those customers to the first server in the queuing network.

It should also be stressed that as long as the *Maximum Time Fractions* of classes visiting a given server add up to more than 1, the controller is effectively allowing flexible sharing of resources. This feature allows for some input burstiness to get transferred inside and through the network. However, the individual value of f_k^{max} determines the maximum amount of burstiness permitted. The *TW Controller* acts as a burstiness filter. It is the filtering property that is responsible for a variance reduction in the overall network, thus leading to performance improvements.

Establishing that the *Traffic Intensity Condition* as a sufficient stability condition requires that one obtains a set of decoupled tandem networks, for which that condition is sufficient.

According to recent research, [Cassandras and Lafortune, 1999, Adler, 1998], it appears that for some tandem networks, namely the GI/GI/1 queue, the *Traffic Intensity Condition* may not be a sufficient stability condition. Typically, such networks exhibit an arrival process or service times following heavy tail distributions. The *TW Controller* is not able to stabilize such tandem networks as long as there is no non-idling policy which stabilizes them. Therefore, given a general queuing network, if through decoupling, at least one of the tandem networks generated is in such a class, the *TW Controller* will not be able to stabilize the entire network also.

However, given its filtering properties, the potential unstable behaviour will be restricted to its source and will not be propagated through the whole network. For instance, if the instability source is only due to the arrival process of some class, the only buffer that may go out of bounds is its own input buffer, leaving the rest of the network unaffected

Although this does not solve the stability problem for those systems, it is the author's belief that it is preferable to contain instability to close quarters, rather than allow it to spread through the entire network.

5.1 Future Work

There are still some open issues that require future work. The foremost is a rigorous demonstration of the stabilization property of the *TW Controller*, without a limit on the customer's processing time. Probably, the only restriction necessary is that all moments of the customer's processing time distributions are finite.

Another important issue is the development of an optimization procedure to use the *TW Controller* as a performance improving tool. This is important, since in more complex network topologies, it is not possible to use the same heuristic trial and error procedure used in Chapter 4. It would be very interesting to compare the performance of the *LBFS* policy supervised by the *TW Controller* with the non-local *Fluctuation Smoothing (FS)* policy presented in [Lu et al., 1994]. It would also be interesting to test if adding the *TW Controller* to the *FS* policies would originate significant performance improvements.

Since the *TW Controller* is a possible implementation of the *Active Idleness* concept, it might be interesting to explore other alternatives. One possibility is the use of regulators in conjunction with the *General Processor Sharing* scheduling policy. Although this might seem similar to the work presented in [Parekh and Gallager, 1993, Parekh and Gallager, 1994], these papers did not use the concept of

Active Idleness or dealt with the stability issue.

Finally, stability of a queuing network is asserted by investigating the stability of tandem networks, when using Theorem 3.2. Also, for tandem networks it is a known fact that idling policies are not necessary to address issues like stability and performance optimization. In a single input, single output tandem network, each server should work at maximum speed, otherwise it only adds waiting time to customers.

Therefore, under the framework of the *TW Controller*, future research on queuing networks stability should concentrate on determining sufficient stability conditions for single input, single output tandem networks, given that the presence of heavy tail distributions may be a problem.

Appendix A

Lu and Kumar Example

[Lu and Kumar, 1991] presented a queuing network that is unstable under a buffer priority discipline. Figure A.1 presents a diagram of the queuing network topology which is composed by two servers with four classes.

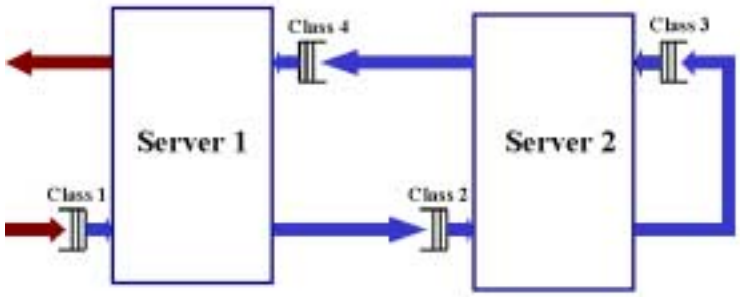

Figure A.1: Lu and Kumar's queuing network topology.

The scheduling policy for which this queuing network presents an unstable behaviour is a mixture of the *First Buffer First Served (FBFS)* with the *Last Buffer First Served (LBFS)* scheduling policies, that is, the first server uses the *LBFS* policy and the second server uses the *FBFS* policy.

[Dai and Weiss, 1996] further refined the results obtained by Lu and Kumar, demonstrating that a sufficient condition for this queuing network to be stable with this scheduling policy is that along with the *Traffic Intensity Condition* it is neces-

sary that $\lambda \times (\mu_2 + \mu_4) < 1$.

Table A.1: Queuing network parameters.

Parameter	Value
λ	1.00
μ_1	0.01
μ_2	0.90
μ_3	0.01
μ_4	0.90

Table A.1 presents a set of parameters for this queuing network that in conjunction with the *FBFS* (server 1) + *LBFS* (server 2) scheduling policy results in an unstable queuing network, although the parameters respect the *Traffic Intensity Condition*.

Traffic Intensity Condition

server 1: $\lambda \times \mu_1 + \lambda \times \mu_4 = 1.0 \times 0.01 + 1.0 \times 0.9 = 0.91$

server 2: $\lambda \times \mu_2 + \lambda \times \mu_3 = 1.0 \times 0.9 + 1.0 \times 0.01 = 0.91$

Note that, although the parameters respect the *Traffic Intensity Condition*, they do not respect Dai and Meyn's necessary stability condition, given that

$$\lambda \times (\mu_2 + \mu_4) = 1.0 \times (0.9 + 0.9) = 1.8 \not< 1 \tag{A.1}$$

Figure A.2 presents the results obtained from the simulation of the queuing network in the form of the server's inventory evolution. Clearly the results show that the queuing network is unstable.

Following the procedure presented in section 4.3, it will be demonstrated through simulation, that the *TW Controller* is able to stabilize this queuing network. The choice of parameters for the *TW Controller* was performed by allocating to each

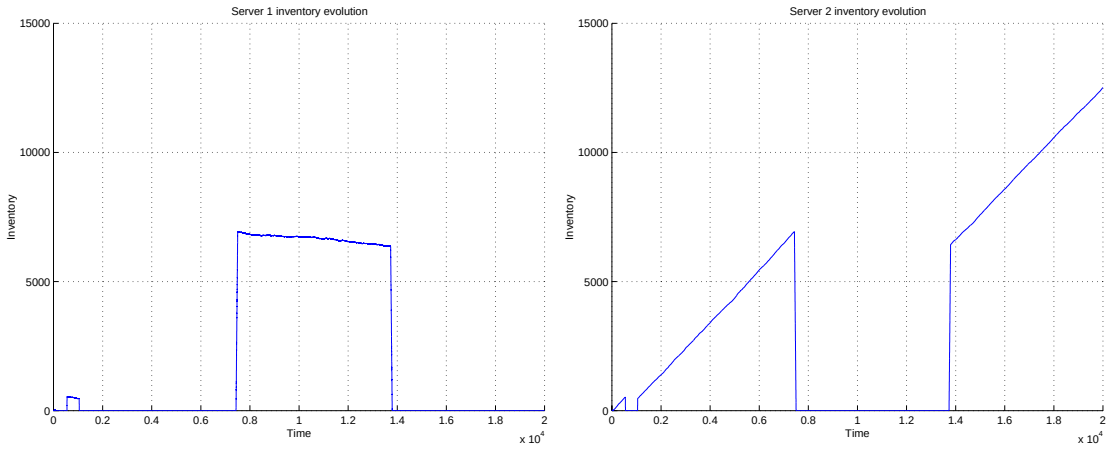

Figure A.2: Inventory evolution for the *FBFS* (server 1) + *LBFS* (server 2) scheduling policy.

class a *Maximum Time Fraction*, f_k^{max}, proportional to $\lambda.\mu_k$. The surplus capacity was shared between all classes. Table A.2 presents the choice of parameters for the *TW Controller*.

Table A.2: *TW Controller* parameters.

Parameter	Value
T_k	0.010
α_k	100.0
f_1^{max}	0.02
f_2^{max}	0.98
f_3^{max}	0.02
f_4^{max}	0.98

Figure A.3 presents a comparison of the server's inventory evolution for the unstable buffer priority scheduling policy used by Lu and Kumar with the same scheduling policy supervised by the *TW Controller*.

Table A.3 presents a comparison of some relevant statistics obtained from the simulations. The cost function used is presented in equation A.2.

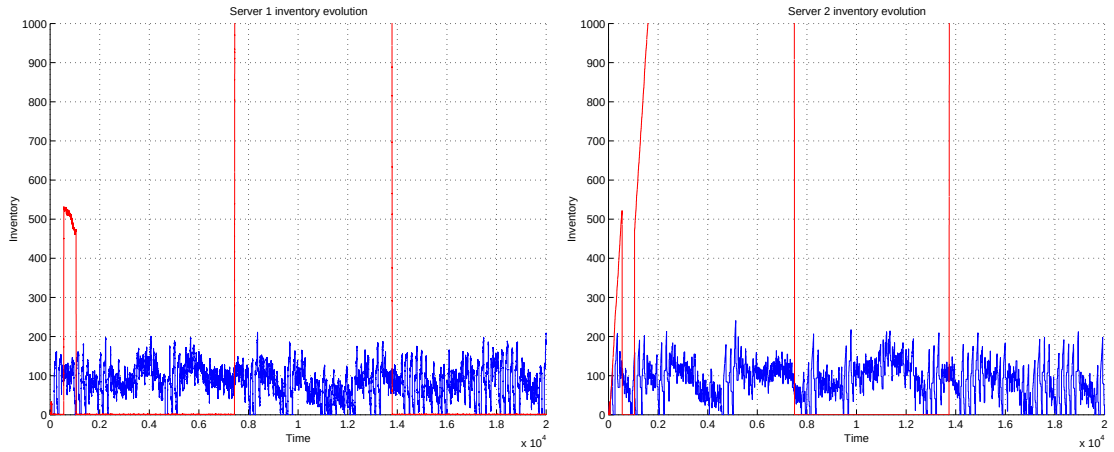

Figure A.3: Comparison of the inventory evolution for the *FBFS* (server 1) + *LBFS* (server 2) (red) and *FBFS* (server 1) + *LBFS* (server 2) + *TW Controller* (blue) scheduling policies.

Table A.3: Comparison of the relevant statistics obtained from the simulation.

Statistic	Original	*TW Controller*	Statistic	Original	*TW Controller*
$\overline{I}_{S_1}$	2109	85	f_1^{∞}	0.00993	0.00988
$\overline{I}_{S_2}$	4135	90	f_2^{∞}	0.65885	0.90928
$\overline{I}_{C_1}$	1018	23	f_3^{∞}	0.00369	0.00979
$\overline{I}_{C_2}$	1959	66	f_4^{∞}	0.34125	0.89133
$\overline{I}_{C_3}$	2176	24	$J(\overline{I}_{C_1}, ..., \overline{I}_{C_4})$	—	399
$\overline{I}_{C_4}$	1090	62			

Table A.4: Comparison of the Active Idle Time used by each server.

Statistic	Original	*TW Controller*
Active idle time at server 1	0	0.0252
Active idle time at server 2	0	0.0224

$$J(\overline{I}_{C_1}, ..., \overline{I}_{C_4}) = 4 \times \overline{I}_{C1} + 3 \times \overline{I}_{C2} + 2 \times \overline{I}_{C3} + \overline{I}_{C4} \tag{A.2}$$

The results show that with Lu and Kumar's scheduling policy, the capacity used by some of the classes is not enough to cope with the arriving rate of customers.

This clearly shows that there is a starvation phenomena generating a loss of capacity which makes the system unstable. In contrast, the *TW Controller* is able to stabilize the queuing network, inserting in the process periods of *Active Idleness* in each server as shown in Table A.4.

It is also possible to show that, similar to what was presented in section 4.5, the *TW Controller* is able to improve the performance of a stable queuing network. To demonstrate this result for the Lu and Kumar example, the *Last Buffer First Served* scheduling policy was chosen since it is a stable scheduling policy for this queuing network topology [Lu and Kumar, 1991]. The choice of the *TW Controller* parameters was performed using the same procedure used for Dai's network in section 4.5. That is, since in server 1 customers of class 4 have always priority over customers of class 1, and since the average processing time of customers belonging to class 4 is very large in comparison with customers of class 1, it is obvious that server 1 will spend most of the time serving customers of class 4. This means that in some occasions this behaviour could starve server 2. For this reason, the *Maximum Time Fraction* of class 4 was set to a value slightly lower than 1. Table A.5 presents the parameters chosen for the *TW Controller*.

Table A.5: *TW Controller* parameters.

Parameter	Value
T_k	0.010
α_k	100.0
f_1^{max}	1.0
f_2^{max}	1.0
f_3^{max}	1.0
f_4^{max}	0.95

Figure A.4 presents a comparison of the servers' inventory evolution for the *LBFS* scheduling policy with the *LBFS* supervised by the *TW Controller*.

Figure A.4 shows a substantial reduction of the inventory for server 2. This result

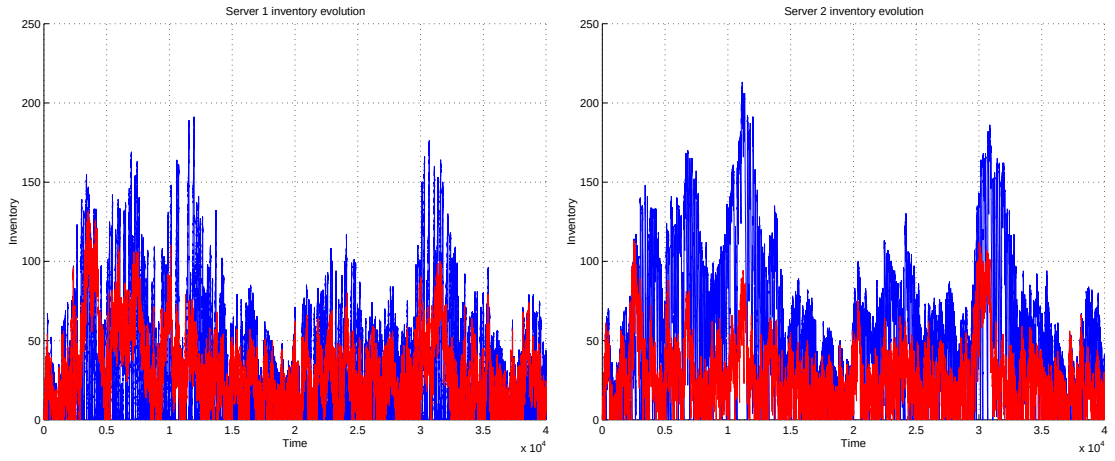

Figure A.4: Comparison of the server inventory evolution for the *LBFS* (blue) and *LBFS + TW Controller* (red) scheduling policies.

is confirmed by the statistics obtained for both simulations presented in Tables A.6 and A.7. Note the reduction of more than 50% for the cost and of 37% for the network's average inventory.

Table A.6: Comparison of the relevant statistics obtained from the simulations.

Statistic	Original	*TW Controller*	Statistic	Original	*TW Controller*
$\overline{I}_{S_1}$	34	28	f_1^{∞}	0.00992	0.00992
$\overline{I}_{S_2}$	54	26	f_2^{∞}	0.90453	0.90493
$\overline{I}_{C_1}$	28	10	f_3^{∞}	0.00987	0.00987
$\overline{I}_{C_2}$	54	26	f_4^{∞}	0.89351	0.89381
$\overline{I}_{C_3}$	0.3	0.3	$J(\overline{I}_{C_1}, ..., \overline{I}_{C_4})$	281	136
$\overline{I}_{C_4}$	6	18			

Table A.7: Comparison of the *Active Idle time* enforced in each server.

Statistic	Original	*TW Controller*
Active Idle time at server 1	0	0.0312
Active Idle time at server 2	0	0

To further stress the performance improvement, Figure A.5 presents a compari-

son of the servers' inventory cumulative probability distributions.

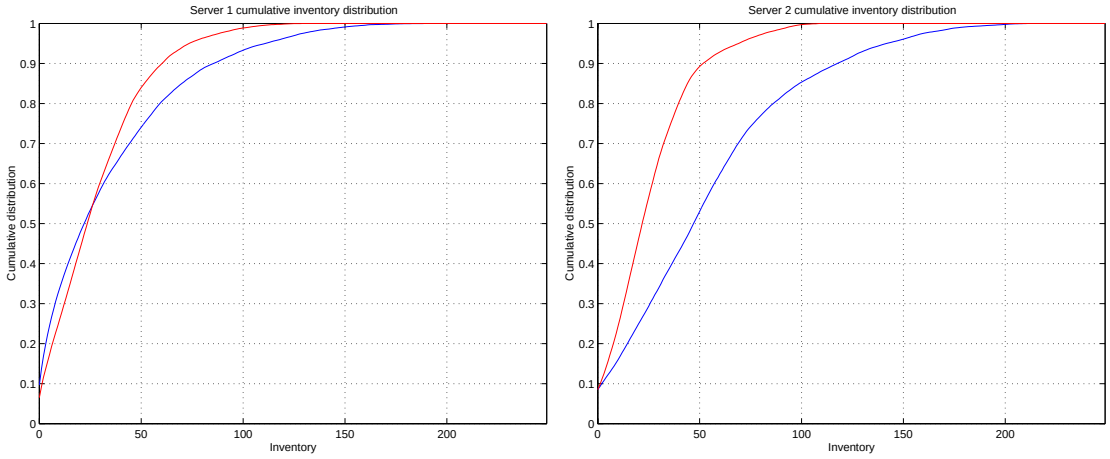

Figure A.5: Comparison of the server inventory cumulative probability distributions for the *LBFS* (blue) and *LBFS + TW Controller* (red) scheduling policies.

The results show that the *TW Controller* was able to improve the performance of server 1 and 2 by reducing the probability of those servers having large values of inventory. It should be noted again that this improvement was made by only adding a small amount of *Active Idleness* to the behaviour of server 1. Note that, since only class 4 is influenced by the *TW Controller*, all *Active Idle* behaviour refers to customers of that class. This implies, as can be observed in table A.6, an increase in the average number of class 4 customers. This reduction of the processing resources available for class 4 customers was enough for the dramatic improvement of the network's performance.

Appendix B

Seidman Example

[Seidman, 1994] presented a queuing network topology that in connection with the *First In First Out (FIFO)* scheduling policy resulted in an unstable queuing network. Figure B.1 presents a diagram of the queuing network topology, which is constituted by four servers with twelve classes.

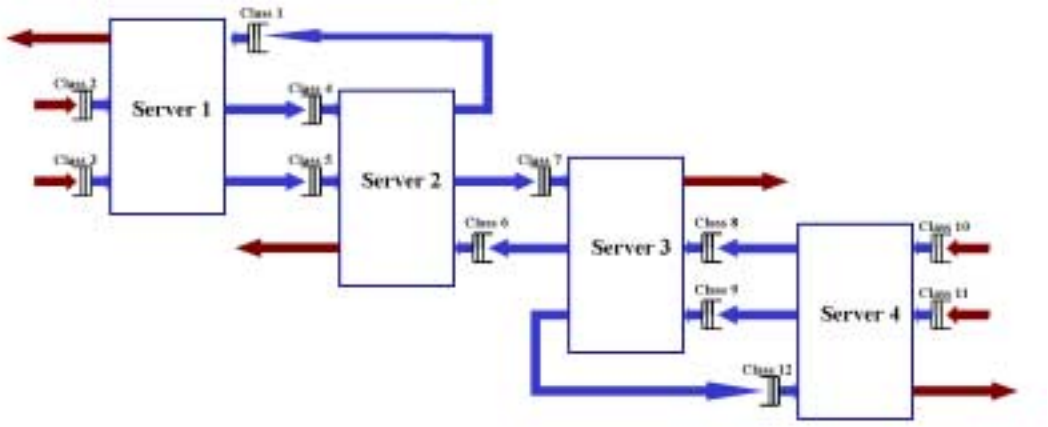

Figure B.1: Seidman's queuing network topology.

Table B.1 presents a set of parameters for this queuing network that in conjunction with the *FIFO* scheduling policy results in an unstable queuing network, although the parameters respect the *Traffic Intensity Condition*.

Table B.1: Queuing network parameters.

Parameter	Value	Parameter	Value
λ_2	1.000	μ_5	0.001
λ_3	1.000	μ_6	0.900
λ_{10}	1.000	μ_7	0.900
λ_{11}	1.000	μ_8	0.001
μ_1	0.900	μ_9	0.001
μ_2	0.001	μ_{10}	0.001
μ_3	0.001	μ_{11}	0.001
μ_4	0.001	μ_{12}	0.900

Traffic Intensity Condition

server 1: $\lambda \times \mu_1 + \lambda \times \mu_2 + \lambda \times \mu_3 = 1.0 \times 0.9 + 1.0 \times 0.01 + 1.0 \times 0.01 = 0.902$

server 2: $\lambda \times \mu_4 + \lambda \times \mu_5 + \lambda \times \mu_6 = 1.0 \times 0.01 + 1.0 \times 0.01 + 1.0 \times 0.9 = 0.902$

server 3: $\lambda \times \mu_7 + \lambda \times \mu_8 + \lambda \times \mu_9 = 1.0 \times 0.9 + 1.0 \times 0.01 + 1.0 \times 0.01 = 0.902$

server 4: $\lambda \times \mu_{10} + \lambda \times \mu_{11} + \lambda \times \mu_{12} = 1.0 \times 0.01 + 1.0 \times 0.01 + 1.0 \times 0.9 = 0.902$

Figure B.2 presents the results obtained for a simulation of the queuing network with the *FIFO* scheduling policy using the queuing network parameters presented in Table B.1. Clearly, the results show that the queuing network is unstable.

Following the procedure presented in section 4.3, it will be demonstrated through a simulation that the *TW Controller* is able to stabilize this queuing network. The choice of parameters for the *TW Controller* was made by allocating to each class a *Maximum Time Fraction*, f_k^{max}, proportional to $\lambda_k.\mu_k$. The surplus capacity was shared between all classes. Table B.2 presents the choice of parameters for the *TW Controller*.

Figure B.3 presents a comparison of the server inventory evolution of the *FIFO* scheduling policy with the same scheduling policy supervised by the *TW Controller*.

Tables B.3 and B.4 present a comparison of some relevant statistics obtained from the simulations. The cost function used is presented in equation B.1.

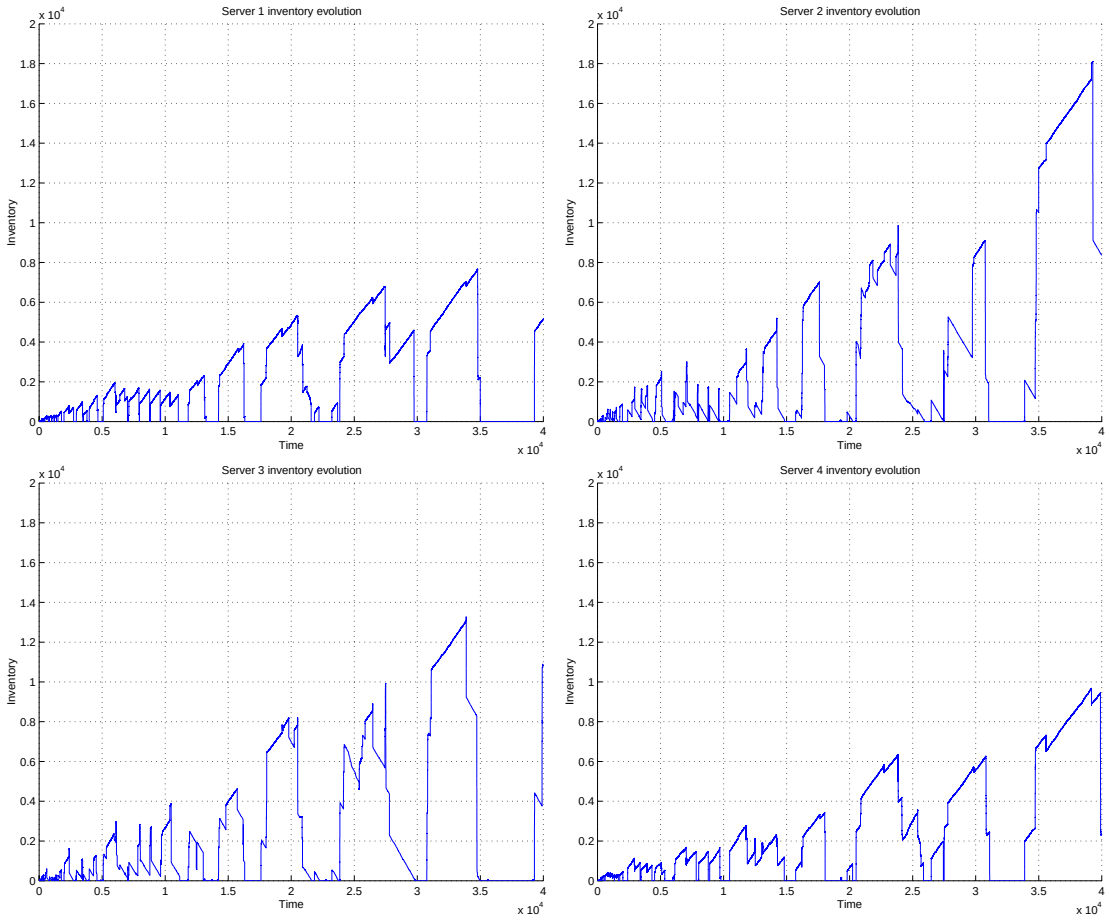

Figure B.2: Server inventory evolution for the *FIFO* scheduling policy.

Table B.2: *TW Controller* parameters.

Parameter	Value	Parameter	Value
T_k	0.010	f_6^{max}	0.996
α_k	100.0	f_7^{max}	0.996
f_1^{max}	0.996	f_8^{max}	0.002
f_2^{max}	0.002	f_9^{max}	0.002
f_3^{max}	0.002	f_{10}^{max}	0.002
f_4^{max}	0.002	f_{11}^{max}	0.002
f_5^{max}	0.002	f_{12}^{max}	0.996

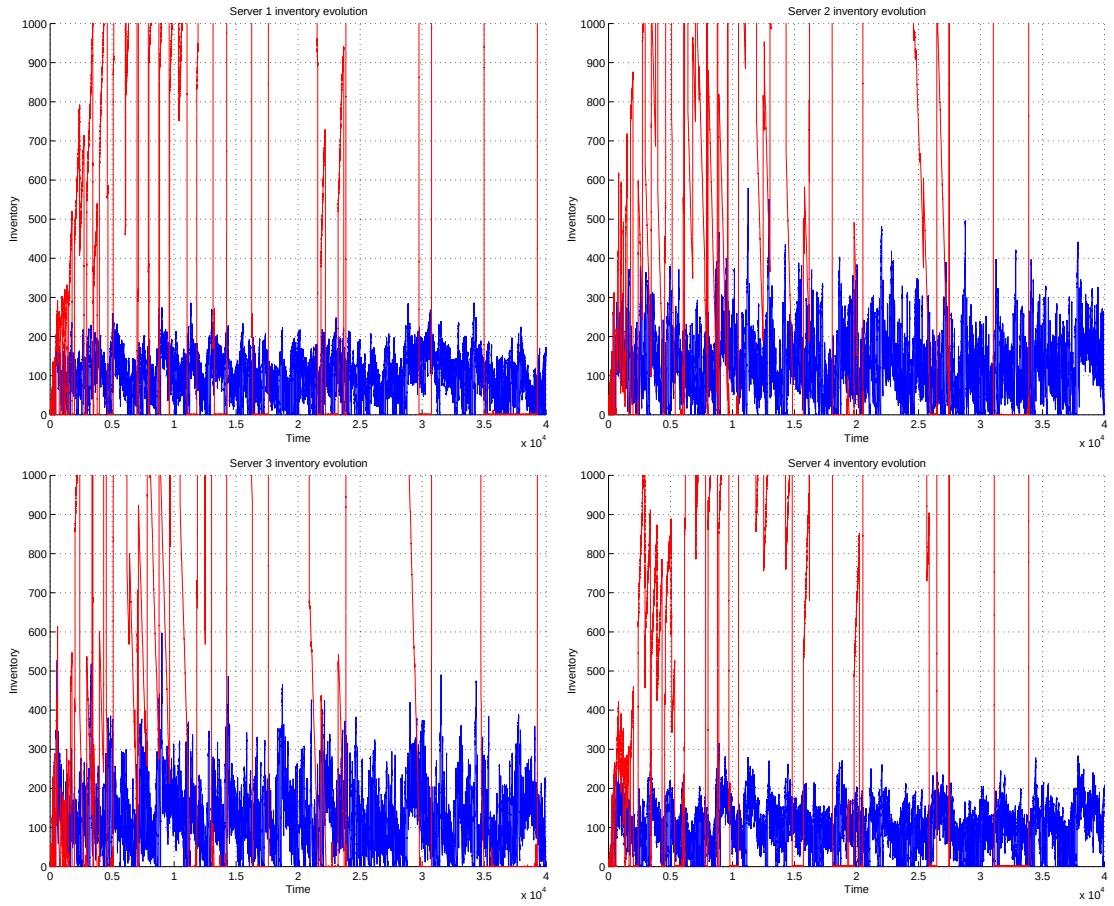

Figure B.3: Comparison of the server inventory evolution for the *FIFO* (red) and *FIFO + TW Controller* (blue) scheduling policies.

$$J(\overline{I}_{C_1}, ..., \overline{I}_{C_{12}}) = 3 \times \overline{I}_{C2} + 2 \times \overline{I}_{C4} + \overline{I}_{C1} + 3 \times \overline{I}_{C3} + 2 \times \overline{I}_{C5} + \overline{I}_{C7} + 3 \times \overline{I}_{C10} + 2 \times \overline{I}_{C8} + \overline{I}_{C6} + 3 \times \overline{I}_{C11} + 2 \times \overline{I}_{C9} + \overline{I}_{C12} \quad \text{(B.1)}$$

The results show once again that the unstable behaviour observed for the *FIFO* scheduling policy is due to the inability of the servers to use enough resources to process the customers, since the scheduling policy creates a starvation phenomena between the servers. The *TW Controller* is able to stabilize the system, adding in the process some *Active Idleness*, as presented in table B.4.

Table B.3: Comparison of the relevant statistics obtained from the simulations.

Statistic	Original	*TW Controller*	Statistic	Original	*TW Controller*
$\overline{I}_{S1}$	2227	107	f_1^∞	0.71312	0.89108
$\overline{I}_{S2}$	3628	118	f_2^∞	0.00099	0.00100
$\overline{I}_{S3}$	2779	119	f_3^∞	0.00097	0.00099
$\overline{I}_{S4}$	2593	106	f_4^∞	0.00089	0.00099
$\overline{I}_{C1}$	841	62	f_5^∞	0.00088	0.00996
$\overline{I}_{C2}$	691	23	f_6^∞	0.78214	0.90212
$\overline{I}_{C3}$	695	23	f_7^∞	0.71697	0.89793
$\overline{I}_{C4}$	1325	26	f_8^∞	0.00088	0.00099
$\overline{I}_{C5}$	1333	26	f_9^∞	0.00088	0.00099
$\overline{I}_{C6}$	970	66	f_{10}^∞	0.00097	0.00099
$\overline{I}_{C7}$	847	67	f_{11}^∞	0.00098	0.00100
$\overline{I}_{C8}$	967	26	f_{12}^∞	0.77581	0.88957
$\overline{I}_{C9}$	964	26	$J(\overline{I}_{C_1}, ..., \overline{I}_{C_{12}})$	—	734
$\overline{I}_{C10}$	823	22			
$\overline{I}_{C11}$	822	22			
$\overline{I}_{C12}$	947	61			

Table B.4: Comparison of the *Active Idle Time* used in each server.

Statistic	Original	*TW Controller*
Active idle time at server 1	0	0.0128
Active idle time at server 2	0	0.0218
Active idle time at server 3	0	0.0234
Active idle time at server 4	0	0.0123

Bibliography

[Adler, 1998] Adler, R. J. (1998). *A Practical Guide to Heavy Tails*. Chapman & Hall.

[Anantharam and Konstantopoulos, 1994] Anantharam, V. and Konstantopoulos, T. (1994). Burst reduction properties of the leaky bucket flow control scheme in ATM networks. *IEEE Transactions on Communications*, 42(12).

[Baskett et al., 1975] Baskett, F., Chandy, K. M., and Palacios, F. G. (1975). Open, closed, and mixed networks of queues with different classes of customers. *Journal of the Association for Computing Machinery*, 22(2):248–260.

[Bennet and Zhang, 1996] Bennet, J. C. and Zhang, H. (1996). WF^2Q: Worst-case fair weightef fair queueing. In *INFOCOM 96*. IEEE.

[Bertsekas and Gallager, 1987] Bertsekas, D. P. and Gallager, R. (1987). *Data Networks*. Prentice Hall.

[Bertsimas et al., 1996] Bertsimas, D., Gamarnik, D., and Tsitsiklis, J. N. (1996). Stability conditions for multiclass fluid queueing networks. *IEEE Transactions on Automatic Control*, 41(11):1618–1631.

[Bramson, 1994a] Bramson, M. (1994a). Instability of FIFO queueing networks. *The Annals of Applied Probability*, 4(2).

[Bramson, 1994b] Bramson, M. (1994b). Instability of FIFO queueing networks with quick service times. *The Annals of Applied Probability*, 4(3).

[Bramson, 1998] Bramson, M. (1998). Stability of two families of queueing networks and a discussion of fluid limits. *Queueing Systems*, 28.

[Bramson, 1999] Bramson, M. (1999). A stable queueing network with unstable fluid model. *The Annals of Aplied Probability*, 9(3):818–853.

[Cassandras and Lafortune, 1999] Cassandras, C. G. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

[Chen, 1995] Chen, H. (1995). Fluid aproximations and stability of multiclass queueing networks: Work-conserving disciplines. *The Annals of Applied Probability*, 5(3).

[Chen and Zhang, 2000] Chen, H. and Zhang, H. (2000). Stability of multiclass queueing networks under priority service disciplines. *Operations Research*, 48(1):26–37.

[Comeford, 2000] Comeford, R. (2000). Computing. *IEEE Spectrum*, pages 45–50. Special Issue on Technology 2000 Analysis and Forecast.

[Conway et al., 1967] Conway, R. E., Maxwell, W. L., and Miller, L. W. (1967). *Theory of Scheduling*. Addison-Wesley.

[Cruz, 1991a] Cruz, R. L. (1991a). A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1).

[Cruz, 1991b] Cruz, R. L. (1991b). A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1).

[Dai, 1995] Dai, J. G. (1995). On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *The Annals of Applied Probability*, 5(1).

[Dai et al., 1999] Dai, J. G., hasenbein, J. J., and Vate, J. H. V. (1999). Stability of a three-station fluid network. *Queueing Systems*, 33:293–325.

[Dai and Meyn, 1995] Dai, J. G. and Meyn, S. P. (1995). Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control*, 40(11).

[Dai and Weiss, 1996] Dai, J. G. and Weiss, G. (1996). Stability and instability of fluid models for reentrant lines. *Mathematics of Operations Research*, 21(1):115–134.

[Demers et al., 1990] Demers, A., Keshav, S., and Shenker, S. (1990). Analysis and simulation of a fair queueing algorithm. *Internetworking: Research and Experience*, 1(1).

[Dovrolis and Ramanathan, 1999] Dovrolis, C. and Ramanathan, P. (1999). A case for relative differentiated services and the proportional differentiation model. *IEEE Network*. September/October.

[Ferrari and Verma, 1990] Ferrari, D. and Verma, D. C. (1990). A scheme for real-time channel establishment in wide-area networks. *IEEE Journal on Selected Areas in Communications*, 8(3).

[French, 1982] French, S. (1982). *Sequencing and Scheduling: An introduction to the Mathematics of the Job-Shop*. Ellis Horwood Limited.

[Golestani, 1991] Golestani, S. (1991). A framing strategy for congestion management. *IEEE Journal on Selected Areas in Communications*, 9(7).

[Golestani, 1994] Golestani, S. (1994). A self clocked fair queueing scheme for broadband applications. In *INFOCOM 94*. IEEE.

[Graves, 1981] Graves, S. C. (1981). A review of production scheduling. *Operations Research*, 29. July-August.

[Humes Jr., 1994] Humes Jr., C. (1994). A regulator stabilization technique: Kumar-Seidman revisited. *IEEE Transactions on Automatic Control*, 39(1).

[Jackson, 1957] Jackson, J. R. (1957). Networks of waiting lines. *Operations Research*, 5:518–521.

[Kelly, 1979] Kelly, F. P. (1979). *Reversibility and Stochastic Networks*. John Wiley & Sons.

[Keshav, 1997] Keshav, S. (1997). *An Engineering Approach to Computer Networking*. Addison Wesley.

[Kleinrock, 1975] Kleinrock, L. (1975). *Queueing Systems, Volume I: Theory*. John Wiley & Sons.

[Kumar, 1993] Kumar, P. R. (1993). Re-entrant lines. *Queueing Systems: Theory and Applications*, 13:87–110.

[Kumar and Meyn, 1995] Kumar, P. R. and Meyn, S. P. (1995). Stable, distributed, and real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Transactions on Automatic Control*, 40(2).

[Law and Kelton, 1991] Law, A. M. and Kelton, W. D. (1991). *Simulation Modeling and Analysis*. McGraw-Hill. Second Edition.

[Lu et al., 1994] Lu, S. C. H., Ramaswamy, D., and Kumar, P. R. (1994). Efficient scheduling policies to reduce mean and variance of cycle-time in semiconductor manufacturing plants. *IEEE Transactions on Semiconductor Manufacturing*, 7(3).

[Lu and Kumar, 1991] Lu, S. H. and Kumar, P. R. (1991). Distributed scheduling policies based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, 36(12).

[Meyer, 1997] Meyer, B. (1997). *Object Oriented Software Construction.* Prentice-Hall. Second Edition.

[Nakata et al., 1999] Nakata, T., Matsui, K., Miyake, Y., and Nishioka, K. (1999). Dynamic bottleneck control in wide variety production factory. *IEEE Transactions on Semiconductor Manufacturing*, 12(3).

[Panwalkar and Iskander, 1977] Panwalkar, S. S. and Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25(1).

[Parekh and Gallager, 1993] Parekh, A. K. and Gallager, R. G. (1993). A generalized processor sharing approach to flow control in integrated services networks: The single node case. *IEEE Transactions on Networking*, 1(3).

[Parekh and Gallager, 1994] Parekh, A. K. and Gallager, R. G. (1994). A generalized processor sharing approach to flow control in integrated services networks: The multiple node case. *IEEE Transactions on Networking*, 2(2).

[Perkins and Kumar, 1989] Perkins, J. R. and Kumar, P. R. (1989). Stable, distributed, and real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Transactions on Automatic Control*, 34(2).

[Ross, 1987] Ross, S. M. (1987). *Introduction to Probability and Statistics for Engineers and Scientists.* John Wiley & Sons.

[Ross, 1996] Ross, S. M. (1996). *Stochastic Processes, Second Edition.* John Wiley & Sons.

[Rubinstein and Melamed, 1998] Rubinstein, R. Y. and Melamed, B. (1998). *Modern Simulation and Modeling*. John Wiley & Sons.

[Rybko and Stolyar, 1992] Rybko, A. N. and Stolyar, A. L. (1992). Ergodicity of stochastic processes describing the operations of open queuing networks. *Problemy Peredachi Informatsii*, 28:2–26.

[Seidman, 1994] Seidman, T. I. (1994). 'first come, first served' can be unstable! *IEEE Transactions on Automatic Control*, 39(10).

[Stiliadis and Varma, 1996] Stiliadis, D. and Varma, A. (1996). Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. In *SIGMETRICS 96*. ACM.

[Stroustrup, 1997] Stroustrup, B. (1997). *The C++ Programming Language*. Addison-Wesley.

[Tanenbaum, 1996] Tanenbaum, A. S. (1996). *Computer Networks*. Prentice-Hall.

[Turner, 1986] Turner, J. S. (1986). New directions in communications (or which way to the information age?). *IEEE Comunications Magazine*, 24(10):8–15.

[Verma et al., 1991] Verma, D., Zhang, H., and Ferrari, D. (1991). Guaranteeing delay jitter bounds in packet switching networks. In *TRICOMM 91*. Chapel Hill.

[Walrand, 1988] Walrand, J. (1988). *An Introduction to Queueing Networks*. Prentice Hall.

[Wein, 1988] Wein, L. M. (1988). Scheduling semiconductor wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing*, 1(3).

[Whitt, 1993] Whitt, W. (1993). Large fluctuations in a deterministic multiclass network of queues. *Management Science*, 39(8):1020–1028.

[Winograd and Kumar, 1996] Winograd, G. I. and Kumar, P. R. (1996). The FCFS service discipline: Stable network topologies, bounds traffic burstiness and delay, and control by regulators. *Mathematical and Computer Modelling*, 23(11/12):115–129.

[Zhang, 1995] Zhang, H. (1995). Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396.

[Zhang and Ferrari, 1993] Zhang, H. and Ferrari, D. (1993). Rate-controlled static priority queueing. In *Proc. INFOCOM 93*, pages 227–236. IEEE.

[Zhang, 1991] Zhang, L. (1991). Virtualclock: A new traffic control algorithm for packet-switched networks. *ACM Transactions on Computer Systems*, 9(2).