EURANDOM PREPRINT SERIES
2010-022

Model Predictive Control for the Acquisition Queue and Related Queueing Networks

J. van Leeuwaarden, E. Lefeber, Y. Nazarathy, J. Rooda

ISSN 1389-2355

Model Predictive Control for the Acquisition Queue and Related Queueing Networks

Johan S.H. van Leeuwaarden[a,b], Erjen Lefeber[c],
Yoni Nazarathy[b,c] and Jacobus E. Rooda[c]

[a]*Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands*
[b]*EURANDOM, The Netherlands*
[c]*Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands*

Abstract

Model Predictive Control (MPC) is a well established method in control theory and engineering practice. It is often the method of choice for systems that need to be controlled in view of constraints. The main idea of MPC is to solve an optimization problem over a given time horizon at each control epoch, and to use the obtained solution for controlling the system until the next control epoch. In many cases this optimization problem can be formulated as a tractable quadratic programming problem. In this paper we apply MPC to discrete-time queueing networks that incorporate multiple product routes, delays, self-generated arrivals and multi-job processing. We compare our MPC-based controller to simple threshold policies and show improved performance. We also mention some open theoretical problems related to this control method.

I. Introduction

Queueing networks are often used to model communication, manufacturing and service networks. Emphasis usually lies on the performance analysis of a model for a given service discipline and routing scheme. It is often challenging to determine the optimal way of scheduling capacity or routing jobs. This paper explores methods for finding such dynamic control rules.

When precise objectives are formulated, for example minimizing steady-state queue sizes, the optimal control problem can in principle be formulated as a Markov Decision Problem (MDP), but in practice, MDPs for larger networks often prove numerically cumbersome. An alternative is to employ much simpler control laws such as priority or threshold policies, but these typically fail to achieve the desired behavior. Analysis and tuning of MDPs and simple control laws can sometimes benefit from using asymptotic scaling regimes. In such cases, there is often some simplification of the underlying stochastic process, and intractable MDPs can then be approximated by more tractable Brownian control problems. Nevertheless, even when using such asymptotic approximations, finding optimal controls is still challenging. For more background see [3], [10], [15].

In this paper we present an alternative control methodology which to the best of our knowledge has not been applied before to queueing networks. Our method uses the concept of Model Predictive Control (MPC). A classic reference is [6]. MPC is a popular tool for generating feedback controllers for dynamical systems (i.e. control laws that observe the current state and use it to decide on the next control action). MPC can deal with non-linearities and state constraints. Two recent papers that apply MPC to logistical and manufacturing networks are [12] and [13], but in general, MPC is not as well known in the operations research community as it is in the systems and control community.

The concept of MPC is simple: At every control epoch solve an optimization problem for the optimal trajectory into the future, then use the first step of the optimal trajectory as the current control decision.

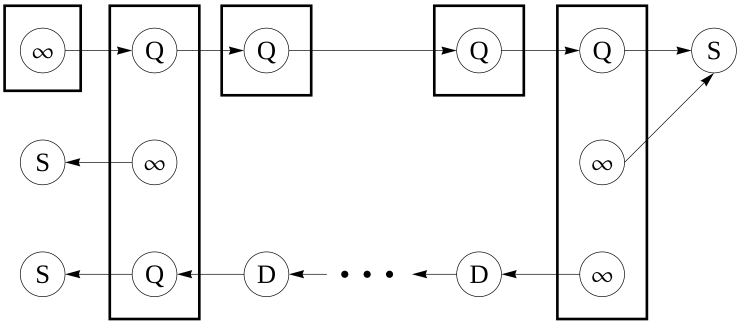


Fig. 1. An example network. This network has 5 servers, 4 routes, 3 sinks, 4 infinite supply sources, 5 queues and a delay of arbitrary size. The control law of the network is a specification of how the servers allocate their effort among their constituent job classes.

At the next time step solve the optimization problem again, and so on. This method allows to incorporate predictions in the control mechanism and has proven useful for systems with delayed feedback. Some theoretical properties of this method have also been established [2], [14]. Specifically, conditions for obtaining a stabilizing controller in the case of deterministic systems are by now well known (and even implemented in commercial control packages [11]). Contrary to the deterministic case, quantifying the performance of stochastic systems controlled by MPC is a largely unexplored area.

The network models we consider are discrete-time multi-class queueing networks that allow for delays, multi-job processing and self-generated arrivals. The latter is motivated by both the acquisition queue [5] and multi-class queueing networks with infinite virtual queues [16], [20]. In these types of queueing models, a server in the network can generate arrivals and is often faced with the control choice of either letting new jobs enter the system or serving jobs that are presently waiting for service.

Figure 1 depicts an example network. The five boxes represent servers. The circles represent job classes. Classes marked with the ∞ sign are called *source classes*. These represent an infinite amount of jobs waiting to enter the system. Classes marked with Q represent *queue classes*; each queue class has an associated queue in which jobs are waiting to be served. Classes marked with D represent *delay classes*. When jobs pass through a delay class they are delayed for one time unit. Concatenating d delay classes results in a delay of d time units. Finally, classes marked with an S are *sink classes* and represent jobs that have left the system. We choose to model these sink classes in order to keep track of the departure processes.

Source and queue classes are associated with servers. In Figure 1, three servers are associated with a single class and two servers are associated with three classes each. All classes associated with the same server have to share that server's capacity. The delay and sink classes are not associated with servers. Jobs that finish processing in a source or queue class, or a unit delay in a delay class, move to the next *downstream* class. This is indicated by the arrows. All routes are deterministic and end up in sink classes.

The networks evolve at discrete time points $n = 0, 1, \ldots$. At each time point all servers allocate their capacities among their constituent source and queue classes. The way in which this capacity is allocated is specified by the control law, the main object of our study. Capacity is used by the source classes to generate new jobs and by the queue classes to process jobs in their queue. Every unit of capacity allocated to a queue class results in the removal of a single job from the queue. Every unit of capacity allocated to a source class results in the generation of a random number of new jobs. Delay classes simply pass, after one time unit, all of their jobs to their downstream class. Thus the randomness in our system is caused by the uncertainty in the number of jobs generated by source classes.

The structure of the paper is as follows. Section II illustrates our basic concepts by means of an example.

Section III describes in detail the queueing network model, presented both as a controlled Markov chain and as a controlled linear system with noise. Section IV describes the MPC based controller, including details of the implementation. Section V presents some further simulation experiments that provide insight into some of the design issues related to MPC. We also mention some open problems in Section V, and conclude in Section VI.

II. An illustrative example – the acquisition queue

To demonstrate the MPC approach we consider the acquisition queue that was introduced in [5]. In this stochastic model, a server has to divide its capacity among the acquisition of new jobs and the service of jobs that are presently waiting in the queue. Within each time slot the server has $c \in \mathbb{N}$ units of capacity to spend. Each unit spent on acquisition in time slot n generates a random number of new jobs that join the queue after a delay of $d \in \mathbb{N}$ time units. Each unit spent on service removes one job from the queue. Figure 2 presents a schematic representation of the model.

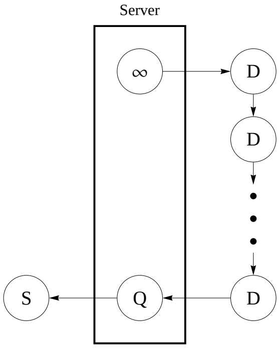


Fig. 2. The acquisition queue falls within our framework. It is a single server that can either acquire new jobs (by working on the source class ∞) or service jobs in its queue. There is a delay of d time units between the time a job is acquired and its arrival to the queue. Serviced jobs accumulate in the sink.

We denote by $U(n)$ the number of acquisition efforts at time n and refer to this choice as the control law of the system. The queue length process $\{Q(n)\}_{n\in\mathbb{N}}$ is then described by the recursion equation

$$Q(n+1) = \big(Q(n) - (c - U(n))\big)^+ + \sum_{i=1}^{U(n-d)} \tilde{u}_{n,i}, \quad n = 0, 1, \dots. \tag{1}$$

Here, $x^+ = \max(0, x)$ and $\{\tilde{u}_{n,i}\}_{n\in\mathbb{N}}$ is a sequence of i.i.d. non-negative integer-valued random variables. The initial conditions are $Q(0)$ and $U(-d)$, $U(-d+1), \dots, U(-1)$. The control $U(n) \in \mathbb{N}$ is a function of $Q(n)$ that satisfies

$$\big(c - Q(n)\big)^+ \le U(n) \le c.$$

Let $S(n)$ be the number of jobs in the sink (jobs that have received service) by time n, with $S(0) = 0$. If we assume that the system is stable and well defined (see [5]), the throughput of the system satisfies

$$\delta = \lim_{n\to\infty} \frac{1}{n} S(n) = m\frac{c}{1+m},$$

with $m = \mathbb{E}[\tilde{u}_{1,1}]$. In [5] the following simple control law was analyzed in full detail:

$$U(n) = \alpha + (c - Q(n))^+, \tag{2}$$

with $\alpha \in \mathbb{N}$ some number for which $\alpha < c/(1+m)$ (for stability).

A sensible control law $U(\cdot)$ should stabilize the system, and in addition, keep the queue lengths relatively small. Consider the *sink error* $S^e(n) = S(n) - \delta n$. If we are able to produce at rate δ then

$\lim_{n\to\infty} S^e(n)/n = 0$. The time-dependent behavior of $S^e(n)$ is also of interest: highly fluctuating or periodic behavior is considered undesirable. Finally, it is desirable for the control law to be computed relatively quickly. The threshold policy (2) certainly meets this criterion. As opposed to that, an MDP formulation of the acquisition queue (or more complex networks) typically does not. The MPC based controller presented in this paper does allow for quick evaluation.

We shall now compare the threshold policy in (2) to MPC[1].

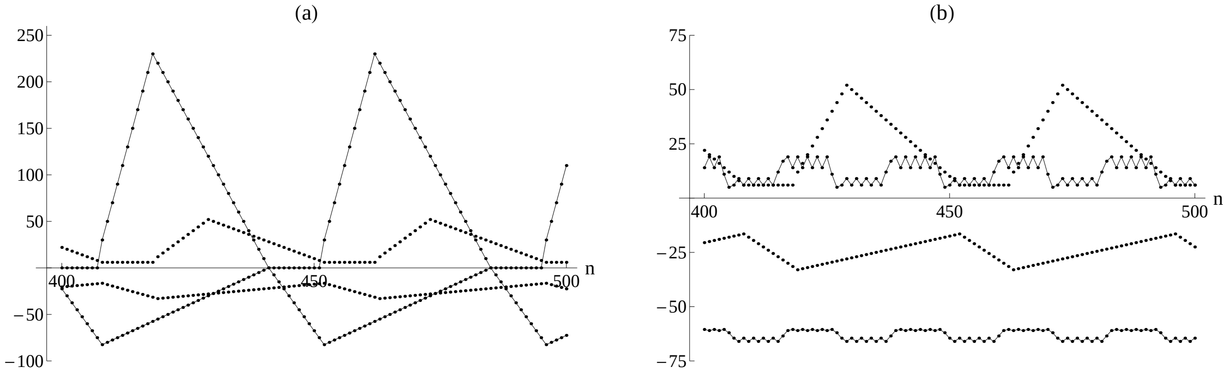


Fig. 3. Trajectories of the acquisition queue with deterministic acquisition sizes, $d = 10$ and $m = 3$. Trajectories above the axis are the queue level. Trajectories below the axis are the sink error. (a) Compares the threshold policy with $\alpha = 0$ (dots with line) and $\alpha = 2$ (dots only). (b) Compares the threshold policy with $\alpha = 2$ (dots only) and an MPC based controller (dots connected by a line).

Figure 3(a) considers the threshold policy and shows the evolution of $Q(n)$ (above the axis) and $S^e(n)$ (below the axis). In this example, the system is deterministic (i.e. $\text{Var}(\tilde{u}_{1,1}) = 0$), and $c = 10$, $d = 10$, $m = 3$. The figure compares $\alpha = 0$ and $\alpha = 2$. It appears that $\alpha = 2$ performs better, both in terms of queue lengths and fluctuations in sink error. In Figure 3(b) we compare the threshold policy with $\alpha = 2$ and MPC. The latter clearly performs better. Figure 4 further compares the threshold policy and MPC, but this time for stochastic acquisition sizes. Again, MPC outperforms the threshold controller.

We now briefly outline how our MPC control for the acquisition queue comes about. For simplicity, take $d = 3$, so that the complete state at time n is described by $Q(n)$, $S(n)$, and the acquisition during the last three time units denoted by $D_1(n), D_2(n)$ and $D_3(n)$.

Assume now that our control law is a general function of the state, and given by both the acquisition effort $U_\infty(n)$ and the service $U_Q(n)$, which satisfy

$$U_\infty(n) + U_Q(n) \le c \quad \text{and} \quad U_Q(n) \le Q(n). \tag{3}$$

The evolution of our system is then described as

$$\begin{bmatrix} D_1(n+1) \\ D_2(n+1) \\ D_3(n+1) \\ Q(n+1) \\ S(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} D_1(n) \\ D_2(n) \\ D_3(n) \\ Q(n) \\ S(n) \end{bmatrix} + \begin{bmatrix} m & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} U_\infty(n) \\ U_Q(n) \end{bmatrix} + \text{noise}, \tag{4}$$

where the noise term is implicitly specified: its first coordinate is

$$\sum_{i=1}^{U_\infty(n)} \tilde{u}_{n,i} - U_\infty(n)m,$$

[1] With $Q = I$, $R = I$ and $N = 30$, see Section IV.

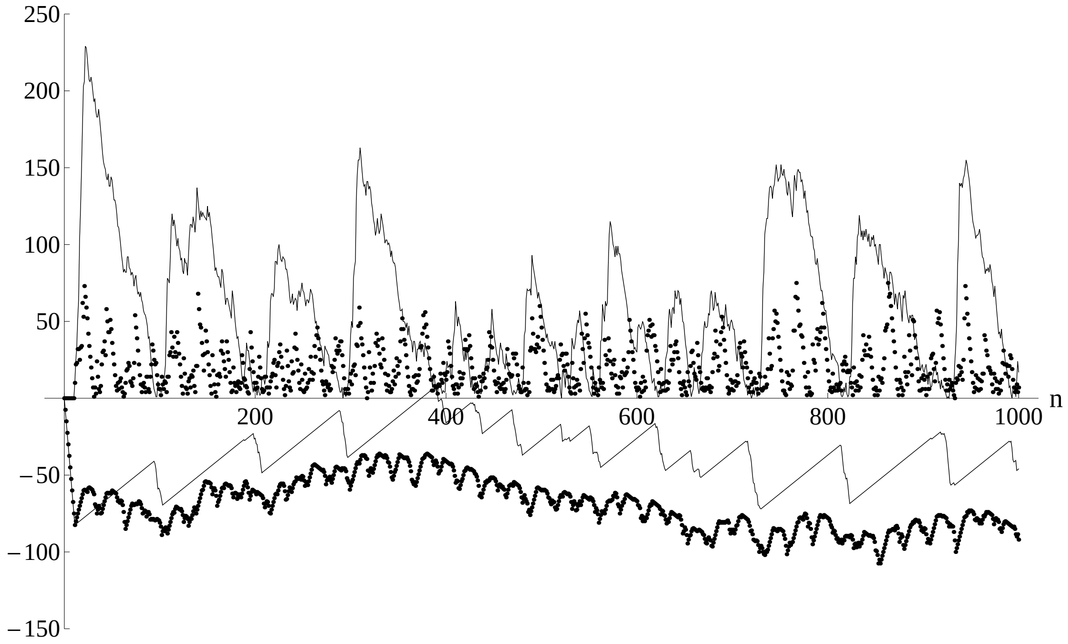


Fig. 4. Trajectories of the acquisition queue with geometrically distributed acquisition sizes, $d = 10$ and $m = 3$. Trajectories above the axis are the queue level. Trajectories below the axis are the sink error. The thin solid curves are for the threshold policy with $\alpha = 2$. The dotted curves are for the MPC based controller.

while all its other coordinates are zero.

The first step of our approach is to identify a *reference trajectory* by studying the system without noise. In principle we may choose any reference trajectory. For the acquisition queue, to make a fair comparison with the controller in (2), we choose as a reference the trajectory in which the sink grows linearly at rate δ and all other quantities remain constant. That is,

$$\bar{D}_1^r(n) = \bar{D}_2^r(n) = \bar{D}_3^r(n) = \bar{Q}^r(n) = \bar{U}_\infty^r(n) = \bar{U}_Q^r(n) = \delta, \quad \bar{S}^r(n) = \delta\, n.$$

We now seek to formulate a *state feedback control law* that attempts to drive the system to the reference. MPC does this by solving, at each time n, the following optimization problem:

$$\begin{aligned} &\min \sum_{i=n}^{n+N-1} \Big(||\hat{X}(i+1) - \bar{X}^r(i+1)|| + ||U(i) - \bar{U}^r(i)|| \Big) \\ &\text{s.t.} \quad \text{constraints (3).} \end{aligned} \tag{5}$$

Here $\hat{X}(n)$ is the prediction of the state at time n (a vector of dimension $d+2$ for the acquisition queue), $\bar{X}^r(\cdot)$ is the state part of the reference trajectory, $U(\cdot)$ is the vector of controls (2-dimensional for the acquisition queue) and $||\cdot||$ is some norm or some norm-like measure (in this paper we consider a 2-norm, i.e. quadratic costs). The minimization is over the controls which are to be applied over the time horizon $n, n+1, \ldots, n+N-1$. This is a $2N$-dimensional vector for the present case. The predicted state $\hat{X}(\cdot)$ is a function of the decision variables and the current state.

MPC works as follows: At each time point the optimal solution of (5) is used to control the system for the next time unit. Note that as opposed to "myopic control" which attempts to make the next "best step", MPC is "look ahead control". It appears to be very suitable for queueing networks of the type we describe. The full details of the controller are surveyed in Section IV. First, we formally define the general network model in Section III.

III. THE NETWORK MODEL

We consider multi-class queueing networks that evolve at discrete time points $n = 0, 1, 2, \ldots$. A network is composed of four types of job classes: *source* (∞), *queue* (Q), *delay* (D) and *sink* (S). Jobs originate in source classes, pass through queue and delay classes, and eventually end up in sink classes.

Let K_∞, K_Q, K_D, K_S denote the number of classes of each type. Classes are indexed by $k = 1, \ldots, K$. We denote the classes of type j by $\mathcal{K}_j$ for $j \in \{\infty, Q, D, S\}$ and we number the classes as follows: $\mathcal{K}_\infty = \{1, \ldots, K_\infty\}$, $\mathcal{K}_Q = \{K_\infty + 1, \ldots, K_\infty + K_Q\}$, etc. For convenience, we use the notation $\mathcal{K}_{\{j,j'\}}$ to indicate the union of $\mathcal{K}_j$ and $\mathcal{K}_{j'}$, for $j, j' \in \{\infty, Q, D, S\}$, sometimes extending the notation to three indices. For example, $\mathcal{K}_{\{\infty,Q,D\}}$ are all classes except the sink classes. Let $K_{\{j,j'\}} = |\mathcal{K}_{\{j,j'\}}|$.

The jobs are processed by L servers indexed by $i = 1, \ldots, L$. These servers perform activities on the source and queue classes. The activity of each such class is performed by a unique server $\sigma(k)$. Let

$$C(i) = \{k \in \mathcal{K}_{\{\infty,Q\}} : \sigma(k) = i\}$$

denote the constituency of server i. Let C denote the $L \times K_{\{\infty,Q\}}$-dimensional constituency matrix. Element (i, k) of this matrix is 1 if $k \in C(i)$, and 0 otherwise. We partition C as

$$C = \begin{bmatrix} C_\infty & C_Q \end{bmatrix}.$$

At each time point, each server i must divide its effort between the constituency activities $C(i)$, generating new material from the source classes, processing existing material in queue classes, idling or performing a combination of these. The number of activities that server i can perform in one time unit is given by the integer $c_i \geq 1$. Let c denote the vector of these elements. Let $U_k(n), k \in \mathcal{K}_{\{\infty,Q\}}$ denote the number of activities that are actually applied to class k at time n. We thus have the constraints

$$\sum_{k \in C(i)} U_k(n) \leq c_i, \quad i = 1, \ldots, L. \tag{6}$$

With each source class $k \in \mathcal{K}_\infty$, we associate a sequence of non-negative i.i.d. *inputs*: $\{\tilde{u}_k(\ell), \ell = 1, 2, \ldots\}$, with $\mathbb{E}[\tilde{u}_k(1)] = m_k$. Let M_∞ be a diagonal matrix with the elements $m_k, k \in \mathcal{K}_\infty$. We denote generic random variables of these i.i.d. sequences by $\tilde{u}_k$, $k \in \mathcal{K}_\infty$. Further, use $\tilde{u}_k^{*v}$ to denote generic random variables whose distribution is the v-fold convolution of the distribution of $\tilde{u}_k$. The action of performing an activity on class $k \in \mathcal{K}_\infty$ is the creation of $\tilde{u}_k$ new jobs. Thus the application of $U_k(n)$ units of processing on class k results in the creation of $\tilde{u}_k^{*U_k(n)}$ jobs. Obviously, the mean of this quantity is $U_k(n)m_k$.

Jobs that leave a class move to downstream classes using some routing scheme. Downstream classes may be of the queue, delay, or sink type. If it is a queue class, number k say, the job joins the queue and awaits service from server $\sigma(k)$. If it is a delay class the job is delayed for one time unit, before moving onto the next downstream class in the next time unit. If it is a sink class, the job has reached its final destination.

In this paper we assume that routing is deterministic. We let $p_{kk'} = 1$ if jobs that move out of class $k \in \mathcal{K}_{\{\infty,Q,D\}}$ move into class $k' \in \mathcal{K}_{\{Q,D,S\}}$. Otherwise $p_{kk'} = 0$. These values are arranged in a matrix $P = (p_{kk'})$, which we partition as

$$P = \begin{bmatrix} P_{\infty Q} & P_{\infty D} & P_{\infty S} \\ P_{QQ} & P_{QD} & P_{QS} \\ P_{DQ} & P_{DD} & P_{DS} \end{bmatrix}.$$

We assume that all routes end up in sink classes. Note that routes may merge.

The state of the network is the number of jobs in the queue, delay and sink classes. We denote it by the $K_{\{Q,D,S\}}$-dimensional vector $X(n)$ partitioned as $[\ X_Q(n)\ X_D(n)\ X_S(n)\ ]$. The state evolves as

$$X_k(n+1) = \begin{cases} X_k(n) + \sum_{k'\in\mathcal{K}_D} X_{k'}(n)p_{k'k} + \sum_{k'\in\mathcal{K}_{\{Q,\infty\}}} \tilde{u}_{k'}^{*U_{k'}(n)} p_{k'k} - U_k(n), & k\in\mathcal{K}_Q \text{ (queue)},\\ \sum_{k'\in\mathcal{K}_D} X_{k'}(n)p_{k'k} + \sum_{k'\in\mathcal{K}_{\{Q,\infty\}}} \tilde{u}_{k'}^{*U_{k'}(n)} p_{k'k}, & k\in\mathcal{K}_D \text{ (delay)},\\ X_k(n) + \sum_{k'\in\mathcal{K}_D} X_{k'}(n)p_{k'k} + \sum_{k'\in\mathcal{K}_{\{Q,\infty\}}} \tilde{u}_{k'}^{*U_{k'}(n)} p_{k'k}, & k\in\mathcal{K}_S \text{ (sink)}. \end{cases}$$

In the above we use the convention $\tilde{u}_k = 1$ for $k \in \mathcal{K}_Q$, i.e. for these classes $\tilde{u}_k^{*U_k(n)} = U_k(n)$. This allows us to represent the job inflow resulting from $U_k(t)$ in the same manner for both the queue and the source classes. Queue activities require material to be in the queue for the activity to be performed. We thus have the constraints

$$U_k(n) \le X_k(n), \quad k \in \mathcal{K}_Q. \tag{7}$$

When the control $U(n)$ is a well specified function of the state $X(n)$ that satisfies constraints (6) and (7) we refer to it as a *state feedback control*. This makes $\{X(n)\}_{n=0}^{\infty}$ a Markov chain.

Description as a linear system with noise

As an alternative to the Markov chain representation, we now represent our network as a linear system with control-dependent zero-mean non-Gaussian noise. That is,

$$X(n+1) = AX(n) + BU(n) + G\tilde{u}\big(U(n)\big). \tag{8}$$

Here, A, B and G are matrices to be defined below and $\tilde{u}\big(U(n)\big)$ denotes a K_∞-dimensional zero-mean random vector with the kth element distributed as

$$\tilde{u}_k^{*U_k(n)} - U_k(n)m_k.$$

Observe that the elements of this noise vector are always zero when $\tilde{u}_k$ is deterministic. Further, when the control action $U_k(n)$ is a large number, the kth noise element is approximately Gaussian distributed.

The matrices A, B and G are now spelled out (I is the identity matrix):

$$\begin{bmatrix} X_Q(n+1) \\ X_D(n+1) \\ X_S(n+1) \end{bmatrix} = \begin{bmatrix} I & P'_{DQ} & 0 \\ 0 & P'_{DD} & 0 \\ 0 & P'_{DS} & I \end{bmatrix} \begin{bmatrix} X_Q(n) \\ X_D(n) \\ X_S(n) \end{bmatrix} + \begin{bmatrix} P'_{\infty Q}M_\infty & P'_{QQ} - I \\ P'_{\infty D}M_\infty & P'_{QD} \\ P'_{\infty S}M_\infty & P'_{QS} \end{bmatrix} \begin{bmatrix} U_\infty(n) \\ U_Q(n) \end{bmatrix} + \begin{bmatrix} P'_{\infty Q} \\ P'_{\infty D} \\ P'_{\infty S} \end{bmatrix} \tilde{u}\big(U_\infty(n)\big).$$

The control $U(n)$ needs to satisfy linear constraints of the form

$$F \begin{bmatrix} X(n) \\ U(n) \end{bmatrix} \le g,$$

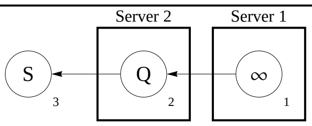
with F and g further specified as

$$\begin{bmatrix} 0 & 0 & 0 & -I & 0 \\ 0 & 0 & 0 & 0 & -I \\ -I & 0 & 0 & 0 & I \\ 0 & 0 & 0 & C_\infty & C_Q \end{bmatrix} \begin{bmatrix} X_Q(n) \\ X_D(n) \\ X_S(n) \\ U_\infty(n) \\ U_Q(n) \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ 0 \\ c \end{bmatrix}. \tag{9}$$

In addition we require that $U(n)$ is integer.

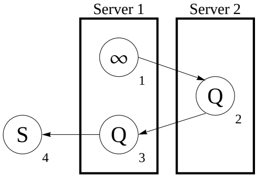
Table I contains four key examples along with their partitioned C and P matrices. Example 1 is the simplest model possible, a single server queue. Example 2 is a simple reentrant line. This example resembles the continuous-time reentrant line in [1]. Example 3 is the acquisition queue presented in Section II. Note the similarity between Examples 2 and 3: the first has a queue delay while the other has a fixed delay. Otherwise, these two networks are the same. Example 4 is somewhat similar to the push-pull network which was first introduced in [9] and further investigated in [8] and [17]. Some simulation experiments of Examples 2 and 4 are reported in Section V. Example 1 is used to illustrate the concept of multi-parametric programming for generating the MPC based control law in Section IV.

Example 1: single server queue



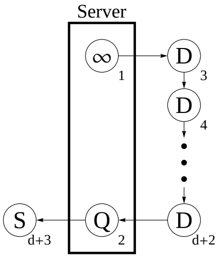
$$C = \left[\begin{array}{c|c} 1 & 0 \\ 0 & 1 \end{array}\right] \qquad P = \left[\begin{array}{c||c} 1 & 0 \\ \hline 0 & 1 \end{array}\right]$$

Example 2: simple reentrant line



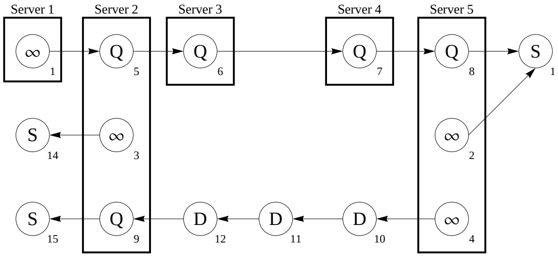
$$C = \left[\begin{array}{c|cc} 1 & 0 & 1 \\ 0 & 1 & 0 \end{array}\right] \qquad P = \left[\begin{array}{cc||c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right]$$

Example 3: acquisition queue



$$C = \left[\begin{array}{c|c} 1 & 1 \end{array}\right] \qquad P = \left[\begin{array}{c|cccccc|c} 0 & 1 & 0 & \cdots & \cdots & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & \cdots & \cdots & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & \cdots & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & & & \ddots & & & \vdots \\ 0 & 0 & & & & & 1 & 0 \\ 1 & 0 & 0 & \cdots & & & 0 & 0 \end{array}\right]$$

Example 4: more complex network



$$C = \left[\begin{array}{cccc|ccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}\right]$$

$$P = \left[\begin{array}{ccccc|ccc|ccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

TABLE I

SOME EXAMPLE NETWORKS AND THEIR PARTITIONED C AND P MATRICES.

The fluid relaxation and reference trajectories

For the MPC approach, it is useful to consider an auxiliary system that is a deterministic fluid relaxation of (8). For this, assume that the noise component does not exist and that the control effort (and thus the queue lengths) need not satisfy integrality constraints. The resulting system is then

$$\bar{X}(n+1) = A\bar{X}(n) + B\bar{U}(n) \tag{10}$$

subject to

$$F \left[\begin{array}{c} \bar{X}(n) \\ \bar{U}(n) \end{array} \right] \leq g, \tag{11}$$

with the matrices A, B, F and the vector g as defined above. For this system we define a *reference trajectory* as a pair of sequences $\{\bar{X}^r(n), \bar{U}^r(n)\}$ such that when the control $\bar{U}^r(\cdot)$ is applied to the fluid relaxation dynamics (10), the state $\bar{X}^r(\cdot)$ is obtained. In addition, the reference trajectory needs to adhere to the constraints (11). In general, the choice of the reference trajectory may depend on the desired throughput of the system. We discuss this briefly in Section V.

Given a reference trajectory, we define the *error dynamics* as

$$X^e(n) = X(n) - \bar{X}^r(n), \qquad U^e(n) = U(n) - \bar{U}^r(n). \tag{12}$$

Combining (12), (10) and (8) then yields

$$\begin{aligned} X^e(n+1) &= AX^e(n) + BU^e(n) + G\tilde{u}\Big(U^e(n) + \bar{U}^r(n)\Big) \\ &= AX^e(n) + BU^e(n) + \text{noise}. \end{aligned} \tag{13}$$

This representation proves useful in the next section.

IV. The MPC based approach

We now describe the MPC approach in more detail. The controller is parameterized by a discrete *time horizon* $N > 0$ and two positive definite matrices Q and R of dimensions $K_{\{Q,D,S\}}$ and $K_{\{\infty,Q\}}$, respectively. At time n, the controller uses the optimal solution of a quadratic programming (QP) problem in which the decision variables are the controls over the time horizon: $n, n+1, \ldots, n+N-1$. Given the current state $X(n)$ and the controls $U(i)$, $i = n, \ldots, n+N-1$, the prediction of the state over the time horizon, denoted by $\hat{X}(\cdot)$, is generated (details below) and appears in the objective and constraints of the QP. With $\hat{X}^e(i) = \hat{X}(i) - \bar{X}^r(i)$ denoting the prediction of the error dynamics, we get the following QP:

$$\min \sum_{i=n}^{n+N-1} \hat{X}^e(i+1)' Q \hat{X}^e(i+1) + U^e(i)' R U^e(i) \tag{14}$$

$$\text{s.t.} \qquad \begin{aligned} \sum_{k \in C(j)} U_k(i) &\leq c_i, \qquad i = n, \ldots, N-1, \quad j = 1, \ldots, L, && (15) \\ U_k(n) &\leq X_k(n), \quad k \in \mathcal{K}_Q, && (16) \\ U_k(i) &\leq \hat{X}_k(i), \quad i = n+1, \ldots, N-1, \quad k \in \mathcal{K}_Q, && (17) \\ 0 &\leq U(i), \qquad i = n, \ldots, N-1. && (18) \end{aligned}$$

The objective function in (14) quadratically penalizes deviations from the reference trajectory in both the state, using the Q matrix, and the control, using the R matrix. Constraints (15) are capacity constraints. Constraints (16) are for the control at time n not to exceed the current queue levels. Constraints (17) are the same, but with respect to the predicted queue levels. Finally, constraints (18) are non-negativity constraints.

There always exists a unique solution to the QP. To see this, first observe that there is always a feasible solution: $U(i) = 0, i = n, \ldots, n+N-1$. Then, since Q and R are positive definite, a unique solution is guaranteed (see [19] for background on quadratic programming). Note that the solution of the QP is an $N \cdot K_{\{\infty,Q\}}$-dimensional vector that specifies a control over the whole time horizon. From this solution, we only keep the first $K_{\{\infty,Q\}}$ coordinates to specify the control decision for the next time step. The remaining coordinates are not used. At every time point, the process is repeated: At time n, the current state is observed, the reference trajectory is calculated, and $X^e(n)$ is obtained. We denote it by X^e_0. Given a control over the time horizon, the prediction for $i > n$ is made by iterating the fluid relaxation dynamics (10). Thus $\hat{X}^e(i)$, which appears in the objective and constraints of the QP, is a function of X^e_0 and the decision variables $U^e(i)$, $i = n, \ldots, n+N-1$.

We denote the first step of the optimal solution as the $K_{\{\infty,Q\}}$-dimensional vector $U^e_{\text{OPT}}(X^e_0)$. Once $U^e_{\text{OPT}}(X^e_0)$ is determined, it is converted from the error dynamic coordinate system to the control coordinate system. We finally represent our controller as

$$U_{\text{MPC}}(n, X(n)) = \Big[\min\big(\max\big(\big[U^e_{\text{OPT}}\big(X(n) - X^r(n)\big) + U^r(n)\big]^{\$}, 0\big), X_Q(n)\big)\Big]^{\%}_{C}. \tag{19}$$

Here the $\max$ and $\min$ operators operate element-wise and $[x]^{\$}$ denotes the element-wise nearest integer to x. In addition, $[u]^{\%}_{C}$ denotes the following operation: Check the validity of $Cu \le c$. Every coordinate for which the inequality is invalid represents a server in which the capacity constraint has been exceeded. In this case reduce the u by integer steps until the inequality is met, by first reducing activities for the source buffers and then reducing from the queue buffers. Within each class one can follow some arbitrary specified rule.

Note that $U_{\text{MPC}}(\cdot)$ is a function of n since the reference trajectory needs to be evaluated so that the current error is plugged into $U^e_{\text{OPT}}(\cdot)$ and converted back into the coordinate system of the control. Further observe that if we take $U^r(n)$ to be constant then $U^e_{\text{OPT}}(\cdot)$ is only a function of the current error X^e_0, and does not depend on the current time.

Detailed description of the QP

We now give a detailed description of the QP. The decision variables of the QP are organized in the $N \cdot K_{\{\infty,Q\}}$-dimensional vector $\underline{\mathrm{U}}^e$. The $N \cdot K_{\{Q,D,S\}}$-dimensional vector $\hat{\underline{\mathrm{X}}}^e$ is the predicted state error over the time horizon $n+1,\dots,n+N$, obtained by iterating (13) and assuming no noise. The predicted error can be represented as

$$\hat{\underline{\mathrm{X}}}^e = \underline{\mathrm{A}} X^e_0 + \underline{\mathrm{B}}\,\underline{\mathrm{U}}^e,$$

where

$$\underline{\mathrm{A}} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix}, \quad \underline{\mathrm{B}} = \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & & \vdots \\ \vdots & & \ddots & \\ A^{N-1}B & & \cdots & B \end{bmatrix}.$$

We need some additional matrices. Denote by $\underline{\mathrm{Q}}$ and $\underline{\mathrm{R}}$ block diagonal matrices of Q and R with dimensions $N \cdot K_{\{Q,D,S\}}$ and $N \cdot K_{\{\infty,Q\}}$, respectively. Denote by $\underline{\mathrm{C}}$ a block diagonal matrix of C with dimension $N \cdot L \times N \cdot K_{\{\infty,Q\}}$, and let $\underline{\mathrm{c}}$ be a block vector that stacks the L-dimensional c vector N times. Define

$$S_{UQ} = \begin{bmatrix} 0 & I \end{bmatrix}, \quad S_{XQ} = \begin{bmatrix} I & 0 & 0 \end{bmatrix}.$$

The $K_Q \times K_{\{\infty,Q\}}$ matrix S_{UQ} is such that when multiplied by a control vector it selects the controls for the queue classes. Similarly, S_{XQ} can be multiplied by a state vector to select the state of the queue classes. In addition, define

$$\underline{\mathrm{S}}^1_{UQ} = \begin{bmatrix} S_{UQ} & 0 & \cdots & 0 \end{bmatrix}, \quad \underline{\mathrm{S}}^1_{XQ} = \begin{bmatrix} S_{XQ} & 0 & \cdots & 0 \end{bmatrix},$$

$$\underline{\mathrm{S}}^+_{UQ} = \begin{bmatrix} 0 & S_{UQ} & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ 0 & & & & S_{UQ} \end{bmatrix}, \quad \underline{\mathrm{S}}^-_{XQ} = \begin{bmatrix} S_{XQ} & 0 & \cdots & 0 \\ & \ddots & & \vdots \\ 0 & & S_{XQ} & 0 \end{bmatrix}.$$

Multiplying $\underline{\mathrm{S}}^1_{UQ}$ by the control vector over the whole time horizon results in the control of the queue classes at the first time step. Similarly, multiplying $\underline{\mathrm{S}}^1_{XQ}$ by the state vector results in the state of the queue classes at the first time step. These two matrices are used to implement constraints (16). The matrix $\underline{\mathrm{S}}^+_{UQ}$ is used to select the control of the queue classes at all time steps other than the first time step. The $(N-1)K_Q \times N \cdot K_{\{Q,D,S\}}$-dimensional matrix $\underline{\mathrm{S}}^-_{XQ}$ selects the queue levels at times $n+1,\dots,n+N-1$ when multiplied by the predicted state. Both $\underline{\mathrm{S}}^+_{UQ}$ and $\underline{\mathrm{S}}^-_{XQ}$ are used to implement constraints (17).

Given arbitrary reference trajectories, the QP is in general time-dependent. Nevertheless, the required coordinates are the controls (for both the source and queue classes) and the states (for only the queue classes). In many cases it is sensible to choose constant values for these references (yielding a linear reference trajectory for the sinks), so that the QP becomes time-independent. We denote by $\underline{\mathrm{U}}^r$ a stacked $N \cdot K_{\{\infty,Q\}}$-dimensional vector of the reference controls. Similarly, $\underline{\mathrm{X}}^r$ is a stacked $N \cdot K_{\{Q,D,S\}}$-dimensional vector of the reference states, of which only the coordinates relevant to $\mathcal{K}_Q$ are used in the QP.

The QP can now be formulated as follows:

$$\min_{\underline{\mathrm{U}}^e} \quad \underline{\mathrm{U}}^{e'}\left(\underline{\mathrm{B}}'\,\underline{\mathrm{Q}}\,\underline{\mathrm{B}}+\underline{\mathrm{R}}\right)\underline{\mathrm{U}}^e + 2X_0^{e'}\,\underline{\mathrm{A}}'\,\underline{\mathrm{Q}}\,\underline{\mathrm{B}}\,\underline{\mathrm{U}}^e + X_0^{e'}\underline{\mathrm{A}}'\,\underline{\mathrm{Q}}\,\underline{\mathrm{A}}X_0^e \tag{20}$$

s.t.

$$\begin{bmatrix} \underline{\mathrm{C}} \\ \underline{\mathrm{S}}^1_{UQ} \\ \underline{\mathrm{S}}^+_{UQ} - \underline{\mathrm{S}}^-_{XQ}\underline{\mathrm{B}} \\ -I \end{bmatrix} \underline{\mathrm{U}}^e \leq \begin{bmatrix} \underline{\mathrm{c}} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \underline{\mathrm{S}}^1_{XQ} \\ \underline{\mathrm{S}}^-_{XQ} \\ 0 \end{bmatrix} \underline{\mathrm{X}}^r + \begin{bmatrix} -\underline{\mathrm{C}} \\ -\underline{\mathrm{S}}^1_{UQ} \\ -\underline{\mathrm{S}}^+_{UQ} \\ I \end{bmatrix} \underline{\mathrm{U}}^r + \begin{bmatrix} 0 \\ S_{XQ} \\ \underline{\mathrm{S}}^-_{XQ}\underline{\mathrm{A}} \\ 0 \end{bmatrix} X_0^e.$$

It is straightforward to verify that (20) agrees with the QP in (14)-(18). Formulation (20) is in the form that is required by most commercial QP solvers.

In search of an explicit controller

When $\underline{\mathrm{U}}^r$ and $\underline{\mathrm{X}}^r$ are constant on their $k \in \mathcal{K}_Q$ coordinates, the QP only depends on X_0^e, and not on the current time. In such cases one might try to find an explicit solution or an approximation for the function $U^e_{\mathrm{OPT}}(\cdot)$. A useful tool in this respect is an algorithm for solving multi-parametric quadratic programs (MPQP) as described in [18] and implemented in [11].

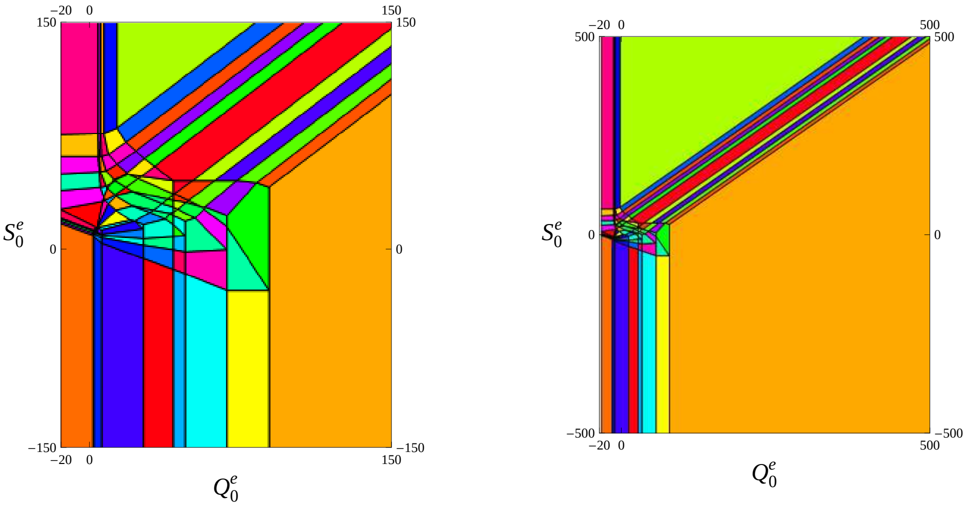


Fig. 5. Using MPQP for the single server queue example 1 with $c_1 = \infty$, $c_2 = 20$, $m_1 = 1$. $Q = I$, $R = I$, $N = 5$. The left figure shows a search space of a box of width 150 around the origin, in this area there is not much structure. The right figure shows the result of increasing the width of the search space to 500. In this case, the structure of the resulting controller is evident.

The output of the MPQP algorithm represents the optimal solution of (20) as a piece-wise affine function of X_0^e over a bounded *search space*. That is, the state space is partitioned into polytopes, and for each polytope an affine function of X_0^e is specified. Figure 5 illustrates the application of this algorithm for Example 1, the single server queue. Running this algorithm off-line is often useful for fast implementation of MPC based controllers (see [11]).

We believe that by using MPQP algorithms, it is also possible to understand the structure of the control law for points far from the origin for relatively simple models. Figure 5 stems hopeful: On the left, the search space is of width 150 – in this area there is no particular structure. On the right, the search space is of width 500 – structure is evidently present. Parameterizing this structure for simple examples by using numerical experiments such as these is currently being investigated. See [13] for a detailed example of applying MPQP for finding optimal controls.

V. Further simulation experiments

We now report some simulation experiments for Examples 2 and 4 (see Table I) and describe some of the observed phenomena.

Example 2: simple reentrant line

We consider the following parameters: $c_1 = 10, c_2 = 100$ and $m_1 = 20$. This implies that server 1 is the bottleneck. The maximal throughput achieved by a stable system is

$$m_1 \frac{c_1}{1+m_1} \approx 9.52,$$

which follows from similar considerations as for the acquisition queue – indeed these models are quite similar. In [1] (see also [20]) the authors have shown that a continuous-time reentrant line is stable when using the Last Buffer First Serve (LBFS) control that prioritizes step 3 over step 1, and a non-idling policy in server 2. This policy ensures that server 1 is maximally utilized.

Figure 6(a) plots the trajectory of the queue levels of this system under LBFS. Observe that server 2 is able to clear its queue almost instantaneously, while server 1 (which is 10 times as slow) is lagging behind. This cyclic behavior is somewhat similar to that of the acquisition queue operating under a threshold control (see Figure 4).

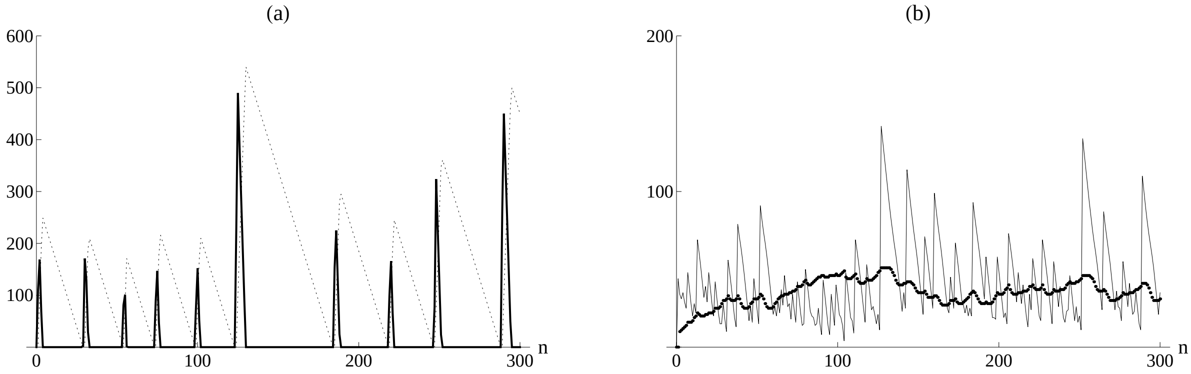


Fig. 6. The simple reentrant line model with $c_1 = 10, c_2 = 100$, $m_1 = 20$ and geometrically distributed input. Comparison of the LBFS policy (a) and MPC based controller (b). The MPC based controller uses $N = 10$, a Q matrix of all ones and $R = I$. Queue 1 is marked by a solid line. Queue 2 is marked by dots. Note that both controllers achieve maximal throughput, 9.52.

Figure 6(b) plots a simulation run of the same system when using MPC with $N = 10$, $R = I$ and Q the matrix with all elements 1. As expected, both the fluctuations and the mean queue sizes are smaller than for LBFS. For comparison: averaging over 20,000 time steps the average total queue level of the LBFS system is 172 and that of the MPC based controller is 112. No substantial difference in throughput is observed. This is a non-trivial observation, since it implies that the MPC based controller decides to

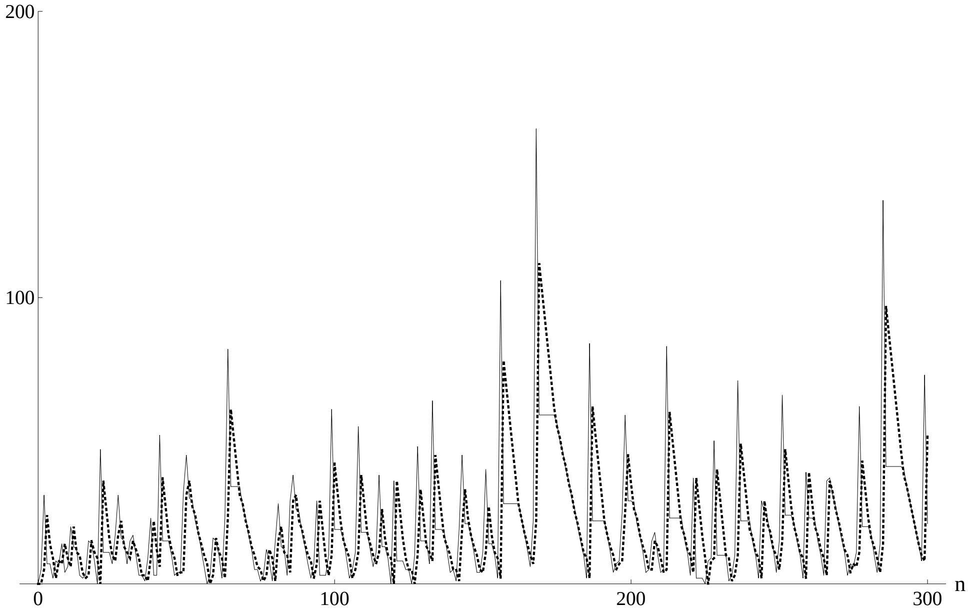


Fig. 7. The same model as in Figure 6 but with the identity Q matrix. In this case the controller does not produce at maximal throughput. The thin solid curve is queue 2 and the dashed curve is queue 3. Observe that both curves are very close to each other.

never idle the system. In general, it is not clear which models have the property that MPC preserves maximal throughput.

In Figure 7 we plot a simulation trace similar to that of Figure 6(b), but with $Q = I$. Notice the totally different behavior of the queue levels in this trace. It appears that queue 2 and queue 3 are almost perfectly synchronized. Here is a possible explanation: choosing a Q matrix of all ones implies minimization of $\left(\sum X_i\right)^2$, while an identity Q matrix implies minimization of $\sum {X_i}^2$. In the latter case, when queue 1 is large, the controller chooses not to remove jobs from queue 2 at a rate faster than queue 1 can be served. We note that similar "self-synchronizing" behavior also appears in Maximum Pressure Policies, cf. [4].

An additional observation (observed over a time horizon of 20,000) is that the maximal throughput of 9.52 was not achieved when $Q = I$. This means that server 1 occasionally idles. A possible remedy for this might be to convert some of the capacity inequality constraints in the QP to equality constraints, thus forcing server 1 (in this example) to work non-stop. It remains to be explored if the system would be stable under such a controller.

Example 4: more complex network

The main reason for presenting this more complex example is to emphasize the applicability of our method for arbitrary networks. We further use it to illustrate that reference trajectories can be obtained in a methodological manner. As for the other examples in this paper, we use a reference with constant queue levels and a linear increase of the sink buffers. Such a trajectory can always be found by solving a linear programming problem (LP), similar to the so-called Static Planning Problem (cf. [7]) used in multi-class queueing networks.

To find a reference trajectory, define the variables $r_i, i = 1, 2, 3, 4$ to be the long range flow rate on each of the routes. Here the route index corresponds to the source queue on which the route begins. For example, route 4 is the route passing through classes $4, 10, 11, 12, 9$ and ending in the sink buffer 15.

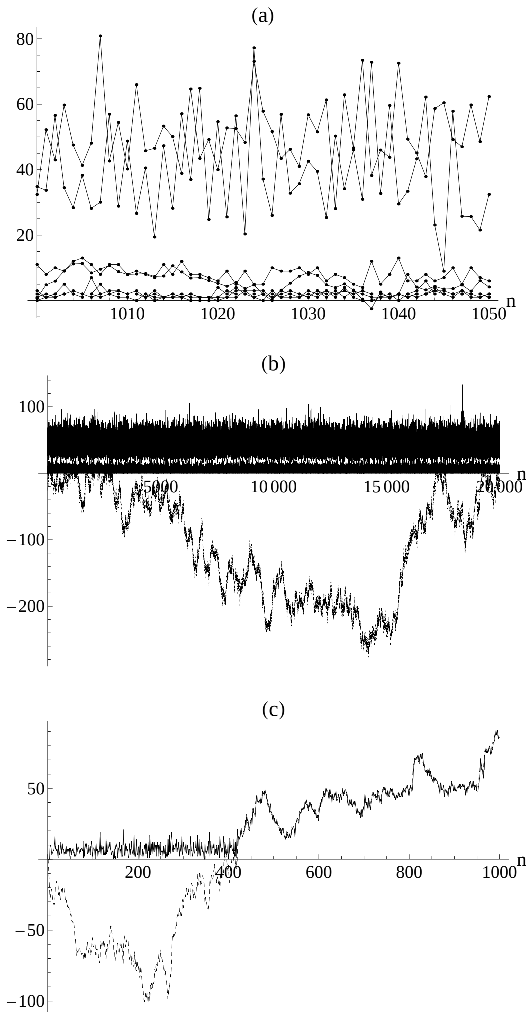


Fig. 8. Trajectories of the model of example 4 using an MPC based controller with $Q = I$, $R = I$ and $N = 3, 4, 5$. (a) $N = 5$. In this case the system appears stable over a longer time horizon, and we only plot the trajectory over the range 1000 to 1050. (b) $N = 4$. Queue levels are stable, but the sink error of 15 is not. (c) $N = 3$. The system is not stable. Only X^e_{15} and X_9 are plotted.

Now use the capacity constraints (6) to indicate restrictions on the variables r_i. For example, server 5 is required to serve routes 1, 2 and 4:

$$r_1 + \frac{r_2}{m_2} + \frac{r_4}{m_4} \leq c_5.$$

Note that in the above constraint, the source classes (2 and 4) are normalized by the means m_2 and m_4, while the queue class (class 8 on route 1) is not. Writing the constraints in this manner, together with the non-negativity constraints, defines a feasible polytope. Any point within this polytope is associated with a reference trajectory.

We find a point on the boundary of the polytope by solving an LP that minimizes $\sum_{i=1}^{4} w_i r_i$. In applications, the weights w_i reflect preference of routes. This is the resulting LP:

$$\max \quad \sum_{i=1}^{4} w_i r_i \tag{21}$$

s.t.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1/m_3 & 1 \\ 1 & 1/m_2 & 0 & 1/m_4 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} \leq \begin{bmatrix} c_1 m_1 \wedge c_3 \wedge c_4 \\ c_2 \\ c_5 \end{bmatrix},$$

$$r_i \geq 0, \quad i = 1,2,3,4.$$

Denote the solution by $r_i^*, i = 1,2,3,4$. Further, let ρ be an arbitrary number in the interval $[0,1]$. We now have the following reference trajectory:

(route 1) $\bar{U}_1^r(n) = \rho r_1^*/m_1, \quad \bar{X}_5^r(n) = \bar{X}_6^r(n) = \bar{X}_7^r(n) = \bar{X}_8^r(n) = \bar{U}_5^r(n) = \bar{U}_6^r(n) = \bar{U}_7^r(n) = \bar{U}_8^r(n) = \rho r_1^*,$
(route 2) $\bar{U}_2^r(n) = \rho r_2^*/m_2,$
(route 3) $\bar{U}_3^r(n) = \rho r_3^*/m_3,$
(route 4) $\bar{U}_4^r(n) = \rho r_4^*/m_4, \quad \bar{X}_{10}^r(n) = \bar{X}_{11}^r(n) = \bar{X}_{12}^r(n) = \bar{X}_9^r(n) = \bar{U}_9^r(n) = \rho r_4^*,$
(sinks) $\bar{X}_{13}^r(n) = \rho(r_1^* + r_2^*)n, \quad \bar{X}_{14}^r(n) = \rho r_3^* n, \quad \bar{X}_{15}^r(n) = \rho r_4^* n.$

When $\rho = 0$ the reference indicates “no job flow” and when $\rho = 1$ the reference indicates “maximal job flow” (i.e. the reference is on the boundary of the polytope).

For a numerical example consider the following parameters: $c_1 = 40$, $c_2 = 60$, $c_3 = c_4 = c_5 = 30$, $m_{15} = m_{13} = m_{12} = 1$, $m_{14} = 1.8$. Use the weights: $w_1 = 10$, $w_2 = 5$, $w_3 = 1$, $w_4 = 10$. The solution of the LP is

$$r_1^* = 1.9081, \; r_2^* = 38.1992, \; r_3^* = 51.2218, \; r_4^* = 6.8702.$$

Every $\rho \in [0,1]$ defines a reference trajectory. We simulate with $\rho = 1$. Figure 8 plots trajectories of X_k for $k \in \mathcal{K}_Q$ and X_k^e for $k \in \mathcal{K}_S$. The parameters of the MPC based controller are $Q = I$, $R = I$ and $N = 3,4,5$. We see that when $N = 5$ both the queue levels and the sink errors are stable over a long run of 20,000 time units. For $N = 4$ we observe that X_{15}^e is not stable. Finally, when $N = 3$, both X_{15}^e and X_9 seem unstable (we only plot two coordinates in (c)). Interestingly, the trajectory of X_9 “leaves the axis” at the same time when X_{15}^e hits the axis, and from that point onwards, the two trajectories closely follow each other. This behavior appears to be consistent over several runs.

The classification of models that are able to operate at $\rho = 1$ in a stable manner is an interesting open question. To our surprise we have seen that this is possible for Example 4. We have also observed (as expected) that changing the value of the time horizon produces different results and behaviors. An alternative option is to use $N = \infty$. While in this case, the QP (20) is not well defined, there are methods for implementing such an MPC based controller, cf. [6]. The basic idea is to partition the time horizon into a finite part and an infinite part, and to use the Ricatti Equation to solve the Linear Quadratic Regulator (LQR) problem for the infinite part. It is possible that in this case, the structure of the controller may be found for certain Q and R matrices. This remains the subject of future work.

VI. Conclusion

We have explored the applicability of MPC to stochastic queueing networks operating in discrete time. We have outlined a methodological way of constructing controllers for such networks. While our method appears to work well for some examples, explicit performance analysis of the behavior of the resulting stochastic processes remains an open question and is the subject of ongoing research. In this respect, one future research goal is to attempt to parallel the stability results that are currently known for deterministic settings [14] to our stochastic queueing network setting.

Of further interest is the study of queueing networks that generate their own arrivals. The current paper takes a step towards bridging the modeling gap between the acquisition queue (which is a special case of our network model), and queueing networks with infinite virtual queues [1], [16], [17]. A byproduct of our simulation results is the observation that the discrete time networks can often be operated on the boundary of the capacity region in a stable manner.

An additional research direction is to explore the use of tools such as MPQP (Figure 5) and LQR for deriving (or guessing) the structure of optimal solutions of the MPC based controller for simple examples such as the single server queue and the simple reentrant line.

Acknowledgment: This work was supported by the Netherlands Organization for Scientific Research (NWO-VIDI grant 639.072.072).

References

[1] I.J.B.F. Adan and G. Weiss. Analysis of a simple Markovian re-entrant line with infinite supply of work under the LBFS policy. *Queueing Systems*, 54(3):169–183, 2006.

[2] D.P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. I.* Athena Scientific, 2007.

[3] H. Chen and D.D. Yao. *Fundamentals of Queueing Networks: Performance, Asymptotics, and Optimization.* Springer, 2001.

[4] J. G. Dai and W. Lin. Maximum pressure policies in stochastic processing networks. *Operations Research*, 53(2):197–218, 2005.

[5] D. Denteneer, J.S.H. van Leeuwaarden, and I.J.B.F. Adan. The acquisition queue. *Queueing Systems*, 56(3):229–240, 2007.

[6] C.E. Garcia, D.M. Prett, and M. Morari. Model predictive control: theory and practice–a survey. *Automatica*, 25(3):335–348, 1989.

[7] J. M. Harrison. Brownian models of open processing networks: Canonical representation of workload. *Ann. Appl. Probab*, 10(1):75–103, 2000.

[8] A. Kopzon, Y. Nazarathy, and G. Weiss. A push pull system with infinite supply of work. *Preprint*, 2008.

[9] A. Kopzon and G. Weiss. A push pull queueing system. *Operations Research Letters*, 30(6):351–359, 2002.

[10] H.J. Kushner. *Heavy Traffic Analysis of Controlled Queueing and Communication Networks*. Springer, 2001.

[11] M. Kvasnica, P. Grieder, M. Baotic, and M. Morari. Multi-parametric toolbox (MPT). *Hybrid Systems: Computation and Control*, pages 121–124, 2004.

[12] M. Laumanns and E. Lefeber. Robust optimal control of material flows in demand-driven supply networks. *Physica A: Statistical Mechanics and its Applications*, 363(1):24–31, 2006.

[13] E. Lefeber, S. Lammer, and J.E. Rooda. Optimal control of a deterministic multiclass queueing system by serving several queues simultaneously. *SE Technical Report, Systems Engineering Group, The Department of Mechancial Engineering, Eindhoven University of Technology*, 2008-09, 2008.

[14] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O. Scokaert. Constrained model predictive control: stability and optimality. *Automatica*, 36:789–814, 2000.

[15] S.P. Meyn. *Control Techniques for Complex Networks*. Cambridge University Press, 2008.

[16] Y. Nazarathy. *On Control of Queueing Networks and the Asymptotic Variance Rate of Outputs*. PhD thesis, The University of Haifa, 2009.

[17] Y. Nazarathy and G. Weiss. Positive Harris recurrence and diffusion scale analysis of a push pull queueing network. *Performance Evaluation*, 67:201–217, 2010.

[18] P. Tøndel, T.A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489–497, 2003.

[19] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 2001.

[20] G. Weiss. Stability of a simple re-entrant line with infinite supply of work – the case of exponential processing times. *J. Oper. Res. Soc. Jpn.*, 47(4):304–313, 2004.