Some kanban–controlled manufacturing systems: a first stability analysis

Thomas I. Seidman
Department of Mathematics and Statistics
University of Maryland Baltimore County
Baltimore, MD 21228, USA
(410)-455-2438
e-mail: ⟨seidman@math.umbc.edu⟩
and
Instituto de Matemática e Estatistica
Universidade de São Paulo

Carlos Humes Jr.
Departamento de Ciência de Computação
Universidade de São Paulo
São Paulo, SP, Brasil
e-mail: ⟨humes@ime.usp.br⟩

ABSTRACT: The blockage/starvation patterns of known instability examples suggest using local demand information — which is precisely what is provided by the widely advocated *kanban* approach to flow control in manufacturing systems. Therefore, we have re-analyzed for stability the examples described in the 1990 Kumar–Seidman paper when modified by introducing kanban control. It is found that this does *not* ensure stability and, in fact, some interesting new instability phenomena arise. Counterintuitively, it is possible that increasing some reserve supply level in a stable system may induce instability.

KEY WORDS: *manufacturing systems, instability, kanbans, decentralized scheduling.*

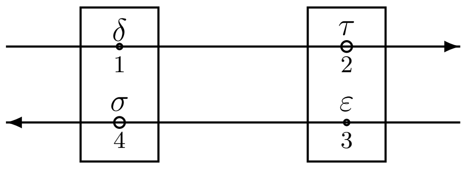
1. Introduction

The stability[1] of decentralized scheduling policies for manufacturing systems has been explored in numerous papers [10], [6], [8], [3], [9], [5], [12], [7], [11], [1]. All of these analyses (except [12]) have concerned themselves with *flow-driven* systems in which the scheduling of each machine is dependent only on the contents of the product buffers and such locally available information as might be carried by this product flow. In observing that the flow blockages occurring in known instability examples [6], [11], [1] might be alleviated by informing a supplier of impending 'starvation', one is led to conjecture a likelihood of greater stability for *demand-driven* systems, with dual flows of product and of controlling orders. This conjecture is somewhat strengthened by the popularity of one such approach, the *kanban*, as a scheduling control mechanism. Kanban systems have been widely advocated and widely implemented although, to the best of our knowledge, there has been no mathematically rigorous analysis of their stability properties.

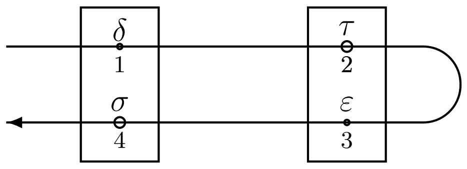
Thus we are led to test the efficacy of the kanban structure as a possible stabilizing control mechanism specifically against the challenge posed by the context of clearing policies, within which

[1] I.e., one wishes to ensure — subject to the 'capacity condition' (2.9) — that total WIP remains bounded in time, equivalently, that there is a bound on the throughput lag for each item of product. Conversely, 'instability' means the existence of counterexamples: systems satisfying the capacity condition for which, with some initial condition, one has WIP becoming unbounded in time.

were found (for flow-driven systems) the first two known instability examples [6]. Each example is a



Example A



Example B

system with two machines, each with two tasks. In each machine there is a comparatively slower task (indicated by an open dot in the figure) with the processing times satisfying the general 'capacity condition' ($\delta + \sigma < 1$, $\tau + \varepsilon < 1$) corresponding to assumed unit input rates. In [6], it was shown that instability was obtained if the processing times satisfied an 'instability condition'

$$\tau + \sigma > 1. \tag{1.1}$$

Those examples were themselves a response to the earlier general conjecture of stability of flow-driven clearing systems, presented in [10], where stability was proved for acyclic networks; as we shall see, even this result need not be valid in the presence of kanbans.

The results of our investigation seem interesting and somewhat unanticipated; see section 4.

2. Kanban systems: formulation and notation

We are considering a model for flow in a manufacturing system following the formulation of [10], etc. Thus, we have a finite set of *tasks*, indexed by i, with each task assigned to a *machine* $\mathcal{M}_m$; we write $m(i)$ for the machine index m associated with task i and correspondingly write $i \in \mathcal{M}_m$, identifying a machine with its set of tasks. Further, each task i is associated with a product stream $\mathcal{P}_p$ corresponding to a material flow of (items of) some product indexed by $p = p(i)$. We assume a fixed (pre-specified) sequencing of tasks within each product stream so the set $\mathcal{P}_p$ has a linear order $(i_1, \ldots, i_J)$. We write $i-$ and $i+$ for the *predecessor* and the *successor* (within $\mathcal{P}_p$) of a task i, where this is meaningful, and denote by $i_*(p)$ and $i^*(p)$ the initial and terminal tasks.

It may be possible to index the machines in such a way that one always has $m(i+) \geq m(i)$ and we then call the system geometry *acyclic*; otherwise (as is the case for our Examples A and B), we call the geometry *non-acyclic* or *re-entrant.*

The standard paradigm of queueing theory, followed in [10], etc., presumes a sequence of *arrivals* at the initial task of each product stream with the system 'pushed' by the incoming flow to maintain production. Our present paradigm is to reverse this so processing will only be done 'as ordered' — i.e., we presume a sequence of orders arriving to the terminal task of each product stream and view ourselves as processing orders which are transmitted in the product stream geometry in the direction opposite to our task sequencing and which 'pull' the concommittant flow and processing of product. To specify the operation of the manufacturing system it is now necessary to specify *both* the protocol for transmitting orders within the system and *also* the scheduling protocol for processing products at each machine when there may be competing tasks.

While this 'pull' framework provides attractive alternatives to the 'push systems' analyzed in [10], etc., it is clear that the analysis is now complicated by the necessity to provide *two* sets of protocols and to track two flows through the system: of orders and of products; this also blurs the distinction between acyclic geometries (no longer possessing the inductive causality as for 'push' systems) and non-acyclic.

In general, for pull systems, one will have for each task both a 'demand buffer' of as-yet-unfulfilled orders and a buffer of product available for processing. For present purposes we make no further differentiation (e.g., as to 'urgency') and it will be sufficient for our analysis to note only the *levels* of these buffers; thus, $k_i(t)$ will denote the number of orders awaiting (at time t) processing for task i, while $x_i(t)$ will denote the 'supply level' of available product there.

The *kanban mechanism* is a standard protocol for transmitting orders: it assumes specification[2] of a reserve supply level K_i so that orders (kanbans) are transmitted from i to $i-$ as $x_i(t)$ drops below K_i. Apart from some possible initial stock above the reserve supply level,one has the identity (for tasks which have internal suppliers)

$$x_i(t) + k_{i-}(t) \equiv K_i \tag{2.1}$$

(where we ignore any transit times for product or kanbans). For an initial task, we suppose product is always available as needed — effectively, that $x_{i_*(p)}(t) \equiv \infty$, without counting this as WIP. For a terminal task, the demand buffer does not consist of internally issued kanbans but of external demand and, without any equivalent of (2.1), we take

$$k_{i^*(p)}(t) =: z_p(t) \tag{2.2}$$

as a definition of z_p, which is just the unfullfilled external demand for the product stream $\mathcal{P}_p$.

We denote by $X_i(t)$ the cumulative number of items of product processed at task i by time t and by $Z_p(t)$ the number of orders received at $i^*(p)$ by time t, including any unfulfilled demand present in the system at the initial time. For an intermediate task i — equivalently for $i+$ in view of (2.1) — simple bookkeeping then gives

$$\begin{aligned} X_{i-}(t) - X_{i-}(s) &= [X_i(t) - X_i(s)] + [x_i(t) - x_i(s)] \\ [k_i(t) - k_i(s)] + [X_i(t) - X_i(s)] &= X_{i+}(t) - X_{i+}(s) \end{aligned} \tag{2.3}$$

For terminal tasks we have, instead,

$$[z_p(t) - z_p(s)] + [X_{i^*(p)}(t) - X_{i^*(p)}(s)] = [Z_p(t) - Z_p(s)]. \tag{2.4}$$

In general, a distributed processing protocol (scheduling policy) will schedule the activity of each machine $\mathcal{M}_m$ in terms of the values of $\{k_i, x_i : i \in \mathcal{M}_m\}$. For our present purposes, we will be considering scheduling policies in which a single task can be *enabled* at any time. We then distinguish three types of possible states $\omega_m(t)$:$\mathcal{A}_i$ (task i is enabled and active)

$\mathcal{I}_i$ (task i is enabled but inactive: the machine is 'waiting')

$\mathcal{S}_i^{i'}$ (task i is 'becoming enabled' in transition from the prior enabled task i')

A task $i \in \mathcal{M}_m$ can be active ($\omega_m = \mathcal{A}_i$) only if it is *available* (meaning that both $k_i(t) > 0$ and $x_i(t) > 0$). The state $\mathcal{S}_i^{i'}$ represents a *setup* period for the transition, which we assume has fixed length (setup time) $\delta_{i',i} \geq 0$.

We assume that the policy is *non-idling,* i.e., that no machine $\mathcal{M}_m$ will be idling (inactive: $\omega_m = \mathcal{I}_i$ for some i) unless it is *blocked* (meaning that *none* of the tasks at $\mathcal{M}_m$ is currently available). We also assume that we will not make a transition from i' to i unless the new task i is available. A *clearing policy* is one in which we further assume that we never make such a transition unless the previously enabled task i' has become unavailable, i.e., once some task $i \in \mathcal{M}_m$ is 'enabled', the machine $\mathcal{M}_m$ is committed to continue working exclusively at the task i (one has $\omega_m = \mathcal{A}_i$) until one of the relevant buffers (of orders or of available product) becomes exhausted.[3]

[2]Physically, one thinks of a fixed number K_i of 'tokens' (or 'order cards', to take a somewhat more literal translation of the Japanese word 'kanban') which would be attached to the stock of product at i when this is at its desired reserve level. When an item of this stock is utilized, the attached token is removed from the product item and transmitted as an order to the 'supplier' (predecessor) $i-$, to be returned attached to the replacement item when that order is fulfilled to resupply the reserve. See, e.g., Chapter 13 of [2] for a more detailed description; our model is there called a 'one card' kanban system.

Alternatively, this occurs in the Queueing Theory literature as a discipline 'with blocking': assume the supply buffer for task i has capacity K_i and that the prior server is 'blocked' (from task $i-$) if this buffer is already full. Noting (2.1), one easily sees that this discipline is precisely equivalent to the kanban mechanism for this link: if desired, the reader may eschew the explicit kanban description and view k_{i-} as the 'unused buffer capacity' $K_i - x_i$, rather than as an independent entity. [This protocol is *not* equivalent to the somewhat more commonly considered discipline for buffers of finite capacity in which 'excess' inputs are permitted to occur but are then discarded from the system.]

[3]The 'minimum assured run length' (no further input), is thus $\min\{k_i(t), x_i(t)\}$. When one of the buffers empties, some other task will be enabled, providing there is one available; otherwise the machine $\mathcal{M}_m$ necessarily idles until it does become possible to enable some other task. For our Examples A and B, there are only two tasks at each machine so all clearing policies are the same and one need not specify any selection procedure for the task to be enabled.

One principal motivation for clearing policies is that one may realistically have a substantial set-up time incurred when switching from one task to another and one then needs runs long enough to amortize the effect of this enforced nonproductive interval.

Associated with each task i is a *processing time* τ_i so, for any time interval (r, s), one has

$$\mu_i(r,s) := \text{meas}\,\{t \in (r,s) : \omega_m(t) = \mathcal{A}_i\} = \tau_i \cdot [X_i(s) - X_i(r)] \tag{2.5}$$

and for each machine $\mathcal{M} = \mathcal{M}_m$ one has

$$s - r \geq \sum_{i \in \mathcal{M}} \mu_i(r,s) = \sum_{i \in \mathcal{M}} \tau_i \cdot [X_i(s) - X_i(r)]\,. \tag{2.6}$$

We assume a (constant) *demand rate* $d_p > 0$ for each product stream (mean arrival rate of orders to $i^*(p)$) so

$$Z_p(t) - Z_p(s) = d_p \cdot (t - s) \qquad (t > s) \tag{2.7}$$

as closely as would be possible working with integers — or, more generally, that there is some 'burstiness constant' $M \geq 1$ for which

$$|[Z_p(t) - Z_p(s)] - d_p \cdot (t - s)| \leq M. \tag{2.8}$$

For the system to be *stable* (each $z_p(\cdot)$ uniformly bounded), each $X_i(\cdot)$ must track $Z_{p(i)}(\cdot)$ with bounded lag so, for long time intervals, one must have $X_i(s) - X_i(r) \approx d_{p(i)} \cdot (s - r)$. In view of (2.6), a corollary to this is the necessity for stability of the *capacity condition*

$$\rho_m := \sum_{i \in \mathcal{M}_m} d_{p(i)}\,\tau_i < 1 \qquad (\text{each } m). \tag{2.9}$$

What one would like, of course, is a stability result that — for any system using the protocols under consideration, any initial conditions, and any demand inputs $Z_p(\cdot)$ satisfying (2.8), subject to (2.9) — one would have each $z_p(\cdot)$ bounded (uniformly in time $t > 0$). The point of this paper is that there are quite simple and 'natural' systems for which this is *false* when using a kanban mechanism to transmit orders with a clearing policy to schedule production.

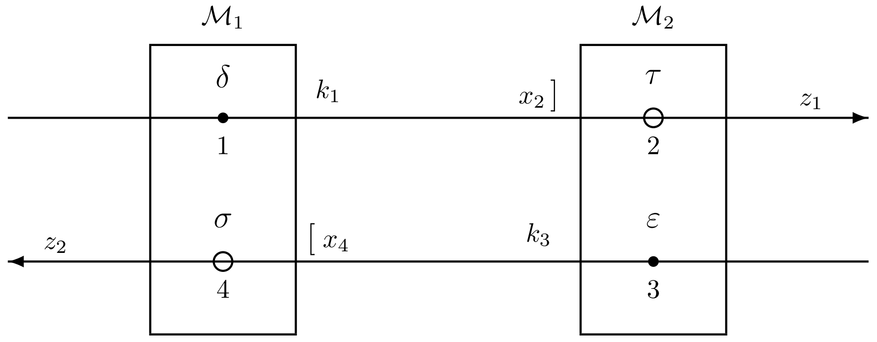
3. Three examples

To study the question of stability of systems operating with kanbans we shall start by analysing systems that have already been shown [6], to be unstable under flow-driven clearing policies.

For our present analysis — of systems under a clearing policy for production, much as in [6], and with order transmission now governed by a kanban mechanism — we consider the geometries, the demand rates, and the processing times for tasks to be fixed, with the reserve supply levels (numbers of kanbans) for each link then chosen in some fashion. This specifies a well-defined dynamical system. We then assume that the initial conditions (from which we begin tracking the scenarios) may be imposed arbitrarily. To simplify the exposition, we normalize to unit demand rates and assume negligible burstiness, i.e., each $d_p = 1$ and $M = 1$ in (2.8). For the first two examples we also neglect set-up times, taking each $\delta_{i',i} = 0$. In this case we can use a simplified set of activity states: we write $u_m = i$ rather than $\omega_m = \mathcal{A}_i$ and write $u_m = *$ when $\mathcal{M}_m$ is blocked; the vector $\mathbf{u} = (u_1, u_2)$ then gives the complete activity state for the system as a whole.

3:A. Two product streams

The first example is a two product system with two machines, corresponding to Example A above. As in [6], we shall consider that "fast" tasks feed "slower" tasks, i.e., given $\boldsymbol{\tau} = (\delta, \tau, \varepsilon, \sigma)$, we have $\delta, \varepsilon \ll \sigma, \tau$. The idea of *reserve supply level* applies only to tasks 2 and 4 and the corresponding numbers of available kanbans are respectively K_2 and K_4; to avoid some expositional complications, we assume in our analysis that $K_2, K_4 > 2$.



Example A (with kanbans)

We begin tracking this system at a moment t_0 corresponding to the completion at $\mathcal{M}_1$ of a clearing run for task 4 due to exhaustion of the demand buffer there, i.e., $z_2(t_0) = 0$. As initial condition, we assume that we have at $t = t_0$

(3:A.1) $$x_4 = K_4,\ k_3 = 0,\ \text{for } \mathcal{P}_2 \text{ and } x_2 = 0,\ k_1 = K_2 \text{ for } \mathcal{P}_1$$

with 'large' unfulfilled demand $z_1(t_0) = A$. At this moment, $\mathcal{M}_2$ is necessarily idle (task 2 starved for supplies; task 3 starved for demand), so the control state has just become $\mathbf{u}(t_0) = (1, *)$. The now-enabled task 1, with its rapid processing time, means that supply is quickly provided to task 2, enabling $u_2 = 2$ at time $t_0 + \delta$.

We are looking for a scenario in which the clearing run initiated at $\mathcal{M}_2$ ends by exhaustion of the unfulfilled demand z_1, terminating at the time $t_0 + T$ and n new orders will have arrived for $\mathcal{P}_1$ by then: $\mathcal{M}_2$ will have processed $A + n$ items, taking time $T - \delta = \tau(A + n)$ to make $z_1 = 0$. To within an uncertainty of at most 1 order (corresponding to the uncertain relation of the times t_0 and $t_0 + \delta + T$ to the exact arrival times for the orders; compare (2.8) with $d_1 = 1$, $M = 1$), we have $n \approx T$ so, for $\delta \ll \tau$, one has

(3:A.2) $$T \approx A\tau/(1 - \tau).$$

Of course, to have such a scenario requires that the product buffer at task 2 should never become exhausted during this interval ($x_2 > 0$) and we must determine conditions ensuring this; we make no claim that these conditions are actually *necessary* for instability — only that we will have shown an instability example when they do hold.

With task 1 enabled at $\mathcal{M}_1$ and $\delta \ll \tau$, it is clear that the clearing run at task 1 ends long before $t_0 + T$. Indeed, $\mathcal{M}_1$ will process $(K_2 + A + n)$ items in the interval $[t_0, t_0 + T]$, taking total time $\delta(K_2 + A + n)$ which will be much less than $T - \delta = \tau(A + n)$ if, say, $A > K_2$. Thus, since T will be large for A large, there must be (approximately) n arrivals at task 4 of orders for the $\mathcal{P}_2$ demand buffer so task 4 will certainly become enabled. To ensure that the scenario does proceed as we are describing it, it is only necessary to ensure that task 4 never has a clearing run long enough for processing at task 2 to empty its reserve supply buffer while task 1 is being temporarily blocked — i.e., since we necessarily have $x_2 = K_2$ at the beginning of such an 'intermediate' run at task 4, we must ensure that

(3:A.3) $$\tau K_2 \geq [\text{run length}] + \delta.$$

Since we are considering a period during which task 3 is blocked by task 2, the length of any run at task 4 is necessarily bounded by σK_4; however, this bound is inadequate for our purpose without imposing undesirable conditions on the choices of K_2, K_4 in relation to σ, τ. Alternatively, the run length here can be bounded in terms of the length T_1 of the preceding clearing run with $\mathbf{u} = (1, 2)$ which ends when the demand buffer at task 1 is exhausted. This will be longest for the first run at task 1, since that commences with k_1 having its maximum value K_2. $\mathcal{M}_1$ then processes the K_2 items already (initially) ordered at task 1 but, meanwhile, new orders will have been received from $\mathcal{M}_2$ so this run corresponds to processing $K_2 + n'$ items, taking time $T_1 := \delta(K_2 + n')$. This requires, of course, that n' kanbans are received to task 1 from $\mathcal{M}_2$ so $(n'-1)$ items must have been completed at task 2, taking time $\tau(n'-1)$. Thus,

(3:A.4)
$$\begin{aligned} &\delta + \tau(n'-1) < \delta(K_2+n') =: T_1 \le \delta + \tau n' \\ \text{i.e.,}\quad & n' = \left\lfloor (K_2-1)\frac{\delta}{\tau-\delta} \right\rfloor, \quad T_1 = \delta \left\lfloor \frac{K_2\tau - \delta}{\tau - \delta} \right\rfloor \approx \delta K_2 \end{aligned}$$

Letting T_2 be the time for which one then has $\mathbf{u} = (4,2)$, we see that $T_2 = \sigma n''$ where n'' is the number of orders arriving for $\mathcal{P}_2$ during the interval since task 4 was previously cleared. To within an uncertainty of 1 order, we have $n'' \approx T_1 + T_2 \approx (\delta K_2 + T_2)$ so $T_2 \approx [\sigma/(1-\sigma)]\delta K_2$. Comparing this to the requirement (3:A.3), we see that — essentially independently of the size of K_2 — we need δ small enough to have

(3:A.5)
$$\frac{\sigma}{1-\sigma}\delta < \tau$$

to ensure that the scenario proceeds here as described.

We now consider the situation at the time $t_1 = t_0 + T$ when the demand buffer at task 2 is finally emptied. If A was initially large, making T proportionately large by (3:A.2), there will have been plenty of slack time at $\mathcal{M}_1$ to have not only kept $x_2 \equiv K_2$ but also to have, altogether, cleared the reserve supply at task 4 of its original K_4 items — necessarily unreplenished because task 3 has been blocked throughout. As already noted, the number of orders arriving for $\mathcal{P}_2$ during this period will also have been approximately the same $n \approx T$ as for $\mathcal{P}_1$ so that the unfulfilled demand is then

(3:A.6)
$$z_2(t_1) =: A' = n - K_4 \approx \frac{\tau}{1-\tau}A - K_4$$

by (3:A.2). At t_1 the state will thus be

(3:A.7)
$$x_2 = K_2,\ k_1 = 0,\ \text{for } \mathcal{P}_1 \text{ and } x_4 = 0,\ k_3 = K_4 \text{ for } \mathcal{P}_2$$

with $\mathbf{z} = (0, A')$ and $\mathbf{u} = (*, 3)$.

This state is, of course, essentially a 'mirror image' of the initial state and, assuming A' is large enough, essentially the same argument *mutatis mutandis* shows that at a time t_2 we will again complete a clearing run at task 4 with the state given again by (3:A.1) and now with $\mathbf{z} = (A'', 0)$ where, as in obtaining (3:A.6), we now have

(3:A.8)
$$\begin{aligned} z_1(t_2) =: A'' &\approx \frac{\sigma}{1-\sigma}A' - K_2 \approx \lambda A - \left[\frac{\sigma}{1-\sigma}K_4 + K_2\right] \\ \text{with}\qquad & \lambda := \frac{\sigma\tau}{(1-\sigma)(1-\tau)}. \end{aligned}$$

To have $z_1(t_2) > z_1(t_0)$, we must then require that $\lambda > 1$ and that A is large enough that

(3:A.9)
$$(\lambda - 1)A > \left[\frac{\sigma}{1-\sigma}K_4 + K_2\right].$$

Note that $\lambda - 1 = (\sigma + \tau - 1)/(1-\sigma)(1-\tau)$ so

(3:A.10)
$$\lambda > 1 \text{ (giving instability)} \quad \Longleftrightarrow \quad \sigma + \tau > 1.$$

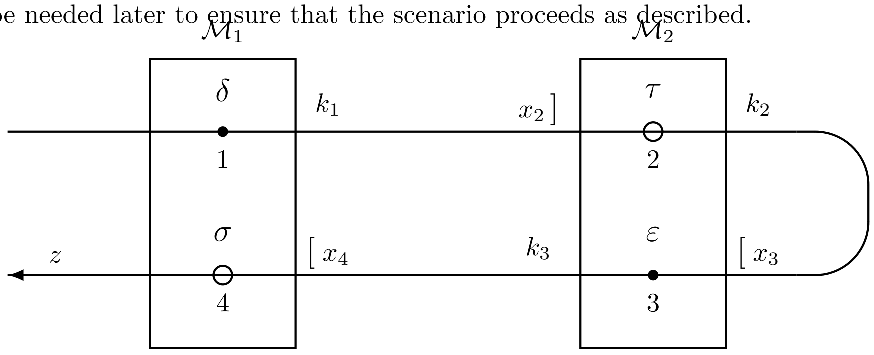
Asymptotically, we may eventually ignore the constant term $[\sigma/(1-\sigma)]K_4 + K_2$ in (3:A.8) and see that (approximately) each cycle repeats the same pattern with the unfulfilled demand multiplied repeatedly by the factor λ. Of course, the cycle time is also scaled proportionately, so the increase of unfulfilled demand will be approximately linear in time: the long-time average *rate of increase* of total WIP ($t^{-1}[z_1 + z_2](t)$ as $t \to \infty$) fluctuates boundedly — although without a limit unless $\sigma = \tau > 1/2$, in which case this is $(2 - 1/\tau)$ — with a positive lim sup, so WIP is certainly unbounded.

3:B. A single re-entrant line

We next analyze stability under kanban control for the second example [6] under the assumption

(3:B.1) $$\tau + \varepsilon < \sigma,$$

which will be needed later to ensure that the scenario proceeds as described.



Example B (with kanbans)

We begin tracking this system also at a moment t_0 corresponding to the completion at $\mathcal{M}_1$ of a clearing run for task 4 — now due to exhaustion of the product buffer, i.e., $x_4(t_0) = 0$. As initial condition, we assume that we have, then,

(3:B.2) $$k_3 = K_4, \quad x_4 = 0; \quad k_2 = K_3, \quad x_3 = 0; \quad k_1 = K_2, \quad x_2 = 0$$

with 'large' unfulfilled demand $z(t_0) = A$.

Note that at this initial moment $\mathcal{M}_2$ is necessarily idle (with both tasks starved for supplies), so the control state is $\mathbf{u}(t_0) = (1, *)$, but the now-enabled operation of task 1, with its rapid processing time, means that supply is quickly provided to task 2, enabling $u_2 = 2$, which we take as characterizing the first phase of this scenario.

Once task 1 is begun at $\mathcal{M}_1$, it will clear the demand indicated by the initial condition $k_1(t_0) = K_2$. Thus, noting that task 1 is assumed much faster than task 2, $\mathcal{M}_2$ will be adequately supplied to ensure that it will continue task 2 without interruption at least until completing $\min\{K_2, k_2(t_0) = K_3\}$ items — and, indeed, since task 3 is blocked during this and so cannot replenish the supply for task 4, $\mathcal{M}_1$ will continue to resupply the product buffer for task 2. Thus we continue to have $u_2 = 2$ until the demand buffer for task 2 is cleared, ending this phase, precisely after $\mathcal{M}_2$ has processed K_3 items. We note that at this point (time $t = t_1 = t_0 + \varepsilon + K_3\tau$) we have

(3:B.3) $$k_1 = 0, \quad x_2 = K_2, \quad k_2 = 0, \quad x_3 = K_3, \quad k_3 = K_4, \quad x_4 = 0$$

and that the combined number of items $x_2 + x_3$ at t_1 is just $K_2 + K_3$.

The clearing of the demand buffer for task 2 enables task 3 at $\mathcal{M}_2$ (i.e., $u_2 = 3$) so, much as above, product is rapidly supplied for task 4, which then permits $u_1 = 4$ (at time $t = t_1 + \delta$), which we take as characterizing the second phase of the scenario. One has, first, that $\hat{K} = \min\{K_3, K_4\}$ items are processed at task 3 while the same number of kanbans are sent to the demand buffer for the blocked task 2, after which one has $u_2 = 2$ while $\mathcal{M}_2$ clears at task 2 at the time $t = t_* = t_1 + \hat{K}\delta + \tilde{K}\tau$ by processing $\tilde{K} = \min\{\hat{K}, x_2(t_1) = K_2\} = \min\{K_2, K_3, K_4\}$ items without interruption. The condition (3:B.1) was imposed just to ensure that there is now still sufficient time for task 3 to recommence and begin resupplying the product buffer for task 4 before $\mathcal{M}_1$ clears the stock of $\hat{K}$ supplied earlier; thus (3:B.1) ensures the uninterrupted continuation of task 4 at $\mathcal{M}_1$. Continued tracking along the same lines easily shows that one has $u_1 = 4$ precisely until $\mathcal{M}_1$ has processed at task 4 a total of $K_2 + K_3$ items (i.e., one has processed all of the items which were in the product buffers x_2, x_3 at time t_1) since task 1 is necessarily blocked during this phase so no new product can be brought into the system.

The second phase then ends at the time $t = t_2$ given by

$$t_2 = t_1 + \delta + (K_2 + K_3)\sigma = t_0 + \delta + \varepsilon + K_3\tau + (K_2 + K_3)\sigma,$$

having processed, altogether, $K_2 + K_3$ items at task 4 to meet the initial unfulfilled demand. The 'internal' state of the system is now again exactly as in (3:B.2), so — apart from the level z of the external demand buffer — the same cycle will repeat exactly so long as z remains positive. During each cycle one has (with unit demand rate) the arrival of new demand in the form of (approximately)

$$[t_2 - t_0] = \delta + \varepsilon + K_3\tau + (K_2 + K_3)\sigma$$

new orders. Thus, one has approximately

(3:B.4) $$z(t_2) - z(t_0) \approx (t_2 - t_0) - (K_2 + K_3) \approx (K_2 + K_3)\left[\left(\sigma + \frac{K_3}{K_2 + K_3}\tau\right) - 1\right].$$

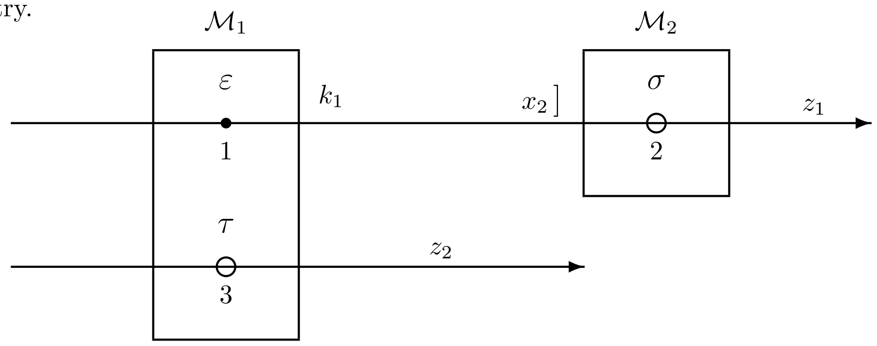
This is positive precisely when one has

(3:B.5) $$\sigma + \alpha\tau > 1 \qquad \alpha := \frac{1}{1 + K_2/K_3}$$

and z then increases per unit time by $1 - [K_2 + K_3] / [K_3\tau + (K_2 + K_3)\,\sigma]$.

3:C. An 'acyclic' instability example

We know (cf., e.g., [5]) that if each machine is governed by a 'usable policy' (i.e., stable for inputs of bounded burstiness in the context of any single server in isolation, for which we note the kanban mechanism is irrelevant), then one always has stability for acyclic flow-driven systems. In particular, we know that clearing policies are 'usable' in this sense for flow-driven servers and so always stable for acyclic geometries. As may be seen by the following example, this may fail for demand-driven systems with kanbans — while noting that we have here written 'acyclic', as this refers only to the flow geometry.



Example C

Here, rather than indexing as above, we refer to the tasks by their processing times (e.g., we say that '$\mathcal{M}_2$ services only task σ'). We will let K be the number of kanbans associated with the only link involving kanbans at all. We consider unit arrival rates for orders ($d_1, d_2 = 1$) and will neglect burstiness (effectively, $M = 1$) in (2.8)); we will also assume that ε is small enough and τ close enough to 1 to have

(3:C.1) $$K + 1 < \min\left\{\frac{\sigma}{\varepsilon}, \frac{\tau}{1-\tau}\right\}.$$

The significant setup time will be δ, on switching from task τ to task ε and we assume $\delta > 1$; for the other setup time (switching from ε to τ) we set $\delta' = 0$ for simplicity .

As we just have two tasks at $\mathcal{M}_1$, operated by a clearing policy, this machine will alternate between production runs at task τ and task ε. Let us consider a 'cycle' of the scenario characterized by this. Such a cycle comprises three phases: (• a setup period of duration δ, • a production run at task ε, processing m items, • a production run at task τ, processing n items) so the cycle duration is $T = \delta + m\varepsilon + 0 + n\tau$. Neglecting burstiness, our first assumption ($\delta > 1$) ensures that at least one order for $\mathcal{P}_1$ arrives at z_1 during the setup period and so ensures that task τ is always available at the conclusion of a production run at $\mathcal{M}_1$ of task ε and $n \geq 1$. Also, n will necessarily be large enough to ensure that $k_1 = K$ at the end of the second phase so task ε is available. The scenario will thus be as described.

We next note that the first part of our second assumption (3:C.1) implies that $m \leq K + 1$ items. To clear the order buffer k_ε, one need consider only the maximum K which could be at k_1 when the run begins, together with the additional order immediately returned by task σ when it begins work on the first item sent: since the time necessary to process $K + 1$ orders at task ε does not permit task σ actually to complete the processing of that first item, it will be unable to return an additional kanban. If, as we now assume, the initial state of the cycle has $k_1 = K$, then we have precisely $m = K + 1$ and we will then have to show that this state recurs at the next cycle.

Since $z_2 = 0$ both at the beginning and end of the cycle, n must be the number of orders for $\mathcal{P}_2$ arriving at z_2 during the cycle, i.e., $n = T$ (to within the uncertainty associated with working with discrete items). With $m = K + 1$, this gives

$$T = \delta + (K+1)\varepsilon + T\tau \quad \text{so } T = \frac{(K+1)\varepsilon + \delta}{1 - \tau}$$

The second part of (3:C.1) then ensures (as $\delta > 1$) that the duration $n\tau \approx T\tau$ of the third phase is at least $K + 1$. Since $\sigma < 1$, by (2.9), there is then certainly time during this for task σ to empty its supply buffer (at most K items) so we will have $k_\varepsilon = K$ at the end of the cycle, i.e., as the initial state for the next cycle as asserted.

Since $d_1 = 1$, the result we have obtained (that $n\tau > K + 1$) implies that we will have at least $K + 1$ orders for $\mathcal{P}_1$ arriving at z_1 during this phase and, with at least one additional order arriving during the setup period $\delta > 1$, that there must be at least $K + 2$ orders arriving during each cycle. On the other hand, only $m = K + 1$ items of $\mathcal{P}_1$ are processed during the cycle so z_1 must increase by at least one order per cycle: this unfulfilled demand increases unboundedly and the system is unstable.

We might also consider the system with the arrows in the diagram here reversed (so task τ supplies product $\mathcal{P}_1$ to task ε). This merely interchanges the roles of orders and products in this product stream so the system would follow essentially the identical scenario as above: again, z_1 must increase by at least one order per cycle and this 'reversed system' is also unstable.

We conclude this section by consideration of a single-machine, single-product system with two tasks (task σ supplying task τ with $\sigma + \tau < 1$ by (2.9)) and setup delays δ_σ, δ_τ. The system will alternate clearing K orders at task σ (taking time $\delta_\sigma + K\sigma$) and processing the K items then in the supply buffer at task τ (taking time $\delta_\tau + K\tau$). Thus, K items pass through the system for each such cycle which takes total time $T = (\delta_\sigma + \delta_\tau) + K(\sigma + \tau)$ with an average of T orders entering the system each cycle. If the setup delays are large enough and the reserve supply level K small enough that $[\delta_\sigma + \delta_\tau]/K > 1 - (\sigma + \tau)$, then this system will also be unstable — with, of course, the simple remedy of increasing K enough to reverse this inequality.

4. Summary and discussion

Let us first review the conclusions to be drawn from each of the examples of the preceding section:

- For the first example from [6] one finds that the kanban structure has little effect on the instability mechanism: as for the flow-driven case analyzed in [6] and for the identical parameter range (3:A.10), one develops a pattern of mutual blockage which repeats on an ever-increasing scale.

What is new, here, is a threshhold effect[4] for the initiation of this pattern, with the threshhold increasing when the number of kanbans for the links would be increased. If, rather than taking A large initially, we were to consider the system in operation subject to the burstiness condition (2.8), it is easy to see that a burst of demand permitted by (2.8) could also initiate the pattern described (if M is large enough) with the possibility that this might grow over some repetitions to the point where corresponding periods of reduced demand, also permitted by (2.8), could no longer interrupt the pattern. To the extent that the burstiness of (2.8) is random, one might expect this to occur (eventually) with probability 1 if possible at all. This possibility is essentially equivalent to having M larger than the minimal A described above, which increases with the K_j so, looking at the converse, the instability cannot occur if K_2, K_4 are large enough compared to the given M. This is consistent with the viewpoint of seeing kanbans as a stabilizing device to insulate the system from the effect of sudden fluctuations in demand.

- For the second example from [6], involving a single re-entrant line, one obtains with kanban operation a threshhold effect again and also a new instability criterion (3:B.5). Our earlier requirement of 'large' initial unfulfilled demand indicates that, as for Example A, the kanban mechanism provides a threshhold effect, insulating to some extent against occurrence of this instability.

  Since one always has $\alpha < 1$ in (3:B.5), this instability condition is favorably comparable to the instability condition (1.1) given for the 'same' system in [6] under flow-driven operation: the kanban mechanism helps, although it does not always ensure stability. Since this depends on $1 > \alpha = K_3/[K_2 + K_3]$, we see that for fixed processing times satisfying (2.9) one can always obtain stability by making K_2 large enough (for fixed K_3). However, if $\sigma + \tau > 1$, then increasing K_3 sufficiently (for fixed K_2) will make α close to 1 and $\sigma + \alpha\tau > 1$ so we have the surprising and counterintuitive fact that increasing the reserve supply level for a link may actually destabilize the system as a whole.

  We also see here a previously unobserved instability mechanism — different from that of [6] and from any previously known example of instability — in that instability occurs with repetition on a fixed scale rather than an exponentially increasing scale.

- Finally, noting that it had been shown [10] that clearing policies are stable for *acyclic* flow-driven systems, we attempted to obtain the corresponding result using the kanban structure — and, instead, obtained our Example C, demonstrating the possibility of instability in this setting as well. Clearly, the introduction of a setup time reduces system capacity, but since we have a strict inequality in (2.9), there is always margin to amortize this by taking long enough production runs, providing this were permitted by the scheduling protocol — indeed, this is the rationale for a clearing protocol and this logic serves quite well in the flow-driven acyclic case.

  We have demonstrated instability for these 'acyclic' settings — which could not have happened for a flow-driven policy. On the other hand, it should be noted that simply taking K large enough (for any fixed values of $\varepsilon, \sigma, \tau$) to avoid (3:C.1) invalidates the scenario description here, presumably giving stability. Of course, the former distinction between acyclic and non-acyclic systems has here been blurred, as already noted.

We were, perhaps, initially somewhat naive in expecting some miracles of stabilization from the use of the kanban mechanism — especially in the context of non-acyclic systems, once one notes that the typical applications in practice have been to acyclic production lines. With a very little thought, the results for Examples A and C seem unsurprising and serve only to corroborate a (corrected) intuition. We continue to find it somewhat counterintuitive that, in Example B, gaining instability may require *decreasing* one of the reserve supply levels. The nature of the repetitive pattern for

[4]This corresponds, roughly, to a protective indifference to operational fluctuations and limited burstiness of the input flow — here, demand rather than product arrivals. We have indicated this briefly — and suggest that this may explain why, with on-line tuning, instability is not observed in practice — but in this paper we have not attempted any formal analysis or proof of this effect.

this example (essential periodicity, rather than 'repetition on an increasing scale') represents a new phenomenon, although it is not clear whether this novelty is significant.

From the quite restricted analysis we have done here, we conclude that the introduction of a kanban mechanism with adequate reserve supply levels can be expected (for general flow geometries, as well as for acyclic production lines) to insulate the system from fluctuations without greatly affecting the anticipated stability analysis. On the other hand, such counterintuitive effects as exhibited in Example B suggest, as indicated by the phrase 'a first stability analysis' in the title, the desirability of making further careful stability analyses for particular cases of interest.

References

[1] M. Bramson, *Instability of FIFO queueing networks*, Ann. Appl. Prob., pp. 414-431, (1994).

[2] J. Browne, J. Harhen, and J. Shivnan, *Production Management Systems; a CIM perspective*, Addison–Wesley, Reading, 1988.

[3] C. J. Chase and P. J. Ramadge, *On real time scheduling policies for flexible manufacturing systems*, IEEE Trans. Autom. Control **AC-37**, pp. 491-496 (1992).

[4] C. Chase, J. Serrano, and P. Ramadge, *Periodicity and chaos from switched flow systems: Examples of the discrete control of continuous systems*, IEEE Trans. Autom. Control **AC-38** 1993, pp. 70–83.

[5] C. Humes Jr., *A regulator stabilization technique: Kumar-Seidman revisited*, IEEE Trans. Autom. Control **AC-39**, pp. 191-196 (1994).

[6] P.R. Kumar and T.I. Seidman, *Dynamic instabilities and stabilization methods in distributed real time scheduling of manufacturing systems,* IEEE Trans. Autom. Control **AC-35**, pp. 289–298 (1990) [see also, pp. 2028–2031 in *Proc. 28th IEEE CDC,* IEEE (1989)].

[7] S. Lou, S. Sethi, and G. Sorger *Analysis of a Class of Real-Time Multiproduct Lot Scheduling Policies*, IEEE Trans. Autom. Control **AC-36**, pp. 243-248 (1991).

[8] S. H. Lu and P.R. Kumar, *Distributed scheduling based on due dates and buffer priorities*, IEEE Trans. Autom. Control **AC-36**, pp. 1406-1416 (1991).

[9] J.R. Perkins, C. Humes, and P.R. Kumar, *Distributed control of flexible manufacturing systems: stability and performance*, IEEE Trans. Rob. and Automation **10**, pp. 133-141 (1994).

[10] J.R. Perkins and P.R. Kumar, *Stable distributed real-time scheduling of flexible manufacturing/ assembly/ disassembly systems*, IEEE Trans. Autom. Control **AC-34**, pp. 139–148 (1989).

[11] T.I. Seidman, *'First Come, First Served' can be unstable!*, IEEE Trans. Autom. Control **AC-39**, pp. 2166–2171 (1994).

[12] A. Sharifnia *Stability and performance of distributed production control methods based on continuous flow models*, pre-print, Boston Univ., (1993).