Systems Engineering Group
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands
http://seweb.se.wtb.tue.nl/

SE-Report: Nr. 2006-04

Control of a reentrant manufacturing system with setup times: the Kumar-Seidman case

E. Lefeber and J.E. Rooda

ISSN: 1872-1567

SE Report: Nr. 2006-04
Eindhoven, October 2006
SE Reports are available via http://seweb.se.wtb.tue.nl/sereports

Abstract

In this paper we consider the control of a reentrant manufacturing system with setup times, as introduced by Kumar and Seidman. In most literature on control of a network of servers with setup times, first a policy is introduced and then the resulting network behavior is analyzed. In manufacturing systems the network typically is fixed and given a priori. Furthermore, optimal steady state behavior is desired. Therefore, this paper follows a different approach. First optimal steady state network behavior is determined, then a feedback controller is presented which makes the network converge towards this desired steady state behavior. The resulting controller is a non-distributed controller: each server needs global state information. For a manufacturing system this is not a problem, since global information typically is available. Finally it is shown that also a distributed controller (each server needs only local state information) can be used to achieve the same result.

1 Introduction

Reentrant manufacturing systems might show some unexpected behavior. In [1] it was shown by simulation that even when each server has enough capacity to serve all jobs, these networks can be unstable in the sense that the total number of jobs in the network explodes as time evolves. Whether this happens or not depends on the policy used to control the flows through the network. In [5] it was shown analytically that using a clearing policy (serve the queue you are currently serving until it is empty, then switch to another queue), certain networks become unstable, even deterministic systems with no setup times. In [8] several clearing policies have been introduced, the so-called Clear a Fraction (CAF) policies. It was shown that these policies are stable for a single server in isolation in a deterministic environment. Furthermore, it was shown that a CAF policy stabilizes a multi server system, provided the network is acyclic. A network is called acyclic if the servers can be ordered in such a way that jobs can only move from one server to a server higher in the ordering. A network is called non-acyclic if such an ordering is not possible. The example in [5] shows that there are non-acyclic networks that can not be stabilized by a CAF policy.

The main reason why CAF policies can fail for a non-acyclic network is because they spend too long on serving one type of job. This results in starvation of other servers and therefore a waste of their capacity. Due to this waste the effective capacity of these other servers is not sufficient anymore, resulting in an unstable system. This observation has led to the development of so-called buffer regulators [4, 9] or gated policies. The main idea is that each buffer contains a gate, so the buffer is split into two parts (before and after the gate). Instead of switching depending on the total buffer contents, switching is now determined based on the buffer contents after the gate. As a result, a server might now leave a buffer earlier, avoiding long periods of serving one type of job. It has been shown in [9] that under certain conditions on these regulators the (possibly non-acyclic) network is stabilized. Since non-acyclic networks are only unstable under certain conditions, applying buffer regulators is not always necessary. Needlessly applying buffer regulators results in a larger mean number of jobs in the network, which from a performance point of view is undesired. Furthermore, it is not known whether these policies result in optimal network behavior.

In [10, 11] a different approach has been developed. First the minimal period is determined during which the network is able to serve all jobs that arrive during that period. Given this minimal period, or any longer period, one can determine how long each server should serve each step. Next, a distributed controller is proposed where each server serves its buffers in a cyclic order until either the buffer becomes empty or the server has spend the time reserved for serving that step. If necessary the next setup is prolonged to make sure that the time reserved for serving a step is fully used. In [10, 11] it was shown that this policy guarantees that all trajectories of the controlled system are bounded and that for constant arrival rates the behavior of the network eventually becomes periodic, i.e. regular behavior is achieved. The policy introduced in [10, 11] has two disadvantages. First, it is not really a state feedback: the current mode of each server has been fixed a priori, independent of the current state. Second, if initially a large number of jobs is in the system, the resulting periodic steady state behavior also contains a large number of jobs in the system, as no attempt is made to reduce the number of jobs, which from a performance point of view is undesired.

The above mentioned references are only a few of the many papers that have been published in this area. However, most of the papers have one thing in common: first a policy (or a class of policies) is proposed, and then the resulting behavior of the network under this policy (these policies) is considered. Sometimes the system behavior is optimized over the class of considered policies. A strength of these results is that they can be applied to general networks. A drawback however is that it is usually unclear if the presented policies result in optimal behavior, or what to do to obtain prescribed or desired system behavior. In particular if a network is known a priori (and not subject to change), which typically is the case for manufacturing systems, one has the possibility of taking a global viewpoint and design a controller for the network which imposes optimal network performance.

Therefore, in this paper the approach we follow is exactly the other way around as in the

above mentioned papers. First we determine optimal system behavior. Next, this desired closed-loop behavior of the system is used as a starting point and then, based on the ideas presented in [6], a global policy is presented which establishes convergence to this optimal behavior.
Finally, we are able to improve the presented controller in such a way that it can be implemented in a distributed way. That is, we can implement a separate controller for each server where both controllers need only local information, i.e. no information about the state at the other server is needed. It turns out that the resulting local controllers are different from local controllers as so far proposed in literature.

2 The Kumar-Seidman case

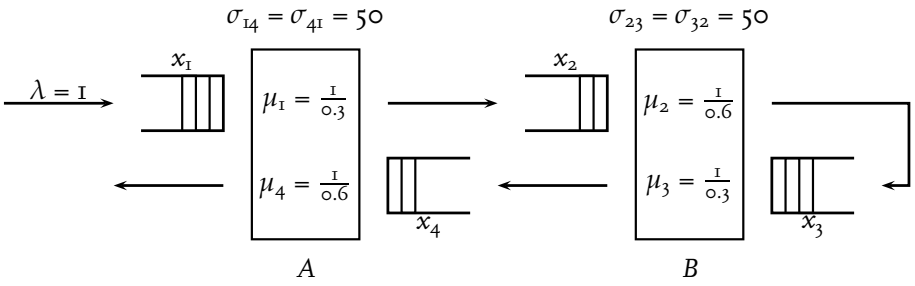

Figure 1: The system introduced in [5].

In [5] Kumar and Seidman presented the manufacturing system shown in Figure 1. A single job-type is considered which first visits machine A, then machine B, then machine B again, and finally machine A again. The successive buffers visited will be denoted by 1, 2, 3, and 4, respectively. The constant input rate λ into buffer 1 is 1 job/time-unit, while the maximal process rates required at the buffers are $\mu_1 = 1/0.3$, $\mu_2 = 1/0.6$, $\mu_3 = 1/0.3$, and $\mu_4 = 1/0.6$, respectively. Lastly, the times for setting up buffers 1 and 4 at machine A are $\sigma_{41} = 50$ and $\sigma_{14} = 50$, the times for setting up to buffers 2 and 3 at machine B are $\sigma_{32} = 50$ and $\sigma_{23} = 50$.
Even though for this system each machine has enough capacity, i.e. $(\lambda/\mu_1) + (\lambda/\mu_4) < 1$ and $(\lambda/\mu_2) + (\lambda/\mu_3) < 1$, it has been shown in [5] that since $(\lambda/\mu_2) + (\lambda/\mu_4) > 1$ and setup times are all positive, using a clearing policy for both machines results in an unstable system.
The state of this system is not only given by the buffer contents x_1, x_2, x_3, and x_4, but also by the remaining setup time at machine A, x_o^A, the remaining setup time at machine B, x_o^B, and the current mode $m = (m^A, m^B) \in \{(1,2), (1,3), (4,2), (4,3)\}$. We say that the system is in mode (1,2) when machine A is processing or setting up for step 1 and machine B is processing or setting up for step 2. Similar for the other modes.
The input of this system is given by rates $u_1 \le \mu_1$, $u_2 \le \mu_2$, $u_3 \le \mu_3$, and $u_4 \le \mu_4$, at which respectively buffers 1, 2, 3, and 4 are being served (a machine not necessarily has to serve at full rate), as well as the current activity for machine A, $u_o^A \in \{❶, ①, ❹, ④\}$, and for machine B, $u_o^B \in \{❷, ②, ❸, ③\}$. The activity ❶ denotes setting up for serving step 1, whereas ① denotes serving step 1. Similarly, the activities for steps 2, 3, and 4 can be distinguished.
The dynamics of this system is hybrid. On the one hand we have the discrete event dynamics

$$
\begin{aligned}
x_o^A &:= \sigma_{14}; & m^A &:= 4 & &\text{if } u_o^A = ❹ \text{ and } m^A = 1\\
x_o^A &:= \sigma_{41}; & m^A &:= 1 & &\text{if } u_o^A = ❶ \text{ and } m^A = 4\\
x_o^B &:= \sigma_{23}; & m^B &:= 3 & &\text{if } u_o^B = ❸ \text{ and } m^B = 2\\
x_o^B &:= \sigma_{32}; & m^B &:= 2 & &\text{if } u_o^B = ❷ \text{ and } m^B = 3.
\end{aligned}
$$

In words: if the system is currently in a mode, and according to the input the current activity becomes "set up to a different mode", both the remaining setup time and current mode change.
On the other hand we have the continuous dynamics

$$\dot{x}_\circ^A(t) = \begin{cases} -1 \text{ if } u_\circ^A \in \{❶, ❹\} \\ 0 \text{ if } u_\circ^A \in \{①, ④\} \end{cases} \qquad \dot{x}_\circ^B(t) = \begin{cases} -1 \text{ if } u_\circ^B \in \{❷, ❸\} \\ 0 \text{ if } u_\circ^B \in \{②, ③\} \end{cases}$$

$$\dot{x}_1(t) = \lambda - u_1(t) \qquad \dot{x}_2(t) = u_1(t) - u_2(t)$$

$$\dot{x}_4(t) = u_3(t) - u_4(t) \qquad \dot{x}_3(t) = u_2(t) - u_3(t).$$

Furthermore, at each time instant the input is subject to the constraints $u_1 \geq 0$, $u_2 \geq 0$, $u_3 \geq 0$, $u_4 \geq 0$, and

$u_\circ^A \in \{❶, ❹\}$	$u_1 = 0$	$u_4 = 0$	for $x_\circ^A > 0$
$u_\circ^A \in \{①, ❹\}$	$u_1 \leq \mu_1$	$u_4 = 0$	for $x_\circ^A = 0, x_1 > 0, m^A = 1$
$u_\circ^A \in \{①, ❹\}$	$u_1 \leq \lambda$	$u_4 = 0$	for $x_\circ^A = 0, x_1 = 0, m^A = 1$
$u_\circ^A \in \{❶, ④\}$	$u_1 = 0$	$u_4 \leq \mu_4$	for $x_\circ^A = 0, x_4 > 0, m^A = 4$
$u_\circ^A \in \{❶, ④\}$	$u_1 = 0$	$u_4 \leq \min(u_3, \mu_4)$	for $x_\circ^A = 0, x_4 = 0, m^A = 4$
$u_\circ^B \in \{❷, ❸\}$	$u_2 = 0$	$u_3 = 0$	for $x_\circ^B > 0$
$u_\circ^B \in \{②, ❸\}$	$u_2 \leq \mu_2$	$u_3 = 0$	for $x_\circ^B = 0, x_2 > 0, m^B = 2$
$u_\circ^B \in \{②, ❸\}$	$u_2 \leq \min(u_1, \mu_2)$	$u_3 = 0$	for $x_\circ^B = 0, x_2 = 0, m^B = 2$
$u_\circ^B \in \{❷, ③\}$	$u_2 = 0$	$u_3 \leq \mu_3$	for $x_\circ^B = 0, x_3 > 0, m^B = 3$
$u_\circ^B \in \{❷, ③\}$	$u_2 = 0$	$u_3 \leq u_2$	for $x_\circ^B = 0, x_3 = 0, m^B = 3$.

In words, these constraints say that in case the server is setting up, no jobs can be served. Furthermore, in case a setup has been completed, only the job type can be processed for which the server has been set up. This processing takes place at a rate which is at most μ_i if jobs of step i are available in the buffer and at the arrival rate if no jobs of type i are available in the buffer ($i \in \{1, 2, 3, 4\}$). Also, it is possible to either stay in the current mode, or to switch to the other mode. In particular it is possible during a setup to leave that setup and start a setup to the other type again. The latter setup is assumed to take the entire setup time.
Having defined the state, input, dynamics and constraints for the system, we can consider the problem of controlling this system, i.e. designing an input u which satisfies the constraints and achieves desired behavior. But before we can do so we first need to specify desired behavior.

3 Desired periodic behavior

As mentioned in the introduction, we do not want to start from a policy which works for a general network and analyze the resulting closed-loop behavior, but given this specific manufacturing system we want to start from desired system behavior and determine a feedback controller which makes the system converge towards this desired behavior. This implies that we first need to define desired periodic behavior. For manufacturing systems this would typically be behavior for which the mean amount of jobs in the system is minimal, since from Little's law we know that this results in the smallest mean flow time (the time a job spends in the system). More precisely, we would like to minimize

$$J = \frac{1}{T} \int_0^T x_1(t) + x_2(t) + x_3(t) + x_4(t) \, \mathrm{d}t$$

over the set of feasible periodic orbits, where T denotes the period of the periodic orbit under consideration.

A first observation is that the set of feasible periodic orbits is not empty. Note that, even though in [5] it has been shown that using a clearing policy for both machines renders the system of Section 2 unstable, it has been made clear in [10, 11] and other papers that this is not some system property, but due to the policy used. To make the latter more clear, consider machine A. Each job needs 0.3+0.6 = 0.9 time-units of processing. During a cycle of serving both step 1 and step 4 in total 50+50 = 100 time-units are lost due to setups. Therefore, during a cycle of 1000 time-units, the 1000 jobs that arrive can also be processed. The same holds for machine B (since the parameters of machine B are identical). These 1000 time-units is also the minimal cycle period as observed in [11], but a longer time period can be used as well. As presented in [2, 7], it is not necessary that a periodic orbit with the smallest mean amount of jobs also has the smallest period. Having a longer period in some cases reduces the mean amount of jobs in the system. At first glance this is counterintuitive. Having a longer period implies not serving jobs at the highest possible rate. Can this be efficient? It turns out that a trade-off needs to be made. Since a small period implies that on average less time is wasted on setups. So one either has to waste capacity by frequent switching, or by occasionally not serving at the highest possible rate.
Consider an optimal periodic orbit with period $T \geq 1000$. First consider machine A. For an optimal periodic orbit a machine should always produce at the highest possible rate [3]. Furthermore, during the period T, both step 1 and step 4 need to serve T jobs. This first observation determines the allowed activities, whereas the second observation determines their respective durations. We obtain that machine A loops through the following activities:

- ❶ for 50 time-units,
- ① at rate $u_1 = 1/0.3$ for $\frac{9}{35}T + 42\frac{6}{7}$ time-units,
- ① at rate $u_1 = 1$ for $\frac{1}{7}T - 142\frac{6}{7}$ time-units,
- ❹ for 50 time-units,
- ④ at rate $u_4 = 1/0.6$ for $\frac{3}{5}T$ time-units.

Note that machine A serves T jobs of step 1 within $\frac{2}{5}T - 100 < \frac{3}{5}T$ time-units. Machine B needs at least $\frac{3}{5}T > \frac{2}{5}T - 100$ time-units for serving step 2. Therefore, the possibility of machine B serving step 2 at rate $u_2 = 1$ can be disregarded. As a result, machine B needs to idle for a duration of $\frac{1}{10}T - 100$ time-units. Only for $T > 1000$ we have idling (not for $T = 1000$), and for the optimal periodic orbit this idling occurs when $x_2 = x_3 = 0$ and machine B has been set up for serving step 2.
Therefore, for the optimal periodic orbit, machine B loops through the following activities:

- ❷ for 50 time-units,
- ② at rate $u_2 = 0$ for $\frac{1}{10}T - 100$ time-units,
- ② at rate $u_2 = 1/0.6$ for $\frac{3}{5}T$ time-units,
- ❸ for 50 time-units,
- ③ at rate $u_3 = 1/0.3$ for $\frac{3}{10}T$ time-units.

Given these actions of both machines it is rather straightforward to determine for a given period T the optimal periodic behavior and the corresponding average number of jobs in the system.
For buffer 1 we have that for a duration of $\frac{3}{5}T + 100$ the buffer contents increase at a rate of 1 after which for a duration of $\frac{9}{35}T + 42\frac{6}{7}$ the buffer contents decrease to the initial level again.

Clearly, for the optimal periodic orbit the buffer contents go from 0 to $\frac{3}{5}T + 100$ and back to 0 again. As a result

$$\int_0^T x_1(t)\,\mathrm{d}t = \frac{9}{35}T^2 + \frac{600}{7}T + \frac{50000}{7}.$$

Similarly, for buffer 3 we have that for a duration of $\frac{3}{5}T$ the buffer contents increase from 0 to T. Next, the buffer contents remain at T for a duration of 50 time-units due to the setup from serving step 2 to serving step 3. Finally for a duration of $\frac{3}{10}T$ the buffer contents decrease from T to 0. This results in

$$\int_0^T x_3(t)\,\mathrm{d}t = \frac{9}{20}T^2 + 50T.$$

Determining the mean number of jobs in buffers 2 and 4 is a little bit more involved. First consider the case where $T > 1000$. In that case we know that buffer 2 is empty when we start serving step 2 at full rate, which coincides with starting to serve step 1 at full rate also. Then, for a duration of $\frac{9}{35}T + 42\frac{6}{7}$ the contents of buffer 2 increase from 0 to $\frac{3}{7}T + 71\frac{3}{7}$. Next, for a duration of $\frac{1}{7}T - 142\frac{6}{7}$ the contents of buffer 2 decrease from $\frac{3}{7}T + 71\frac{3}{7}$ to $\frac{1}{3}T + 166\frac{2}{3}$. Finally, for a duration of $\frac{1}{5}T + 100$ the contents of buffer 2 decrease from $\frac{1}{3}T + 166\frac{2}{3}$ to 0. As a result

$$\int_0^T x_2(t)\,\mathrm{d}t = \frac{1}{7}T^2 + \frac{100}{7}T - \frac{50000}{7}.$$

When machine A completes serving step 1, machine B still needs $\frac{1}{5}T + 100$ for clearing buffer 2. Therefore, when machine A is about to start serving step 4 it still takes $\frac{1}{5}T + 100$ time-units for jobs to arrive from machine B, which implies that buffer 4 should already contain $\frac{1}{3}T + 166\frac{2}{3}$ jobs. Therefore, for buffer 4 we have the following: For a duration of $\frac{2}{5}T$ buffer contents are equal to $\frac{1}{3}T + 166\frac{2}{3}$ jobs. Then for a duration of $\frac{1}{5}T + 100$ buffer contents decrease from $\frac{1}{3}T + 166\frac{2}{3}$ to 0. Then for a duration of $\frac{3}{10}T$ buffer contents increase from 0 to $\frac{1}{2}T$ and finally for a duration of $\frac{1}{10}T - 100$ buffer contents decrease from $\frac{1}{2}T$ to $\frac{1}{3}T + 166\frac{2}{3}$ jobs. As a result

$$\int_0^T x_4(t)\,\mathrm{d}t = \frac{17}{60}T^2 + \frac{200}{3}T.$$

The above derivation for the mean wip levels of buffers 2 and 4 was done under the assumption that buffer 2 is empty when machine B starts serving step 2 at full rate, which is the case for $T > 1000$. For $T = 1000$ it might be that buffer 2 contains jobs when machine B starts serving step 2 at full rate. In the latter case, for optimal behavior, machine A starts serving step 1 exactly at the time buffer 2 becomes empty. As a result, machine A also starts serving step 4 later, which leads to a reduction of the initial contents of buffer 4 by exactly the same amount as the initial contents of buffer 2 has increased. From this observation we can conclude that the mean number of jobs in the system therefore does not change.
To summarize, for a given period $T \geq 1000$ the periodic orbit which minimizes the mean number of jobs in the system results in:

$$J = \frac{1}{T}\int_0^T x_1(t) + x_2(t) + x_3(t) + x_4(t)\,\mathrm{d}t = \frac{17}{15}T + \frac{650}{3}.$$

Clearly, the minimal mean number of jobs is achieved for $T = 1000$.
As mentioned, several periodic orbits of period $T = 1000$ exist which minimize the mean amount of jobs in the system. Therefore, we also consider the mean amount of work in the system. The amount of work associated with a job is its total remaining process time. A job which is in buffer 1 needs 0.3 time-units processing at step 1, 0.6 time-units at step 2, 0.3 time-units at step 3 and 0.6 time-units at step 4. So the amount of work associated with a job in buffer 1 equals 1.8. Therefore, the amount of work in the system is given by $1.8x_1 + 1.5x_2 + 0.9x_3 + 0.6x_4$. As the mean amount of jobs in buffers 1 and 3 is the same, we

are interested in the periodic orbit for which the mean amount of jobs in buffer 2 is minimal. That is, the periodic orbit for which machine A and machine B start serving respectively steps 1 and 2 at the same time. The time-evolution of both the buffer contents and the amount of work for this orbit has been depicted in Figure 2.

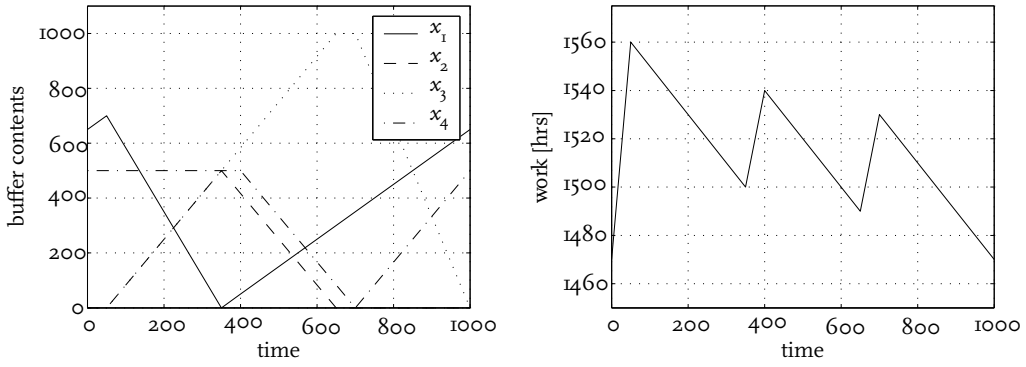

Figure 2: Evolution over time of both buffer contents and amount of work for the desired periodic behavior

- From $t = 0$ till $t = 350$ the system is in mode (1,2). From $t = 0$ till $t = 50$ both machines are setting up, from $t = 50$ till $t = 350$ both machines are serving at full rate. At the end of this mode $x_1 = 0$, $x_2 = 500$, $x_3 = 500$, and $x_4 = 500$.
- From $t = 350$ till $t = 650$ the system is in mode (4,2). From $t = 350$ till $t = 400$ machine A is setting up, from $t = 400$ till $t = 650$ machine A is serving at full rate. Machine B is serving at full rate all the time. At the end of this mode $x_1 = 300$, $x_2 = 0$, $x_3 = 1000$, and $x_4 = 83\frac{1}{3} = 50/0.6$.
- From $t = 650$ till $t = 1000$ the system is in mode (4,3). Machine A is serving at full rate all the time. From $t = 650$ till $t = 700$ machine B is setting up. From $t = 700$ till $t = 1000$ machine B is serving at full rate. At the end of this mode $x_1 = 650$, $x_2 = 0$, $x_3 = 0$, and $x_4 = 500$.

For this periodic orbit the mean amount of jobs in the system equals 1350 and the mean amount of work equals 1515 time-units. Notice that for the desired periodic orbit the system never is in mode (1,3). Furthermore, the largest amount of work in the system is reached at $t = 50$, in mode (1,2).

4 Non-distributed feedback

In the previous section we determined optimal periodic behavior for the manufacturing system introduced in Section 2. For the periodic orbit as depicted in Figure 2, both the mean amount of jobs and the mean amount of work in the system is minimal. In particular this implies that the mean flow time, i.e. the time a job spends in the system, is minimal. Given this desired periodic behavior, we next want to have a feedback controller which makes the manufacturing system introduced in Section 2 converge towards this desired behavior from any initial condition. Given that we are in a manufacturing setting, global information is available, which in particular implies we do not have to restrict ourselves to distributed controllers. A non-distributed controller can be implemented relatively easily.

In [6] we introduced for arbitrary networks of switching servers with setup times an approach for deriving a feedback controller from a given desired periodic orbit. This approach guarantees convergence similar to the desired periodic orbit. However, it might be that some buffers always contain a fixed number of additional number of jobs, compared to the desired periodic orbit. Something similar holds for the controllers presented in [10, 11], but the controller that follows from applying the ideas in [6] results in smaller additional amount of jobs in the system.

Before presenting the controller, we first consider the desired behavior as depicted in Figure 2. The system cyclically visits the modes (1,2), (4,2), and (4,3). In particular the system does not visit mode (1,3). Similar to the controller presented in [6] we therefore let the feedback be such that the system cyclically visits the modes (1,2), (4,2), and (4,3). If the system happens to be initially in mode (1,3), we switch to mode (1,2), since the largest amount of work in the system is reached in mode (1,2), cf. [6].

Next, we need to determine when to leave each mode. Consider mode (1,2) of the desired periodic orbit. In this mode, only contents of buffer 1 decreases and reaches the value 0. Buffers 2 and 3 both increase. Since machine B serves from buffer 2 to buffer 3, these buffers can be considered together. Both buffers increase until $x_2 + x_3 = 1000$. Therefore, the feedback leaves mode (1,2) when $x_1 = 0$ and $x_2 + x_3 \geq 1000$. Notice that it might happen that machine A needs to continue serving buffer 1 at the arrival rate whenever $x_1 = 0$ and $x_2 + x_3 < 1000$. Similarly, when $x_1 = x_2 = 0$ and $x_3 < 1000$, also machine B might need to continue serving buffer 2 at the arrival rate.

In mode (4,2) the contents of both buffer 2 and buffer 4 decreases, until respectively $x_2 = 0$ and $x_4 = 83\frac{1}{3}$. The contents of both buffer 1 and buffer 3 increases, until respectively $x_1 = 300$ and $x_3 = 1000$. Notice that by staying in mode (4,2) it is not possible for x_3 to become arbitrarily large. Therefore, the feedback leaves mode (4,2) when $x_2 = 0$, $x_4 \leq 83\frac{1}{3}$, and $x_1 \geq 300$. All three conditions need to be met. Therefore, it might be required for machine A or machine B to idle.

Similarly, the feedback leaves mode (4,3) when both $x_3 = 0$ and $x_1 \geq 650$. Again, it might be required for machine A or machine B to idle.

Notice that the condition on x_1 for leaving mode (4,2) and mode (4,3) are automatically fulfilled once the system has been in mode (1,2). Therefore, we drop these conditions in the feedback.

The above can be summarized as follows:

Proposition 1. *Consider the system as depicted in Figure 1 in closed-loop with the following feedback:*

- *If initially in mode (1,3), switch to mode (1,2).*
- *If in mode (1,2), stay in this mode until both $x_1 = 0$ and $x_2 + x_3 \geq 1000$. Then switch to mode (4,2). Both machines serve at the highest possible rate (which might be the arrival rate).*
- *If in mode (4,2), stay in this mode until both $x_2 = 0$ and $x_4 \leq 83\frac{1}{3}$. Then switch to mode (4,3). Both machines serve at the highest possible rate (which might be 0).*
- *If in mode (4,3), stay in this mode until $x_3 = 0$. Then switch to mode (1,2). Both machines serve at the highest possible rate (which might be 0).*

Then the resulting closed-loop system converges towards the behavior as depicted in Figure 2.

Proof. Let $t_{12}^{(k)}$ denote the time at which mode (1,2) is entered for the k^{th} time ($k \geq 1$), and let $(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, x_4^{(k)})$ denote the buffer contents at $t_{12}^{(k)}$ for buffers 1, 2, 3, and 4, respectively. Let $t_{42}^{(k)}$ and $t_{43}^{(k)}$ denote the moment at which mode (4,2) respectively mode (4,3) is entered, and define the durations of these modes: $\tau_{12}^{(k)} = t_{42}^{(k)} - t_{12}^{(k)}$, $\tau_{42}^{(k)} = t_{43}^{(k)} - t_{42}^{(k)}$, and $\tau_{43}^{(k)} = t_{12}^{(k+1)} - t_{43}^{(k)}$ respectively.

It needs to be shown that

$$\lim_{k\to\infty}(x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, x_4^{(k)}) = (650, 0, 0, 500). \tag{1}$$

A first observation is that at $t = t_{43}^{(k)}$ we have $x_2 = 0$ and $x_3 \geq 1000$, since in mode (1,2) at least 1000 jobs are processed by machine A and in mode (4,2) buffer 2 is emptied. As a result, at $t = t_{12}^{(k+1)}$ we have $x_2^{(k+1)} = 0$, $x_3^{(k+1)} = 0$, $x_4^{(k+1)} \geq 500$.

A second observation is that $\tau_{42}^{(k+1)} \geq 300$ (since $x_4^{(k+1)} \geq 500$), and $\tau_{43}^{(k)} \geq 350$ (since machine B needs to process at least 1000 jobs and requires a setup).

From these observations it follows that without loss of generality we can assume $x_1^{(k)} \geq 650$, $x_2^{(k)} = 0$, $x_3^{(k)} = 0$, $x_4^{(k)} \geq 500$ by considering $k \geq 2$. Under these assumptions we would like to determine $x_1^{(k+1)}$ and $x_4^{(k+1)}$.

During mode (1,2), first both machines need to be set up, which takes 50 time-units. At $t_{12}^{(k)} + 50$ buffer 1 contains $x_1^{(k)} + 50$ jobs. If machine A serves at full rate, x_1 effectively reduces at a rate of $(1/0.3) - 1 = 7/3$ jobs per time-unit. Therefore, clearing buffer 1 takes $\frac{3}{7}(x_1^{(k)} + 50)$, during which $\frac{10}{7}(x_1^{(k)} + 50)$ jobs are being processed by machine A. Notice that since $x_1^{(k)} \geq 650$ also 1000 jobs have been processed, so $\tau_{12}^{(k)} = 50 + \frac{3}{7}(x_1^{(k)} + 50)$. Also, at $t_{42}^{(k)} = t_{12}^{(k)} + \tau_{12}^{(k)}$ we have $x_2 = x_3 = \frac{5}{7}(x_1^{(k)} + 50)$.

Next, from the condition on x_2 we obtain $\tau_{42}^{(k)} \geq \frac{3}{7}(x_1^{(k)} + 50)$, and from the condition on x_4 we obtain $\tau_{42}^{(k)} \geq \frac{3}{5}x_4^{(k)}$, so $\tau_{42}^{(k)} = \max\left(\frac{3}{7}(x_1^{(k)} + 50), \frac{3}{5}x_4^{(k)}\right)$.

At $t_{43} = t_{42}^{(k)} + \tau_{42}^{(k)}$ we have $x_3 = \frac{10}{7}(x_1^{(k)} + 50)$, so $\tau_{43} = 50 + \frac{3}{7}(x_1^{(k)} + 50)$.

Since $x_4 = 0$ at $t_{43} + 50$ we get $x_4^{(k+1)} = \frac{5}{7}(x_1^{(k)} + 50)$. Furthermore, $x_1^{(k+1)} = \tau_{42}^{(k)} + \tau_{43}^{(k)} = \max\left(\frac{3}{7}(x_1^{(k)} + 50), \frac{3}{5}x_4^{(k)}\right) + 50 + \frac{3}{7}(x_1^{(k)} + 50)$.

To summarize, we have for $k \geq 2$ that

$$x_1^{(k+1)} = \max\left(\frac{3}{7}(x_1^{(k)} + 50), \frac{3}{5}x_4^{(k)}\right) + 50 + \frac{3}{7}(x_1^{(k)} + 50) \tag{2a}$$

$$x_2^{(k+1)} = 0 \tag{2b}$$

$$x_3^{(k+1)} = 0 \tag{2c}$$

$$x_4^{(k+1)} = \frac{5}{7}(x_1^{(k)} + 50) \tag{2d}$$

Using this map we need to show that (1) holds.

To simplify analysis of (2), define $y_1^{(k)} = \frac{1}{7}(x_1^{(k)} + 50) - 100$ and $y_4^{(k)} = \frac{1}{5}x_4^{(k)} - 100$, or $x_1^{(k)} = 7y_1^{(k)} + 650$, and $x_4^{(k)} = 5y_4^{(k)} + 500$. Then using (2) we obtain for $k \geq 2$:

$$y_1^{(k+1)} = \frac{3}{7}\max(y_1^{(k)}, y_4^{(k)}) + \frac{3}{7}y_1^{(k)} \qquad y_1^B \geq 0 \tag{3a}$$

$$y_4^{(k+1)} = y_1^{(k)} \qquad y_4^B \geq 0. \tag{3b}$$

From (3) we have

$$0 \leq y_1^{(k+1)} \leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}).$$

Also

$$0 \leq y_1^{k+2} \leq \frac{6}{7}\max\left(\frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}), y_1^{(k)}\right) \leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}).$$

Therefore,

$$0 \leq \max(y_1^{k+2}, y_4^{k+2}) \leq \frac{6}{7}\max(y_1^{(k)}, y_4^{(k)}),$$

from which we can conclude that

$$\lim_{k\to\infty} y_1^{(k)} = \lim_{k\to\infty} y_4^{(k)} = 0,$$

which shows that (1) holds. □

Remark 2. Proposition 1 only claims convergence towards the steady state cycle depicted in Figure 2. No claims are made concerning optimal transient behavior.

5 Distributed controller implementation

The controller derived in the previous section is a non-distributed controller. Machine A needs information about the state at machine B to determine what to do, and machine B requires information about the state at machine A. Since we consider a manufacturing problem, global information is available and a non-distributed controller can be implemented in a rather straightforward way.
Nevertheless, in this case the controller can be implemented in a distributed way. That is, such that machine A does not require information about the state at machine B and machine B does not require information about the state at machine A. This becomes clear from the proof of Proposition 1.
Notice that we know from the proof of Proposition 1 that for $k \geq 2$ when machine B switches from serving step 2 to serving step 3, $x_2 = 0$ and $x_2 + x_3 \geq 1000$. That is, buffer 2 needs to be empty, and machine B should have served at least 1000 jobs (for $k \geq 2$). Furthermore, notice that before switching from mode (4,2) to mode (4,3) it might happen that machine B needs to idle (in case x_4 is still too large). However, instead of first idling and then serving step 3, machine B can also first switch and serve step 3 and then idle for the same duration. As long as machine A stops serving step 4 at the same time as in the feedback of Proposition 1, the mapping (2) still holds. So as long as the behavior of machine A is still according to that specified by the feedback of Proposition 1 we can implement the following controller for machine B:

- Serve step 2 at the highest possible rate (which might be at the arrival rate or even idling) until both $x_2 = 0$ and at least 1000 jobs have been served. Then switch to step 3.
- Serve step 3 at maximal rate until $x_3 = 0$. Then switch to step 2.

It remains to determine a controller for machine A. As mentioned above, this controller needs to make sure that, after a finite transient, the overall system behavior still satisfies mapping (2). Notice that in this mapping only x_1 and x_4 play a role. This enables us to come up with a controller for machine A. From the proof of Proposition 1 we know that, for $k \geq 2$, during mode (1,2) machine A serves $\bar{x}_1^{(k)} = \frac{10}{7}(x_1^{(k)} + 50)$ jobs. From (2d) we know that after machine A has served step 4, machine B is empty and $\frac{1}{2}\bar{x}_1^{(k)}$ jobs remain in buffer 4. At the time machine A starts serving step 4, buffer 4 contains $x_4^{(k)}$ jobs. From this it follows that machine A has served $x_4^{(k)} + \bar{x}_1^{(k)} - \frac{1}{2}\bar{x}_1^{(k)} = x_4^{(k)} + \frac{1}{2}\bar{x}_1^{(k)}$ jobs. Therefore, we can implement the following controller for machine A:

- Serve step 1 at the highest possible rate (which might be at the arrival rate) until both $x_1 = 0$ and at least 1000 jobs have been served. Then switch to step 4. Let $\bar{x}_1$ denote the number of jobs served.
- Let $\bar{x}_4$ denote the number of jobs in buffer 4. Serve $\bar{x}_4 + \frac{1}{2}\bar{x}_1$ jobs from buffer 4. Then switch to step 1.

The above can be summarized in the following

Proposition 3. *Consider the system as depicted in Figure 1 in closed-loop with the following feedback:*

- *Controller for machine A:*
  - *If serving step 1, continue until both $x_1 = 0$ and at least 1000 jobs have been served. Then switch to step 4. Let $\bar{x}_1$ be the number of jobs served during this mode.*

– *Let $\bar{x}_4$ denote the number of jobs in buffer 4 when the setup to serving step 4 has completed. If serving step 4, continue until $\bar{x}_4 + \frac{1}{2}\bar{x}_1$ jobs have been served. Then switch to step 1.*

- *Controller for machine B:*
  - *Serve step 2 at the highest possible rate (which might be at the arrival rate or even idling) until both $x_2 = 0$ and at least 1000 jobs have been served. Then switch to step 3.*
  - *Serve step 3 at maximal rate until $x_3 = 0$. Then switch to step 2.*

Then the resulting closed-loop system converges towards the behavior as depicted in Figure 2.

Proof. Notice that the cycle for machine B always takes at least 1000 time-units, since at least 1000 jobs need to be served by each step during the cycle. First, assume that the cycle at machine B repeatedly takes exactly 1000 time-units. In that case machine A also takes a period of 1000 time-units. Furthermore, serving step 1 takes 300 time-units and serving step 4 takes 600 time-units, which implies that buffer 4 contains 500 jobs when machine A starts serving step 4. This implies that the system operates according to the desired periodic orbit.
As soon as the cycle for machine B takes strictly more than 1000 time-units, machine B will synchronize with machine A, cf. Section 3, i.e after a finite transient, $x_2 = x_3 = 0$ and machine B is waiting for jobs to arrive from machine A. After this transient, machine A guarantees that (2) holds again, which guarantees (1). □

6 Simulation experiments

To support our claims we performed several simulations in which we compared the distributed controller presented in the previous section with the controller proposed in [11]. We chose the parameters of the latter controller according to the desired periodic orbit, i.e. such that this controller is able to stay on the desired periodic orbit once the system is on it.
In the first simulation we started with an empty system, i.e. $x_1 = x_2 = x_3 = x_4 = 0$, where we initiated the controllers for machine A and B in respectively "serving step 1" and "serving step 2". The resulting responses of the controlled systems are given in Figure 3 for both controllers. In this figure we see that the controller proposed in [11] achieves regular behavior, as claimed in [11]. Furthermore, we see that buffer 1 is never emptied, resulting in a larger mean number of jobs in the system. For the controller of Proposition 3 we obtain convergence towards the desired periodic orbit. Notice that the total number of jobs oscillates between 1150 at the end of mode (4,3) and 1550 at the time the system starts serving step 4, as should be the case.
In the second simulation we do not start with an empty system, but with initially 1000 jobs in each buffer, i.e. $x_1 = x_2 = x_3 = x_4 = 1000$, where the controllers were initiated as in the first simulation. The resulting responses of the controlled systems are given in Figure 4 for both controllers. In this figure we clearly see the focus on regular behavior for the controller introduced in [11]. None of the buffers is cleared and the mean amount of jobs in the system is not reduced. However, the controller presented in Proposition 3 is able to reduce the number of jobs in the system and makes the system converge again to the desired optimal behavior.
Our third simulation experiment is a discrete event simulation. First of all, the hybrid fluid model as introduced in Section 2 has been replaced by its discrete event counterpart. Second, all process times and setup times are made stochastic by drawing them from independent exponential distributions. That is, process times for step 1 are drawn from an exponential distribution with mean 0.3, process times for step 2 are drawn from an exponential distribution with mean 0.6, setup times for switching from step 1 to step 4 are drawn from an exponential distribution with mean 50, etc. Again we initiated the system in $x_1 = x_2 = x_3 = x_4 = 1000$, but this time assuming that the setups to serving step 1 and step 2 respectively have already been completed. Implementing the controller of Proposition 3 in this stochastic setting is

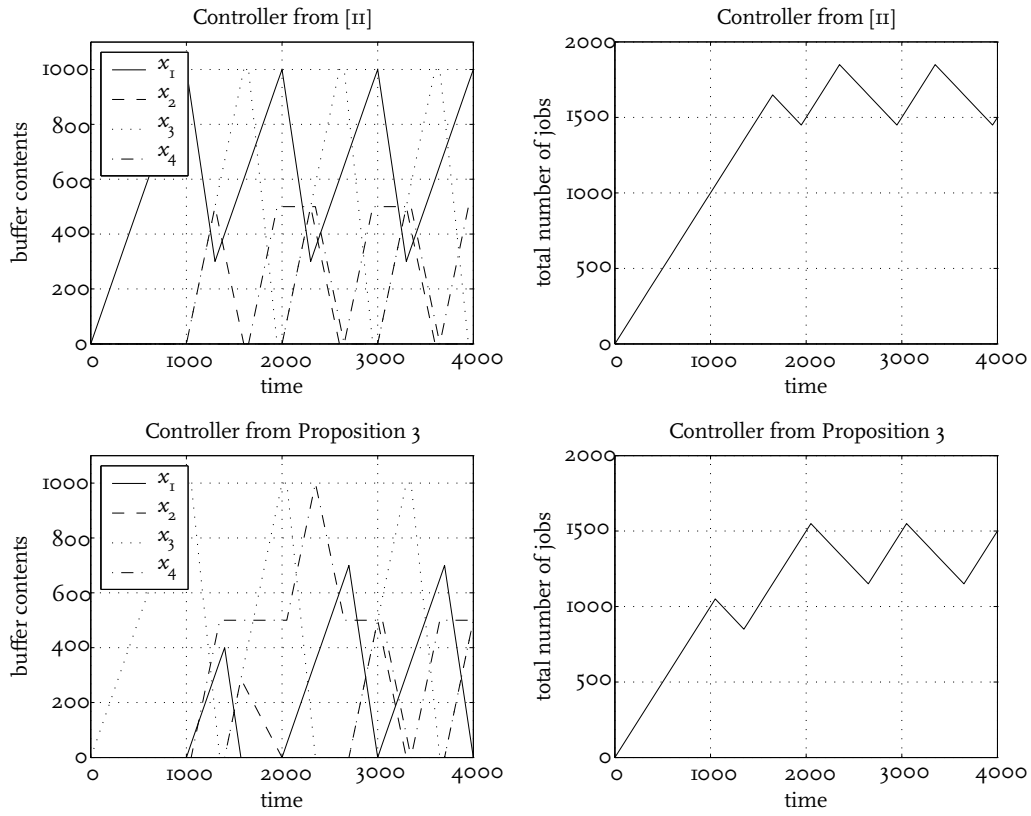

Figure 3: The buffer contents and total number of jobs for a deterministic system initiated in $(x_1, x_2, x_3, x_4) = (0, 0, 0, 0)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

rather straightforward. The controller introduced in [11], however, requires predetermined maximum durations. Therefore, we considered a preemptive resume policy. That is, in case the machine has not yet completed a job but it is time to switch, the machine stops serving the job and completes this service as soon as the machine has switched back. The resulting responses of the controlled systems in this stochastic discrete event environment are given in Figure 5 for both controllers. By controlling the seeds of the random generator we made sure to have a fair comparison between both controllers. The successive inter arrival times are the same for both experiments, as are the successive process times at each machine and the successive setup times. We looking at the results depicted in Figure 5 we see that similar remarks as for the second simulation can be made. Due to the stochasticity we see some varying behavior over time. The controller introduced in [11] keeps the number of jobs in the system between 4000 and 6000, whereas the controller of Proposition 3 keeps the number of jobs in the system between 1500 and 2500.

7 Conclusions

In this paper we considered the reentrant system introduced by Kumar and Seidman. Since we are interested in optimal network behavior, instead of proposing a control policy for this reentrant network and analyzing the resulting network behavior, we first determined the optimal behavior for this network and then determined a policy which guarantees convergence

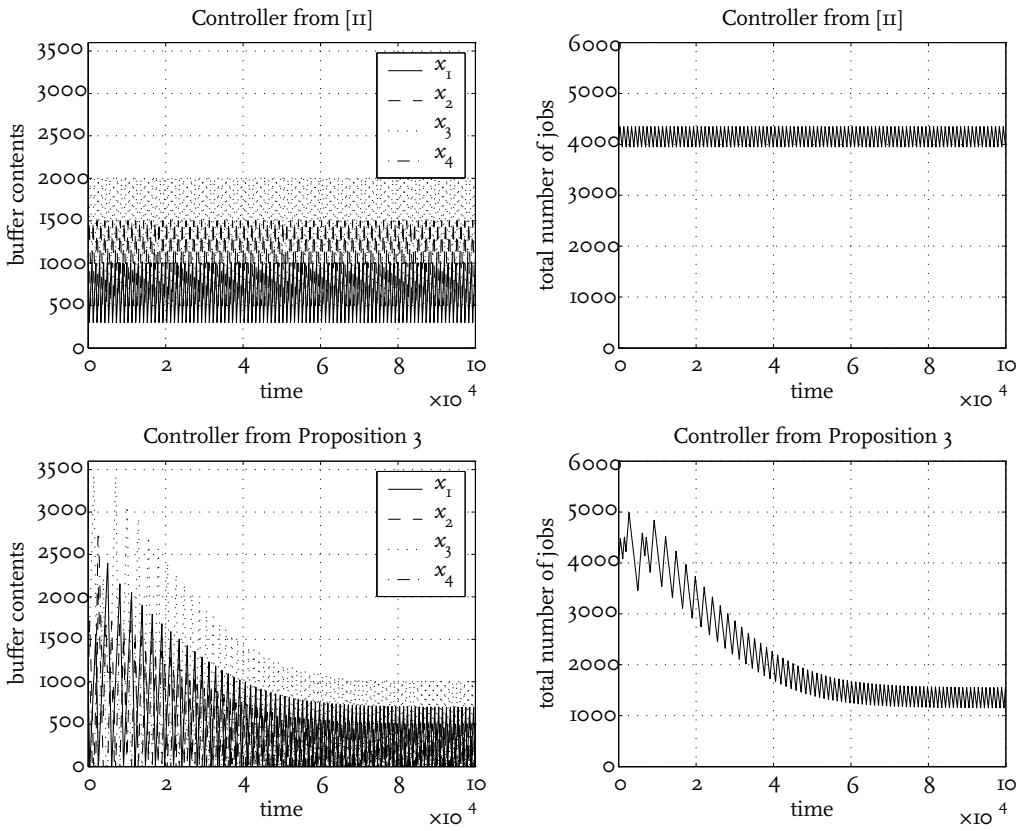

Figure 4: The buffer contents and total number of jobs for a deterministic system initiated in $(x_1, x_2, x_3, x_4) = (1000, 1000, 1000, 1000)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

towards this optimal behavior.
The resulting controller was a non-distributed controller, i.e. each machine needs to have global information for determining when to switch. Since we are in a manufacturing setting, this can be implemented rather easily. However, it turned out that this non-distributed controller can be implemented in a distributed way, i.e. such that each machine only requires local information for determining when to switch. We showed that these distributed controllers also guarantee convergence of the system towards the desired behavior. The proposed distributed controllers have been implemented successfully in a discrete event simulation study containing stochasticity.
Most of the literature on distributed control on networks starts from a policy which works for an arbitrary network and then analysis its performance. This is a good approach for networks that are unknown or frequently subject to change. However, for networks that are fully known, and not subject to change, a different approach should be used. In this paper we showed that it is possible to come up with distributed controllers that have been designed for a particular network in order to arrive at optimal network behavior. The resulting distributed controllers are different for each server and take into account knowledge of the network topology. Furthermore, the controllers do not fit in the class of standard controllers, such as clearing, gated, or k-limited policies.

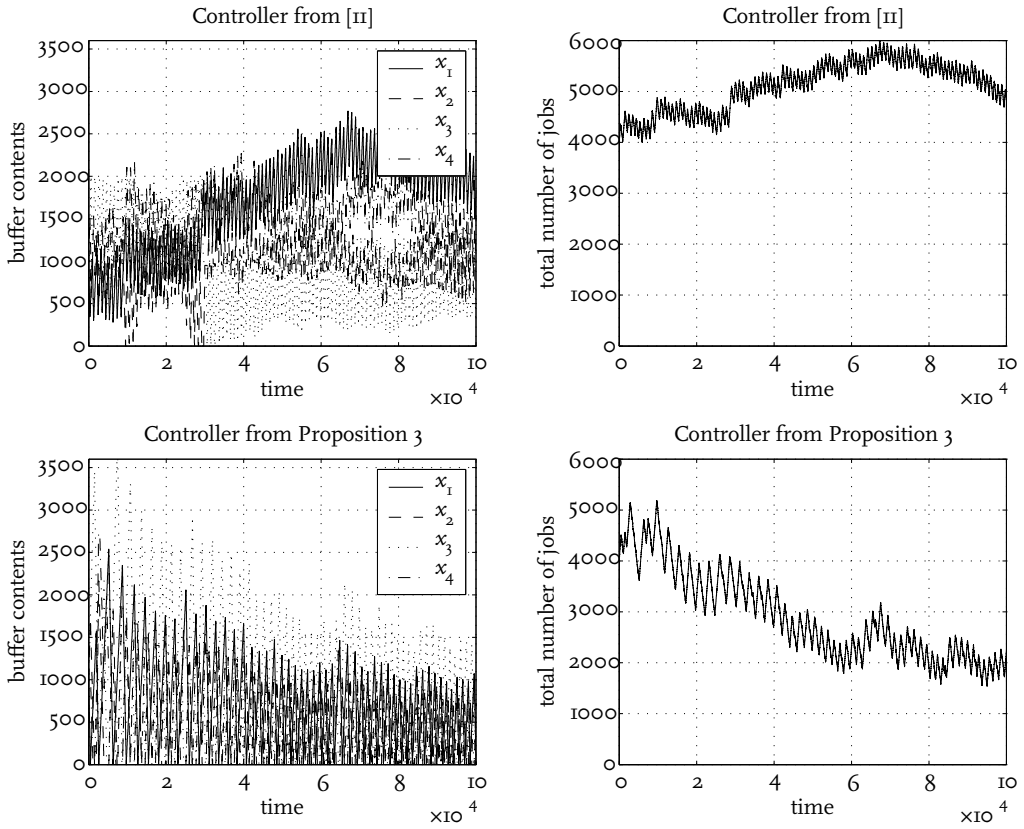

Figure 5: The buffer contents and total number of jobs for a stochastic system initiated in $(x_1, x_2, x_3, x_4) = (1000, 1000, 1000, 1000)$ for both the distributed controller proposed in [11] and the distributed controller of Proposition 3.

Acknowledgments

The authors sincerely thank Joost van Eekelen for providing the code for the simulations.

Bibliography

[1] J. Banks and J.G. Dai. Simulation studies of multiclass queueing networks. *IIE Transactions*, 29:213–219, 1997.

[2] J.A.W.M. van Eekelen, E. Lefeber, and J.E. Rooda. Feedback control of 2-product server with setups and bounded buffers. In *Proceedings of the American Control Conference*, Minneapolis, Minnesota, USA, June 2006.

[3] J.A.W.M. van Eekelen, E. Lefeber, and J.E. Rooda. State feedback control of switching servers with setups. *Discrete Event Dynamic Systems*, 2007. submitted.

[4] C. Humes, Jr. A regulator stabilization technique: Kumar Seidman revisited. *IEEE Transactions on Automatic Control*, 39(1):191–196, January 1994.

[5] P.R. Kumar and T.I. Seidman. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. *IEEE Transactions on Automatic Control*, 35(3):289–298, March 1990.

[6] E. Lefeber and J.E. Rooda. Controller design of switched linear systems with setups. *Physica A*, 363(1), April 2006.

[7] T.L. Olsen and W.-M Lan. Multi-product systems with both setup times and costs: fluid bounds and schedules. *Operations Research*, 2005. Accepted for publication.

[8] J. Perkins and P.R. Kumar. Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Transactions on Automatic Control*, 34(2):139–148, February 1989.

[9] J.R. Perkins, C. Humes, Jr, and P.R. Kumar. Distributed scheduling of flexible manufacturing systems: Stabiliy and performance. *IEEE Transactions on Robotics and Automation*, 10(2):133–141, April 1994.

[10] A.V. Savkin. Regularizability of complex switched server queueing networks modelled as hybrid dynamical systems. *Systems and Control Letters*, 35:291–299, 1998.

[11] A.V. Savkin. Optimal distributed real-time scheduling of flexible manufacturing networks modeled as hybrid dynamical systems. In *Proceedings of the 42nd Conference on Decision and Control*, pages 5468–5471, Honolulu, Hawaii, USA, December 2003.