Active Idleness Scheduler with Simulated Annealing ⋆

Carlos F. Bispo *

* *Instituto de Sistemas e Robótica – Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal (e-mail: cfb@isr.ist.utl.pt).*

Abstract: In this paper we address the problem of optimizing the Time Window Controller (TW-Controller). This controller was first introduced in 2001 as a supervising mechanism for distributed scheduling of multiclass queuing networks with the objective of stabilizing those networks. It was then also shown that the TW-Controller possesses the ability to improve performance of stable networks. We revise the controller and present a series of formal results concerning its main properties and features. Then, we propose to use Simulated Annealing on a simulation-based optimization approach and present numerical results that demonstrate the controller's ability to improve performance over any given scheduling policy.

Keywords: Queuing Networks, Distributed Scheduling, Simulated Annealing, Stability

1. INTRODUCTION

In this work we address the issue of stability and performance improvement for multiclass, non-acyclic, and stochastic queuing networks. The stability problem has gained relevance when a series of published results showed that the Traffic Intensity Condition, although a necessary condition for stability of a network, is not sufficient. Examples of such results are Lu and Kumar (1991); Seidman (1994); Dai (1995), all operating with distributed and non-idling scheduling policies. These, and other similar examples, motivated a long series of work to address the issue of determining the stability of general queuing networks. Many authors focused on topological issues, by trying to identify features concerning the number of servers, the flow of customers, and individual processing times for which stability could be established for all, or a subset, of non-idling and distributed policies. Others focused on deriving provably stable scheduling policies for broad classes of queuing networks' topologies. This paper proposes that the concept of stability should be replaced by that of stabilizability. That is, given a queuing network that is unstable under some non-idling and distributed scheduling policy, what changes can be made to the policy in order to stabilize the network.

Such was the approach of the Time Window Controller (TW-Controller), Moreira (2001). This controller was presented as a supervisor of any given network, influencing the scheduling decisions of a given scheduling policy. Initially this controller was constructed with the purpose of provably stabilize a given network, Moreira and Bispo (2002). After that, the issue of performance improvement became a question to be taken into account, Moreira and Bispo (2003). It is in that line of development that this work was inspired. The main objective of this work is develop an optimizing block to be added to the TW-Controller, which operates in parallel with the controller, see Figure 1. This block uses Simulated Annealing and a discrete-event simulator to produce performance estimates. The Simulated Annealing approach has been proven to be very effective when applied to non convex optimization, for which gradient based algorithms may fail to produce good solutions, Cerný (1985); Marques et al. (1991); A.S.N. Silva and Alvarenga (2005); Kolahan et al. (2007). The utilization of this optimization algorithm will allow the evaluation of the TW-Controller's potential to improve performance over any given scheduling policy. In the revision of the TW-Controller we present a set of formal results which characterize the controller's main features and properties. The main property is the fact that the controller stabilizes a large set of queuing networks.

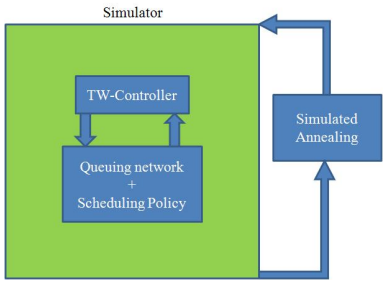

Fig. 1. Global architecture.

1.1 Stability of Queuing Networks

Queuing networks is one of the most used tools to model Discrete Event Dynamic System (DEDS), Cassandras and Lafortune (1999), useful to model realities like communication networks or manufacturing systems. Customers arrive to the network entering into some queue, or buffer, where they wait their turn to be processed. Each server chooses customers from a set of queues. After being processed, each customer may move to other buffer or leave the network.

⋆ This work was supported by the FCT (ISR/IST plurianual funding) through the PIDDAC Program funds.

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 and closed 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. 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 an idling scheduling policy, each server has the possibility of staying idle even when there are customers waiting. In a non-idling scheduling policy, each server will keep processing customers as long as there exists, at least, one customer waiting to be served.

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 their past history, they are processed in the same manner. 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.

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 network is to compute the invariant probability distribution for the number of customers, Kleinrock (1975); Kelly (1979); Walrand (1988). The stability of the network is then asserted by computing the average value of the invariant distribution which, if finite, implies stability.

The problem with this approach is that, outside the narrow class of networks possessing a product form solution for the invariant, it is very rare to obtain an explicit solution for the invariant, Baskett et al. (1975); Kelly (1979); Kumar and Kumar (1994). Combined with this, the queuing networks community conjectured that the stability problem was linked to the question of whether each server in the network has enough resources to process all customers that arrive to it. 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 1. (Traffic Intensity) Consider a multiclass queuing network composed of I single server stations. If class k is processed by server i, μ_k denotes the mean processing time of class k at server i. If class k arrives from the outside world, λ_k denotes the mean arrival rate of class k at server. Denoting by $c(i)$ the set of classes served at server i, the Traffic Intensity at server i is computed by the following expression:

$$\rho_i = \sum_{k=1, k \in c(i)}^{K} \lambda_k \mu_k \tag{1}$$

Note that class $k \in c(i)$ 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 $i = 1, 2, \ldots, I$ that $\rho_i < 1$.

In the early 90s several authors published a series of different networks which, combined with certain scheduling policies, were unstable even though all of them satisfied the Traffic Intensity Condition. One of the best known cases is the network proposed in Lu and Kumar (1991), which presented a relatively simple network with only two servers and four classes, combined with buffer priority scheduling policies as simple as the Last Buffer First Served (LBFS) and the First Buffer First Served (FBFS). In the specific case of this network, the combination that generates instability is to apply the policy LBFS to the first server and the FBFS the second server. This phenomenon was shown to also occur in more complex networks, as the one proposed in Seidman (1994), composed of four servers and twelve classes, which when combined with the First In First Out (FIFO) policy became unstable. An also unstable variant of Lu and Kumar (1991) was presented in Dai (1995) and Bramson (1994a,b) presents a demonstration of the instability generated by FIFO policies for some topologies. The initial conjecture that the Traffic Intensity Condition was sufficient to guarantee the stability of any network was proved to be wrong with these and other examples, even though it is sufficient for a large number of networks, as the Jackson network is an example, Jackson (1957).

The realization that the Traffic Intensity Condition is not sufficient to ensure stability for general networks when the policy is non idling originated a long series of work. Examples of stability determination as a function of the topology can be found in Kumar and Kumar (1994); Kumar and Meyn (1995); Dai and Meyn (1995); Kumar and Meyn (1996); Bertsimas et al. (1996); Bramson (1998, 1999); Chen and Zhang (2000); Lotker et al. (2004). We refer the reader more specifically to Chen and Zhang (2000) for a more detailed literature review on the subject. Other authors choose the path of proposing provably stable policies as in Perkins and Kumar (1989); Demers et al. (1990); Lu and Kumar (1991); Parekh and Gallager (1993); Bramson (1998); Maglaras (1999); Dai and Lin (2005, 2008).

One relevant exception is Kumar and Seidman (1990), where the focus was placed on stabilizability for the Clear-a-Fraction policies. That is, a supervisor mechanism was proposed to regulate the performance of the policies ensuring that stability is always attainable. Using the authors' own words "... there exists an "universal" safety mechanism, which can moreover be trivially implemented by a supervisor, in a distributed fashion, at the various machines. Essentially this mechanism consists simply of: 1) truncating all long production runs; and 2) maintaining a separate first-come first-serve (FCFS) priority queue Q_m for large buffers, at each machine m." The term universal is misleading as their results are established only for deterministic arrival and processing rates. To our knowledge there is no supervisor mechanism that provably stabilizes any policy for any network in a stochastic setting. Other examples in the area of regulation are Humes, Jr. (1994); Anantharam and Konstantopoulos (1994); Winograd and Kumar (1996)

Along a similar path is Moreira (2001); Moreira and Bispo (2002, 2003) which presented a supervisor mechanism that provably stabilizes any non idling distributed scheduling policy for a wide range of networks in a stochastic setting. We propose to review that approach complementing it with a general optimization procedure. We will be focusing on the MaxPressure policies of Dai and Lin (2005, 2008) because, besides ensuring stability for a wide class of networks, they are asymptotically optimal in heavy-traffic for quadratic queuing costs and ϵ-asymptotically optimal for linear costs. Also, their application does not require the knowledge of the external arrival rates, which is a very interesting property given that estimating those can be very difficult and because, in general, those rates are not stationary. Another interesting feature of the MaxPressure policies is the fact that they may exhibit an idling behavior in general. However, these policies are not distributed, as each server has to know the state of the overall system in order to chose which buffer to serve next. We will be using our regulator on these policies to demonstrate its performance improvement capabilities.

The paper is organized as follows. We start by introducing the TW-Controller, as in Moreira (2001), and present its formal guarantees, Section 2. In Section 3 we describe the Simulated Annealing implementation and how it interacts with the controller. A sample of numerical data is presented in Section 4, displaying the controller's ability to improve performance even over very good scheduling policies. The paper ends with the presentation of a set of main conclusions and directions for future research in Section 5.

2. THE TIME WINDOW CONTROLLER

This controller was presented for the first time in Moreira (2001). The idea behind its development was that it would work as a mechanism of supervision for any scheduling policy, rather than as a scheduling policy. Here we describe the main concepts behind the operation of this controller and how they are implemented. Latter in this section we will present the formal guarantees of stabilizability and performance improvement.

We construct the TW-Controller by presenting all of its constituents. The first of them is the Processing History for each class.

Definition 2. For each class $k = 1, 2, \ldots, K$, the Processing History of class k is defined as the function $H_k(t)$, such that $H_k(t) = 1$ if any customer of class k is being processed at the time instant t and $H_k(t) = 0$ if there is no client of class k being processed at that instant.

Function $H_k(t)$ describes the time intervals in which class k was being processed, since the beginning until the present time. The next constituent to be described is the Time Window.

Definition 3. For each class $k = 1, 2, \ldots, K$, the Time Window, also designated TW, associated to that class is defined as the finite interval of length T_k, starting at $t - T_k$ and ending at t, with t being the present time.

The Time Window of a class represents the amount of the class history that will be needed by the TW-Controller to make decisions concerning each class.

Definition 4. 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

$$f_k(t) = \beta_k \int_{t-T_k}^{t} e^{\alpha_k(\tau - t)} H_k(\tau) d\tau, \tag{2}$$

where $\alpha_k \in [0, \infty[$ is the Smoothing Parameter and β_k is the normalization parameter that ensures $f_k(t) = 1$ when $H_k(\tau) = 1$ for $\tau \in [t - T_k, t]$. This normalization parameter is given by

$$\beta_k = \frac{\alpha_k}{(1 - e^{\alpha_k T_k})}. \tag{3}$$

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. If $\alpha_k = 0$, the Time Fraction is the exact percentage of time during which, in the last T_k units of time, clients of class k were served. The objective of using a Smoothing Parameter bigger than zero is to weight the recent processing history differently than the older.

The next constituent of the controller is the concept of Blocked Class, which has the following definition.

Definition 5. A class k is said to be Blocked at time t if $f_k(t) > f_k^{\max}$, where $f_k^{\max}$, is the maximum time fraction allowed in terms of short/medium range for class k.

A Blocked class is a class that has exceeded its share, in terms of short/medium range, of workload that was imposed to the server that processes it. Grouping all the above constituents, we are able to define the Time Window Controller.

Definition 6. The Time Window Controller, or TW-Controller, can be applied to any multiclass, non-acyclic, queuing network, where each server is controlled by a non-idling scheduling policy. It is constructed through a K-tuple, $[f_1^{\max}, f_2^{\max}, \ldots, f_K^{\max}]$*, together with a value* $T_k = T$*, for* $k = 1, 2, \ldots, K$*, determining a single Time Window length and a value* $\alpha_k = 0$ *for* $k = 1, 2, \ldots, K$*, as a single Smoothing Parameter for all the classes. Each server chooses which class to serve based on its scheduling policy, being that clients on buffers from blocked classes are invisible in the decision process.*

Before going any further, it is important to describe the dynamics of the decision process when the TW-Controller is operating. While all the classes served by a given server are not blocked, all the scheduling decisions are made as if the controller does not exist. When one or more classes are blocked, the scheduling decisions may be affected, given that the group of eligible classes for processing is reduced. Let t^b be the time instant in which class k turns to the state of blocked, i.e., $f_k(t^b) > f_k^{\max}$. While that class is blocked, the value of its fraction will drop, because there are no more clients of that class being processed, until it is exactly equal to $f_k(t^u) = f_k^{\max}$ for some $t^u > t^b$, in which it becomes again an eligible class for processing. Over time, and depending on the scheduling policy used by each server, the different classes will turn, or not, to the blocked state and will always unblock in a finite amount of time.

It is important to take in account that it could happen that a given server does not start any job after concluding another, because all the waiting customers belong to classes that are blocked. Under these circumstances, a server will remain idle until the arrival of a customer of an unblocked class, or until one of the blocked classes that has waiting customers, reaches its unblocking instant. This type of idleness is termed Active Idleness, because it corresponds to a choice made by the controller. On the other hand, the idleness that happens due to the absolute lack of customers, in a context without controller, is called Passive Idleness. The instability that occurs in many queuing networks published in the past, stems from the fact that many buffers spend a lot of the time empty, inducing idleness in the server. The problem is that this type of idleness happens by accident and due to the scheduling policies being local. One of the ideas behind the development of this controller, is that is preferable to choose the idling instants in an active way, than to suffer the consequences of the Passive Idleness. The objective of the controller is to reduce all the network idleness by introducing some Active Idleness.

2.1 Formal guarantees

In this section we present the formal guarantee that will allow us to state that this controller stabilizes a broad class of networks. The Stabilizable Networks are now introduced.

Definition 7. Stabilizable Networks - Let Ω *be a multiclass, non-acyclic, queuing network, with* I *servers and* K *classes of customers, where each service station is controlled by the non-idling scheduling policy* Λ*. Furthermore, the network satisfies the Traffic Intensity Condition. Assume that there exists a maximum processing time for each client of each class,* Ψ_k*. That is, no client of class* k *has a processing time larger than* Ψ_k*. Each class is served at a single server and, after being processed, each customer of a given class always moves to the same buffer.*

The assumption of a maximum processing time is a formal limitation but has no serious consequences in the real world, because there is always some sort of bound on the service duration in real settings.

Lemma 1. Under the conditions for the construction of the controller explained in Definition 6, for the networks described in Definition 7 and given the conditions (a) to (d), presented below, the sum of the fractions of all the classes served by the same server, at any instant t_c*, at which any given service is concluded, verifies the following equation:*

$$\sum_{k=1, k\in c(i)}^{K} f_k(t_c) < 1, \qquad i = 1, 2, \ldots, I. \tag{4}$$

(a) $\sum_{k=1,k\in c(i)}^{K}(\lambda_k\mu_k)+\epsilon_i = 1, \quad i = 1, 2, \ldots, I$ and $\epsilon_i > 0$

(b) $\alpha_k = 0, \qquad k = 1, 2, \ldots, K$

(c) $f_k^{\max} = \lambda_k\mu_k + \epsilon_i/(2N_i), \qquad k \in c(i)$

where N_i *is the total number of classes processed by server* i*.*

(d) $T_k > \frac{\Psi_k}{\frac{\epsilon_i}{2N_i}}, \qquad k = 1, 2, \ldots, K$ and $k \in c(i)$.

This proof, as well as all others, is presented in the Appendix. Here we describe the meaning of the conditions stated above. The first condition, (a), describes the Traffic Intensity Condition. Basically it says that each server has a non-negative slack. Meaning that, the average work load in each server is below the existent capacity. Condition (b) states that there is no smoothing, the value is equal to zero. Condition (c) defines the maximum fraction value for each class. The value defined is slightly over the long term necessary value for each class, taking it from the existing slack. The last condition, (d), imposes the minimum value for the size of the Time Window for each class.

Lemma 1 establishes that all buffers have access to their fractions without their Time Fraction ever exceeding $\lambda_k\mu_k + \epsilon_i/N_i$, thus making the system uncoupled. What is sought with this is to allow an independent analysis of the different cycles in the network, reducing each analysis to a simple tandem and thereby also reducing the complexity of the problem.

Theorem 1. For a network under the conditions of definition 7, there exists an instance of the controller parameters $(T, \alpha, f^{\max})$ *that stabilizes the network, in the following sense. The long term arrival rate of each class to its buffer equals the external arrival rate of the buffer that "feeds" it. The stabilizing instance is defined in conditions (a) to (d) of Lemma 1.*

This result allows us to draw a set of interesting consequences and conclusions.

Corollary 1. Let $(T^*, \alpha^*, f^{*\max})$ *be the optimal instance of the controller's parameters for some given performance criterion. If all the buffers are visible by that cost function, this instance guarantees stability.*

Corollary 2. No matter what scheduling policy is chosen, there exists an instance of the controller's parameters that does not influence its decisions.

Corollary 3. If, for any cost function, the optimal Maximum Time Fraction is equal to one for every class, then the scheduling policy being used is optimal for the performance criterion applied.

At this time, we should discuss the meaning of the sum of all the Maximum Time Fractions being more than one. The Maximum Time Fractions defined in Lemma 1, have only the purpose of allowing us to construct an instance of the controller to establish Theorem 1. They serve the objective of ensuring the uncoupling of the classes served by each server. Normally, it does not make sense to break up the total processing capacity of each server in a way that does not allow taking advantage of its flexibility in serving more than one class. If at a given instant of time there are too few customers of a given class, the processing capacity that is allocated to it, is lost. As a result, the general solution for the value of the Maximum Time Fraction, would be allowing some kind of coupling between the different classes served on the same server to occur. This is done by letting the sum of these fractions to be above one. Because the optimum fractions are obtained after a search process on all the solution space, the resulting coupling would naturally be the optimal such coupling.

Corollary 4. For a given instance of the controller parameters $(T^*, \alpha^*, f^{*\max})$*, we can have* $\sum_{k=1, k\in c(i)}^{K} f_k^{\max} > 1$*, and still ensure the system to be stable.*

Corollary 5. Under the conditions of Definition 7 and Lemma 1, the Traffic Intensity Condition is necessary and sufficient for stabilizability.

2.2 Determination of the unblocking instant

When a class becomes blocked, it is necessary to determine at which instant of time it will become again available to being considered for processing. In the following we present two results that allow us to construct the algorithm that generates the unblocking instant. These two results will be presented without proof as they are simply instrumental to constructing the simulator, and their proofs do not add to such an objective.

Theorem 2. For a given class k that got blocked at the end of a service at time t_c*, when the waiting time until unblocking is done by only removing inactivity from its History at the far end of the Time Window, the waiting time until unblocking is calculated according to the following expression*

$$\Delta = -\frac{1}{\alpha} \ln\left(\frac{f_k^{\max}}{f_k(t_c)}\right) \tag{5}$$

Theorem 3. For a given class k that got blocked at the end of a service at time t_c*, when the waiting time until the unblocking is done by only removing service time from its History at the far end of the Time Window, the waiting time until unblocking is calculated according to the following expression*

$$\Delta = -\frac{1}{\alpha} \ln\left(\frac{\alpha f_k^{\max} + \beta e^{-\alpha T}}{\alpha f_k(t_c) + \beta e^{-\alpha T}}\right) \tag{6}$$

To compute the actual unblocking instant for any class that gets blocked we use these two results in the following algorithm.

Algorithm to determinate the unblocking time

S1: Identify if the Time Window shift will imply the exit of inactivity, or on the contrary implies the exit of activity. Make $z = 0$ and let t_c be the current time. If we are in the first situation go to S2, otherwise go to S3.

S2: Using Theorem 2, determine the necessary shift without removing any activity. If the obtained shift is such that it does not imply the exiting of any activity, then go to S4, adding to z the value now calculated. Otherwise, it is necessary update the Time Fraction of the class until the time instant where the first activity begins in the Time Window and advance to S3, adding to z the calculated value.

S3: In this step, the beginning of the Time Window is set on the time instant in which an activity starts, therefore, we use Theorem 3 to determinate the necessary shift. In case this shift does not exceed the duration of the activity we pretend to remove, the value of this shift is added to z and go to S4. Otherwise, the maximum shift possible, under these circumstances, is added to z and the Time Fraction of the class is updated. Go back to S2.

S4: It has already been determined, with success, the necessary shift for the Time Fraction to be equal to the Maximum Time Fraction. So, the buffer will get unblocked at the time instant $t_c + z$.

3. SIMULATED ANNEALING

Simulated Annealing, Kirkpatrick (1984), is a known optimization method used in functions with a behavior which limits the success of gradient-based algorithms. The procedures that this algorithm uses are easy to find in the literature. One crucial choice when setting up this type of approach concerns the neighborhood generation. Since we will be concerned with Maximum Time Fractions, our neighborhood generator will produce a neighbor simply by changing one of the parameters at a time. We tested alternative generators, but this achieved the most interesting results.

Another important topic, in terms of implementation, is to define how the temperature and step of the system will change along the simulation. This variation may occur in different ways, one of the most common and easy implementation being the geometric series of ratio r, or the harmonic series. In this case the geometric series was used to define the variation of temperature during the simulation, and the ratio is ϕ. Normally one defines this value between 0.70 and 0.99. For the step variation, the harmonic series was chosen. The reason relates to the fact

that, the sum of all the elements of a harmonic series is unbounded, which is relevant for line search algorithms.

Two stopping conditions were defined for the simulated annealing. One is the total number of simulations executed. The second, concerns the concept of phase change.

Definition 8. Phase Change. By keeping track of the cost reduction for every temperature decrease, a phase change is said to have occurred when there is a significant cost reduction for a small temperature reduction. We say that no phase change is occurring if the successive costs do not present significant reductions.

Consider $C(z_i)$ as the cost of configuration z_i. We say that a phase change happens when, for a certain neighbor configuration $z_(i+1)$, the corresponding cost satisfies the following condition:

$$C(z_{i+1}) \leq 0.95C(z_i). \quad (7)$$

The particular choices of values for the maximum number of simulations, maximum number of simulations between phase changes, and the value 0.95 used in (7) derive from several tests conducted.

4. RESULTS

We will present three different sets of numerical data that illustrate the capacity of the Time Window Controller, together with Simulated Annealing, to improve the performance of systems that are already stable by introducing a certain amount of Active Idleness. In the first two experiences, the performance measure consists of linear queuing costs, while the last experience will use pure quadratic costs on the queue lengths.

4.1 Lu and Kumar topology with linear costs

In this first case we will use the Lu and Kumar network, Lu and Kumar (1991). For this experience the simulation time was 20000 periods and the costs assumed to be linear.

Table 1. Network parameters.

Parameter	λ_1	μ_1	μ_2	μ_3	μ_4
Value	1.00	0.01	0.90	0.01	0.90

The scheduling policy applied was the Last Buffer First Served to all the servers. In Table 1 we present the arrival rate at buffer 1, λ_1, and the average processing time of each class, μ_i, and in Table 2 we present the initial parameters for the controller.

Table 2. Initial prameters of the controller.

$f^{\max}$	T	α
[1.0 1.0 1.0 1.0]	50	0.2

Figure2 presents the evolution of the average cost for the configurations that were accepted by the Simulated Annealing.

Just by looking at the figure we can understand why a gradient based algorithm would not be as effective as the Simulated Annealing, as very small changes in each of the Maximum Time Fractions may induce drastic variations in performance. Table 3 displays the cost of the original

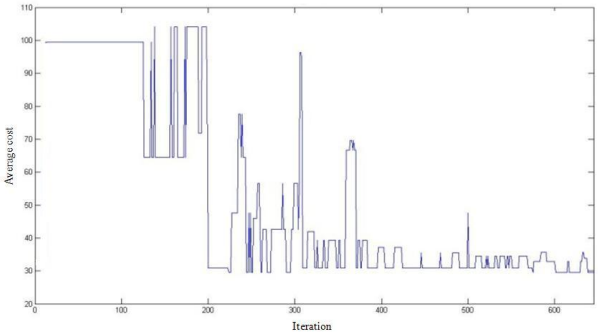

Fig. 2. Average cost of the configurations accepted by the Simulated Annealing.

Table 3. Average cost and improvement regarding the performance of the system without the controller

Configuration	Cost	Improvement
[1.0 1.0 1.0 1.0]	99.52	–
[0.883 1.0 0.998 0.982]	29.54	70.32%

policy and the best solution obtained with Simulated Annealing.

The improvement obtained with the controller is about 70% regarding the performance of the network when the controller is not used. This improvement was obtained by forcing one of the buffers to stay idle for a very small amount of time, therefore introducing active idleness on the network. The active idleness was obtained by blocking buffer 4 for about 164 periods, which represent 0.82% of the total simulation time. This is a very considerable improvement, even more if we consider that the Lu and Kumar topology is a low complexity network, which makes it harder to obtain substantial improvements than on bigger and more complex networks.

4.2 Seidman topology with linear costs

For this experience a more complex network was chosen, so it would be possible to apply a more efficient scheduling policy and yet we could see some improvement on the network's performance. The chosen network was the Seidman's network, Seidman (1994), along with the Maximum Pressure scheduling policy, Dai and Lin (2005).

Table 4. Network parameters.

Parameter	λ_2	λ_3	λ_{10}	λ_{11}	μ_1	μ_2
Value	1.0	1.0	1.0	1.0	0.9	1E-3
Parameter	μ_3	μ_4	μ_5	μ_6	μ_7	μ_8
Value	1E-3	1E-3	1E-3	0.9	0.9	1E-3
Parameter	μ_9	μ_{10}	μ_{11}	μ_{12}		
Value	1E-3	1E-3	1E-3	0.9		

This policy guarantees stability and assures an ϵ-asymptotic optimality, when traffic approaches 100% – Heavy Traffic. The parameters for this network are as in Tables 4 and 5.

Table 5. Initial parameters of the controller.

$f^{\max}$	T	α
[1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0]	50	0.2

Using the network configuration and the controller's parameters above described the result obtained for the evolution of network's average cost, is as shown in Figure 3.

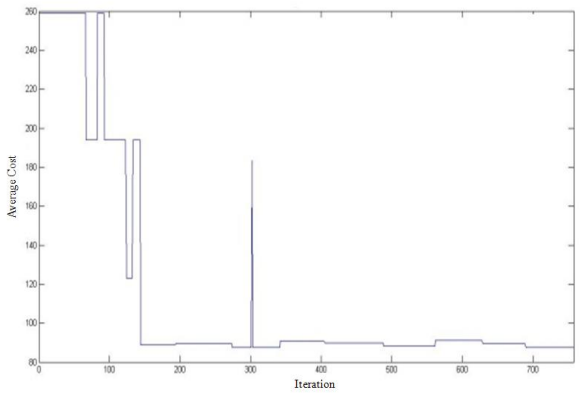

Fig. 3. Average cost of the configurations accepted by the Simulated Annealing.

A very significant improvement on the performance of the network is visible. Initial value was close to 260 and the best found is close to 90. Let us analyze it more precisely, as we look to Table 6, which presents the exact initial cost with no controller and the improvement brought by the controller.

Table 6. Average cost and improvement regarding the performance of the system without the controller

Configuration	Cost	Improvement
[1.0000 1 1 1 1 1 1 1 1 1 1 1]	258.9	–
[0.9012 1 1 1 1 1 1 1 1 1 1 1]	87.5	66.2%

The improvement produced by the controller was of 66%. This improvement was obtained by forcing buffer 1 to be idle for some periods of time. Specifically, in 378 periods of all simulation time. This idleness only represents about 7.5% of the total time and made the cost of the network drop by 66%, regarding the initial policy.

4.3 Seidman topology with quadratic costs

For Seidman's network we also tested with quadratic costs, and used the Max Pressure policy, which is asymptotically optimal with quadratic costs. The topology, the network parameters, and the duration of each simulation are all the same. The Simulated Annealing was used and the final result is as shown on Table 7

Table 7. Average cost and improvement regarding the performance of the system without the controller

Configuration	Cost	Improvement
[1.0000 1 1 1 1 1 1 1 1 1 1 1]	34405	–
[0.9012 1 1 1 1 1 1 1 1 1 1 1]	2952	91.42%

The influence of the controller in the original scheduling decisions has produced some Active Idleness in the system. More specifically, buffer 1 was forced to be idle 375 periods of all simulation time. This kind of inactivity represented 7,5% of the total simulation time. Forcing this buffer to be blocked during a small amount of the total time, has produced a huge improvement on the system's performance.

5. CONCLUSIONS

The main contributions and achievements of this work are the following.

- There was a formal "arrangement" or "cleanup" on the properties of the Time Window Controller.
- The Time Window Controller keeps improving the performance of the systems, even when they use more elaborated and effective scheduling policies, as the Maximum Pressure Policies.
- The Simulated Annealing revealed to be an effective tool to obtain a set of Maximum Time Fraction which improves the performance of any given network under any scheduling policy.

The first topic is related with the theorems, proofs, and mathematical expressions presented in this work, which represent an important contribution concerning the work published until now. In these contributions we can highlight the formal guarantee of stability of the controller and the expressions that enable to computation the unblocking time of each buffer, which are new.

The second topic constitutes the major contribution of this work, given the theoretical properties of the Maximum Pressure policies and the type of gains we were able to achieve. It is the first time the TW-Controller with Active Idleness is tested against extreme quality policies. This was only possible through Simulated Annealing, given it provides the needed generality to test different policies and optimize the parameters of our controller.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to a former student, Pedro M.P. Batista, who implemented the experimental set up being used for this paper.

REFERENCES

Anantharam, V. and Konstantopoulos, T. (1994). Burst redution properties of the leaky bucket flow control scheme in ATM networks. *IEEE Transactions on Communications*, 42(13).

A.S.N. Silva, R.S. and Alvarenga, G. (2005). Uma aplicação de simulated annealing para o problema de alocação de salas. *Biblioteca Digital Brasileira de Computação*.

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

Bertsimas, D., Gamarnik, D., and Tsitsiklis, J. (1996). Stability conditions for multiclass fluid queing networks. *IEEE Transactions on Automatic Control*, 41(11), 1618–1631.

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

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

Bramson, M. (1998). Stability of two families of queueing networks and a discussion of fluid limits. *Queuing Systems*, 28(1-3), 7–31.

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

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

Cerný, V. (1985). A thermodynamical approach to the travelling salesman problem. *Journal of Optimization Theory and Applications*, 45(1), 41–51.

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

Dai, J.G. and Lin, W. (2005). Maximum pressure policies in stochastic processing networks. *Operations Research*, 53(2).

Dai, J.G. and Lin, W. (2008). Asymptotic optimality of maximum pressure policies in stochastic processing networks. *The Annals of Applied Probability*, 18(6), 2239–2299.

Dai, J. (1995). On positive harris recurrence of multiclass queuing networks: A unified approach vie fluid limit models. *The Annals of Applied Probability*, 5(1).

Dai, J. and Meyn, S. (1995). Stability and convergence of moments for multiclass queuing-networks via fluid limit models. *IEEE Transactions on Automatic Control*, 40(11), 1889–1904.

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

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

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

Kelly, F. (1979). *Reversability and Stochastic Networks*. John Wiley & Sons.

Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34(5/6), 975–986.

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

Kolahan, F., Abolbashari, M., and Mohitzadeh, S. (2007). Simulated annealing application for structural optimization. *International Journal of Mechanical, Industrial and Aerospace Engineering*, 1(4).

Kumar, P. and Meyn, S. (1995). Stability of queuing-networks and scheduling policies. *IEEE Transactions on Automatic Control*, 40(2), 251–260.

Kumar, P. and Meyn, S. (1996). Duality and linear programs for stability and performance analysis of queuing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 41(1), 4–17.

Kumar, P. and Seidman, T. (1990). Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3).

Kumar, S. and Kumar, P. (1994). Performance bounds for queuing-networks and scheduling policies. *IEEE Transactions on Automatic Control*, 39(8), 1600–1611.

Lotker, Z., Patt-Shamir, B., and Rosen, A. (2004). New stability results for adversarial queuing. *SIAM Journal on Computing*, 33(2), 286–303.

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

Maglaras, C. (1999). Dynamic scheduling in multiclass queueing networks: Stability under discrete-review policies. *Queuing Systems*, 31(3-4), 171–206.

Marques, V., Bispo, C., and Sentieiro, J. (1991). A system for the compactation of two-dimensional irregular shapes by simulated annealing. In *IEEE International Conference on Industrial Electronics, Control and Instrumentation*. Kobe, Japan.

Moreira, J. (2001). *Distributed Scheduling with Active Idleness: A key to the stabilization of multiclass queuing networks*. Master's thesis, Instituto Superior Técnico, Technical University of Lisbon, Portugal.

Moreira, J. and Bispo, C. (2002). Stability or stabilizability? Seidman's FCFS example revisited. In *10th Mediterranean Conference on Control and Automation*. Lisbon, Portugal.

Moreira, J. and Bispo, C. (2003). Performance improvement through active idleness. In *11th Mediterranean Conference on Control and Automation*. Rhodes, Greece.

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

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

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

Ross, S. (1996). *Stochastic Processes*. John Wiley & Sons.

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

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

Winograd, G. and Kumar, P. (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.

Appendix A. PROOFS

All proofs were omitted because their inclusion would exceed the page limit of the submission. The 9 pages long version is available at http://users.isr.ist.utl.pt/~cfb/INCOM2012.pdf