Dynamic Scheduling for Networked Control Systems

Indranil Saha
UC Berkeley and UPenn
indranil@seas.upenn.edu

Sanjoy Baruah
UNC Chapel Hill
baruah@cs.unc.edu

Rupak Majumdar
MPI- SWS and UCLA
rupak@mpi-sws.org

ABSTRACT

An integrated approach, embracing both control and scheduling theories, is proposed for implementing multiple control loops upon shared network and computational resources, where the network may additionally introduce delays and packet losses. Each control system is first analyzed from a control-theoretic perspective in order to determine the asymptotic rate at which control signals must be computed to maintain stability and optimal performance despite network losses. Since required completion rates for control tasks are asymptotic, and network packet drops uncertain, the problem of scheduling multiple such control tasks upon shared computational resources does not map to known problems in real-time scheduling. It is therefore formalized here as a new form of periodic task scheduling problem – one in which each task has an associated asymptotic completion rate requirement. Sufficient schedulability conditions are derived, and a dynamic scheduling algorithm designed, for solving such scheduling problems. This integrated methodology thus provides an effective way to incorporate network loss and delay constraints in the design of cyber-physical systems over integrated architectures. The use of this methodology is illustrated, and its efficacy demonstrated, upon an example system of five inverted pendulums.

1. INTRODUCTION

There is a trend in complex cyber-physical systems development toward integrated architectures, in which multiple real-time control loops are implemented on standardized and shared computing and communication platforms [12, 14]. While integrated architectures open up the possibility for modular design and cost-effective development of complex systems, they also lead to challenges in the design of real-time schedulers: sharing of the computational resources and communication medium may cause unwanted interferences between components, such as packet losses in the network or unschedulability in the processor.

In this paper, we describe a methodology for designing a dynamic scheduler for a group of control tasks in an integrated, networked system that is subject to communication and computation constraints. Our goal is to design a dynamic scheduler that schedules the tasks for individual controllers in such a manner that global asymptotic stability is ensured for each control system, and each controller optimally rejects disturbances (as measured by the $\mathcal{L}_\infty$ to RMS gain), in spite of network packet losses and shared computational resources. We model an integrated architecture as a collection of plants and discrete-time controllers, where the state of each plant is sampled by sensors and transmitted to the corresponding controller through a shared network. Each plant is modeled as a discrete-time linear time-invariant system. All the controllers execute upon a shared CPU. The network introduces a delay in the transmission of the state, and may additionally drop some packets. We assume that there is an upper bound on the rate of packet drop by the network, but no known deterministic mechanism for modeling the drop of individual packets.

A number of papers address the problem of maintaining stability in the presence of network induced delay (e.g. [15, 23]), or packet loss (e.g. [24, 9, 16]) or simultaneously delay and loss (e.g. [21, 7]). However, most of these results ignore the problem of *scheduling* the control tasks on a shared processor. The scheduling problem for a set of control systems sharing computation resources was first introduced by Branicky, Phillips, and Zhang [3], who studied the effect of "dropping" control computations in case all control tasks could not be scheduled. In order to ensure stability, they computed a minimal rate $r_{\min} \in [0, 1]$ at which control signals must be computed and sent to the plant. We call $r_{\min}$ the *minimum successful transmission rate*; $r_{\min} = 1$ means that the control signal is computed in every sampling step, $r_{\min} = 0$ means it is never computed, and $r_{\min} \in (0, 1)$ means that the control signal needs to be computed at least $r_{\min}$ fraction of steps in the long run. Majumdar et al. [11] showed how to determine the transmission rate $r_{\text{opt}}, r_{\text{opt}} \geq r_{\min}$, for each control system, so that each control system achieves optimal disturbance rejection performance. We call r_{opt} the *optimal successful transmission rate*. Now, a simple schedulability check that scales the resource requirement of each control task by r_{opt}, and a statically synthesized schedule [11] then ensure that all control loops are stable and achieve optimal performance.

In the presence of network packet losses, one might expect the above procedure to apply, when r_{opt} is replaced by $r_{\text{opt}} + r_{\text{net}}$, where r_{net} is the bound on the rate of drop of packets by the network. Unfortunately, there is a subtle problem that makes the above scheme inapplicable. In the schedulability check above, the scheduler can choose whether or not to compute a task, but once a decision is made, the corresponding computation is guaranteed to execute. In the case of network drops, the choice is not entirely to the scheduler: the scheduler might decide to compute a task, but the state of the plant required to compute the task may be dropped by the network. In other words, the scheme would require a clairvoyant scheduler that predicts when network drops will or will not happen in deciding to

schedule a task. Clearly, this is unrealistic.

In this paper, we formulate and solve the dynamic scheduling problem for sets of control systems sharing computation resources and a network that can drop packets. We make three contributions.

First, we extend the analysis of [3] and [11] to determine the rate at which control signals should be computed in order to achieve stability as well as optimal disturbance rejection performance. Majumdar et al. [11] showed that the problem of finding the lower bound on the disturbance rejection performance for a fixed computation rate is a convex optimization problem. By analyzing the characteristics of this optimization problem, we show in this paper that in the absence of any packet drop by the network, the rate r_{opr} at which a controller attains its best disturbance rejection capability is either $r_{\min}$ (the rate required to stabilize the plant) or $r_{\max}$ (the maximum rate at which the control task can be computed, constrained by the requirement that schedulability must be ensured for the multiple control tasks that are executing upon a shared processor). We call r_{opr} the *operating successful transmission rate*.

Second, we formulate the dynamic scheduling problem in the presence of network packet drops. We assume the packet drop is bounded by a rate r_{net}, but that there is no mechanism that predicts which packets may be dropped by the network. Since the control-theoretic requirements are *asymptotic*, our dynamic scheduling problem does not map to known scheduling problems from real-time scheduling theory. We formulate a novel periodic task scheduling problem, where there is an associated asymptotic completion rate requirement for each task.

Third, while optimal schedulability for our problem remains open, we derive sufficient conditions for schedulability and design a dynamic scheduler that maintains the rate of computation for each controller at its operating successful transmission rate r_{opr} despite packet drops in the network. We illustrate our dynamic scheduling approach on an example of five inverted pendulums implemented upon a shared processor and using a shared network.

We compare our dynamic scheduler with the static scheduler proposed in [11]. The static scheduler is synthesized for the rate $r_{\text{opr}} + r_{\text{net}}$, and is capable of maintaining the rate in a range $[r_{\text{opr}}, r_{\text{opr}} + r_{\text{net}}]$, by over-provisioning the computation of control tasks. Thus, while the schedulability test for the dynamic scheduling has to pass for r_{opr}, the schedulability test for the static scheduler has to pass for $r_{\text{opr}} + r_{\text{net}}$ for the control systems, rendering the static scheduling scheme unrealizable in many cases. The other shortcomings of the static scheduler are that its synthesis is computationally highly expensive, and its over-provisioning in control signal computation leads to unnecessary increase in control cost. We show that our dynamic scheduler provides similar performance to that of the static scheduler presented in [11], while decreasing control cost significantly avoiding the over-provisioning inherent in the static scheduling scheme.

2. NETWORKED CONTROL SYSTEMS

In this section, we describe a control theoretic formulation of the behavior of a networked, discrete-time, linear time-invariant control system with delay, in which the control signal may not be computed at every step. For simplicity, we assume one time unit delay between the sensing and control computation and actuation.

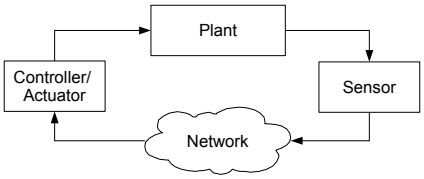


Figure 1: Linear control system with dropout

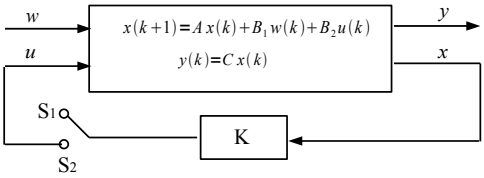


Figure 2: Linear control system with dropout

We assume the following architecture for the system, following [20, 21, 9]. The state of the plant is sensed at a regular interval and sent to the controller through a network (see Figure 1). We assume that the transmission of the plant's state through the network and the computation of the control signal takes less than one sampling period. We divide a sampling period into two sub-periods. At the end of the first sub-period, the state of the plant is available for control computation. In the second sub-period, the control signal is computed and the control signal is directly applied to the plant precisely at the end of the second sub-period (at the end of a period).

Due to network failure some packets from the sensor may be dropped and may never reach the controller. The controller itself may also decide not to compute the control signal in some rounds. The goal of the controller is to maintain a successful transmission rate of the control signal over a period of time so that the control system is *exponentially stable* and also has the desired performance in terms of *disturbance rejection*.

2.1 Scheduled Linear Control Systems

A linear time-invariant (LTI) control system in discrete time [1] is described by a difference equation:

$$\begin{aligned} x(k+1) &= Ax(k) + B_1 w(k) + B_2 u(k), \\ y(k) &= Cx(k), \end{aligned} \tag{1}$$

where $x(k) \in \mathbb{R}^n$ denotes the state of the system at the k^{th} time step, $w(k) \in \mathbb{R}^l$ denotes a disturbance signal at the k^{th} time step, $u(k) \in \mathbb{R}^m$ denotes a control input at the k^{th} time step, $y(k) \in \mathbb{R}^p$ denotes the outputs (the measurement of the states by the sensors) at the k^{th} time step, and A, B_1, B_2, and C are real-valued matrices of the appropriate dimensions. Figure 2 shows a discrete time LTI system with disturbance input w, control input u, and output y (ignore the switch for the moment). Such a discrete time system can be obtained from a continuous time system using a *sampling time* in the standard way [1]. The time steps $k = 0, 1, \ldots$ are multiples of the given sampling time.

We consider *state feedback controllers* of the form $u(k) = -Kx(k-1)$, where the matrix K denotes the *gain* of the controller. Note that the control input at the k'th time step is computed based on the state of the plant at the $(k-1)$'th time step. The delay models transmission delay from the

sensor to the controller, and the computation time of the controller. For simplicity, we assume that this one time unit delay is enough to accommodate the transmission time of the sensor data to the controller and the computation time of the controller. The aim of the controller is to ensure that the closed-loop system has certain properties, such as exponential stability, and a certain performance. Standard control-theoretic computations allow us to obtain state feedback controllers with the desired properties (see [1]).

We now describe *scheduled linear control systems*. The intuition behind this model is as follows. In each discrete time step $k = 1, 2, \ldots$, the current state $x(k)$ is observed, and the scheduler tries to schedule the control computation task if possible. If the control task is scheduled, the signal $u(k) = -Kx(k-1)$ is computed and applied to the actuators. In time steps k at which the control signal is not computed (i.e., in which the scheduler does not schedule the control task), the controller retains its previous value: $u(k) = u(k-1)$.

The scheduled control system is modeled as a networked control system with a loss of "data packets" between the controller and the plant, that is, the link between the controller and the plant is modeled as a channel with a switch (see Figure 2). In each time step, if the switch is closed (indicating that the scheduler ran the control task), the control signal is the updated control law, but if the switch is open (indicating that the scheduler could not run the control task), the control signal is unchanged from the previous control signal. Cycles in which the switch is open are said to incur *dropouts* of the control signal. We now relate the effect of dropouts on the performance of the control system.

In Figure 2, when the switch is closed (position S_1), the output of the controller is transmitted to the plant, and when the switch is open (position S_2), the output of the switch is held at the previous value. Similar to [3], the dynamics of the switch can be modeled formally as follows:

$$\text{When switch is in position } S_1: \quad u(k) = -Kx(k-1), \quad (2)$$

$$\text{When switch is in position } S_2: \quad u(k) = u(k-1). \quad (3)$$

Define the signal $s(k) = 1$ if the switch is in position S_1 at the k^{th} time step, and $s(k) = 2$ if the switch is in position S_2 at the k^{th} time step. These correspond to the control input being computed or not.

By choosing $X = [x^T, u^T]^T$ as the new state vector, the closed loop system with dropout, depicted in Figure 2, is given by:

$$\begin{aligned} X(k+1) &= \widetilde{A}_{s(k)}X(k) + \widetilde{B}_{1s(k)}w(k) \quad (4)\\ y(k) &= \widetilde{C}_{s(k)}X(k), \end{aligned}$$

where, using the dynamics of the switch in (2) and (3), we obtain

$$\widetilde{A}_1 = \begin{bmatrix} A & B_2 \\ -K & 0_{m\times m} \end{bmatrix}, \widetilde{B}_{11} = \begin{bmatrix} B_1 \\ 0_{m\times l} \end{bmatrix}, \widetilde{C}_1 = [C\ 0_{p\times m}];$$

and

$$\widetilde{A}_2 = \begin{bmatrix} A & B_2 \\ 0_{m\times n} & I_{m\times m} \end{bmatrix}, \widetilde{B}_{12} = \begin{bmatrix} B_1 \\ 0_{m\times l} \end{bmatrix}, \widetilde{C}_2 = [C\ 0_{p\times m}],$$

where $0_{m\times n}$ denotes the zero matrix in $\mathbb{R}^{m\times n}$ and $I_{m\times m}$ denotes the identity matrix in $\mathbb{R}^{m\times m}$.

Though we assume here that the transmission delay is less than one time period and the transmission delay and computation time can be accommodated in one time period, we can accommodate transmission delay spanning over multiple time periods by introducing new state variables to hold the old values of the states in equation (2)- (4). Once we update these equations to capture time delay for more than one time unit, all the results in the paper will follow seamlessly.

We define the successful transmission rate as the rate at which the switch in Figure 2 is in position S_1 [8]. Thus, the *successful transmission rate* r is given by

$$r = \lim_{L\to\infty} \frac{1}{L}\sum_{k=0}^{L}(2 - s(k)). \quad (5)$$

The dropout rate means the rate at which the switch in Figure 2 is in S_2. If the successful transmission rate for a control system is r, its dropout rate is $1 - r$. Clearly, in the former case, the scheduler runs the control task and updates the actuator, and in the latter case, it does not. The following theorem provides a sufficient condition on the successful transmission rate that guarantees the stability of the closed loop system.

THEOREM 1. *Consider the LTI control system in* (4). *Let* r *be the successful transmission rate. Assume that the closed loop system with no dropout and no disturbance is stable (i.e.,* $\max_i|\lambda_i(A - B_2K)| < 1$, *where* $\lambda_i(A - B_2K)$ *is the* i*-th eigenvalue of the matrix* $(A - B_2K)$. *Then the LTI control system with dropout in* (4), *with no disturbance, is exponentially stable for all* $r_{\min} < r \leq 1$, *where* $r_{\min} = \frac{log(\beta_2)}{log(\beta_2)-log(\beta_1)}$, $\beta_1 = \max_i|\lambda_i(\widetilde{A}_1)|^2$, *and* $\beta_2 = \max_i|\lambda_i(\widetilde{A}_2)|^2$, $\beta_1 < 1$ *and* $\beta_2 > \beta_1$.

PROOF. Following Theorem 6 in [24] it can be shown that for the LTI control system in (4), if there exists a Lyapunov function $V(x(kh)) = x^T(kh)Px(kh)$ and scalars α_1 and α_2 such that

$$\alpha_1^r\alpha_2^{1-r} > 1 \quad (6)$$

$$\widetilde{A}_1^T P\widetilde{A}_1 \leq \alpha_1^{-2}P \quad (7)$$

$$\widetilde{A}_2^T P\widetilde{A}_2 \leq \alpha_2^{-2}P \quad (8)$$

then the system is exponentially stable.

Let $\beta_i = \alpha_i^{-2}$ for $i = 1, 2$. From (7) and (8) we get that $\beta_1 = \max_i|\lambda_i(\widetilde{A}_1)|^2$ and $\beta_2 = \max_i|\lambda_i(\widetilde{A}_2)|^2$. Now by taking the log in (6) we get,

$$r > \frac{log(\beta_2)}{log(\beta_2) - log(\beta_1)}.$$

Since $\beta_1 < 1$ and $\beta_2 > \beta_1$, we have $r_{\min} < 1$. □

Note that if $\beta_2 < 1$, then $r_{\min} < 0$. If $\beta_2 < 1$, then the open loop system is stable, and a controller is not required for the stability of the system. For any unstable system, $\beta_2 > 1$ and $0 < r_{\min} < 1$.

2.2 Bound on $\mathcal{L}_\infty$ to RMS gain

We now recall the analysis of [11] relating dropouts to control performance. We take the $\mathcal{L}_\infty$ to RMS gain as our notion of performance. For the discrete-time LTI control system in (4), the $\mathcal{L}_\infty$ to RMS induced gain from w to y is defined as follows:

$$\sup_{\|w\|_\infty\neq 0, X(0)=0} \frac{\left(\limsup_{l\to\infty}\frac{1}{l}\sum_{j=0}^{l} y^T(j)y(j)\right)^{\frac{1}{2}}}{\|w\|_\infty}, \quad (9)$$

where $\|w\|_\infty := \sup\{\|w(k)\|_2,\ k \geq 0\}$, and $\|w(k)\|_2 = \sqrt{w^T(k)w(k)}$.

The $\mathcal{L}_\infty$ to RMS induced gain is a performance criterion showing the effect of the disturbance on the output of the plants [8]. Having smaller $\mathcal{L}_\infty$ to RMS induced gain implies better performance in the sense that the effect of disturbance on the output of the plant is smaller.

The following theorem identifies a relationship between the successful transmission rate and the upper bound of the $\mathcal{L}_\infty$ to RMS gain for the control system in (4).

THEOREM 2 ([11]). *Consider the discrete time LTI control system in* (4) *with the successful transmission rate* r. *The* $\mathcal{L}_\infty$ *to RMS gain is less than positive constant* γ *if there exists a piecewise continuous function* $V : \mathbb{R}^{n+m} \rightarrow \mathbb{R}_{\geq 0}$, *such that* $V(0) = 0$, *and* $\gamma_1, \gamma_2 \in \mathbb{R}$ *such that*

$$r\gamma_1^2 + (1-r)\gamma_2^2 < \gamma^2, \tag{10}$$

and

$$V\left(\widetilde{A}_i X + \widetilde{B}_{1i} w\right) - V(X) \leq \gamma_i^2 w^T w - y^T y, \text{ for } i = 1, 2. \tag{11}$$

Using $V(X) = X^T P X$, *where* P *is a symmetric positive definite matrix, the inequality (11) becomes an LMI as follows:*

$$\begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \leq 0. \tag{12}$$

Note that by choosing $V(X) = X^T P X$ and for a given successful transmission rate r, we can minimize γ, the upper bound of the $\mathcal{L}_\infty$ to RMS induced gain, by solving the following optimization problem:

$$\begin{aligned} &\text{minimize } \ r\gamma_1^2 + (1-r)\gamma_2^2 \\ &\text{subject to} \\ &\quad \begin{bmatrix} \widetilde{A}_i^T P \widetilde{A}_i - P + \widetilde{C}_i^T \widetilde{C}_i & \widetilde{A}_i^T P \widetilde{B}_{1i} \\ \widetilde{B}_{1i}^T P \widetilde{A}_i & \widetilde{B}_{1i}^T P \widetilde{B}_{1i} - \gamma_i^2 \end{bmatrix} \leq 0 \\ &\qquad\qquad\qquad\qquad \text{for } i = 1, 2 \\ &\quad \gamma_1, \gamma_2 \in \mathbb{R}, \\ &\quad P > 0 \end{aligned} \tag{13}$$

Although we only deal with linear control systems in this paper, our analysis can be extended to determine a bound on $\mathcal{L}_\infty$ to RMS gain for nonlinear systems. The challenge is to obtain a suitable Lyapunov function. If the plant and the controller can be represented as polynomials with respect to their arguments ($x(k)$, $w(k)$ and $u(k)$), then, by using SOS programming [13], it is possible to search for a suitable Lyapunov function to ensure a bound on the $\mathcal{L}_\infty$ to RMS gain for the nonlinear control system.

2.3 Finding the Operating Rate

We now consider the problem of choosing the successful transmission rate for a given controller to achieve the best performance. The successful transmission rate at which the best performance is achieved is called the *optimal successful transmission rate* and is denoted by r_{opt}. A lower bound on the rate for each system is given by Theorem 1: this is the rate $r_{\min}$ required to ensure stability. An upper bound $r_{\max}$ on the rate is decided by the scheduling constraints. We will discuss this issue in Section 3. If there is no scheduling constraint, $r_{\max} = 1$. Now, the following theorem provides the possible successful transmission rates that may achieve optimal performance.

THEOREM 3. *The* $\mathcal{L}_\infty$ *to RMS gain of the discrete time LTI control system in* (4) *attains the minimum value for the successful transmission rate to be either at* $r_{\min}$ *or at* $r_{\max}$.

PROOF. Let us assume that the values of γ_1 and γ_2 for which the $\mathcal{L}_\infty$ to RMS gain attains the minimum value are $\gamma_{1\text{opt}}$ and $\gamma_{2\text{opt}}$. Now there may be three cases:

Case 1: $\gamma_{1\text{opt}} < \gamma_{2\text{opt}}$. Note that the LMIs in the constraints (13) in the optimization problem in (13) do not depend on r. Thus a solution for γ_1 and γ_2 is valid for any successful transmission rate. In this case, if we decrease the value of r for the same values of γ_1 and γ_2, the value of γ also decreases with r. In this case, the minimum value of γ is obtained at $r_{\min}$.

Case 2: $\gamma_{1\text{opt}} > \gamma_{2\text{opt}}$. By using similar argument as Case 1, we can show that the $\mathcal{L}_\infty$ to RMS gain attains the minimal value at $r_{\max}$.

Case 3: $\gamma_{1\text{opt}} = \gamma_{2\text{opt}}$. In this case, γ becomes equal to γ_2 and thus remains constant for any successful transmission rate. Thus $r_{\max}$ and $r_{\min}$ both gives the optimal value for the $\mathcal{L}_\infty$ to RMS gain. □

In [11], numerical solutions to convex programming problems were used to compute the performance for different successful transmission rates and the optimum performance was found to vary arbitrarily in the interval between $r_{\min}$ and $r_{\max}$. Theorem 3 provides a precise characterization of the optimum at one of the end points of the interval. Our hypothesis is that the performance profile computed in [11] suffers from numerical instabilities in the solvers used to solve the convex optimization problems in computing the bound on the performance for different successful transmission rates.

As shown in [11], a static scheduler can be synthesized to maintain the computation rate of the control systems to their optimal successful transmission rate. However, if the network drops packet with a rate r_{net}, then the scheduler can maintain the rate only in an interval $[r_{\text{opr}} - r_{\text{net}}, r_{\text{opr}}]$, where r_{opr} is called the *operating successful transmission rate* of the control system. The following theorem provides the candidate values for the operating successful transmission rates for the control systems for synthesizing a static scheduler.

THEOREM 4. *The operating successful transmission rate for a static scheduler is either* $r_{\max}$ *or* $r_{\min} + r_{\text{net}}$.

PROOF. If $r > r_{\max}$, then scheduling constraints will be violated. If $r < r_{\min} + r_{\text{net}}$, then due to packet drop in the network, the successful transmission rate may be less than $r_{\min}$, and the system may be unstable. If we choose any other rate r, $r_{\min} + r_{\text{net}} < r < r_{\max}$, we can use the reasoning of Theorem 3 and show that by shifting the operating rate either towards $r_{\max}$ or towards $r_{\min}$, it is possible to decrease the average value of the $\mathcal{L}_\infty$ to RMS gain in the operating region. □

2.4 Example

As a motivating example, we use the model of the inverted pendulum from [22]. The state-space representation of an inverted pendulum is given by:

$$\begin{aligned} \dot{x} &= Ax + B_1 w + B_2 u; \\ y &= Cx, \end{aligned}$$

where

$$A = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & \frac{\rho}{ml^2} \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ \frac{1}{ml} \end{bmatrix}, \qquad (14)$$
$$B_1 = \begin{bmatrix} 0.1 \\ 0 \end{bmatrix}, \quad C = [0.001,\ 0].$$

In this model, $x = [x_1, x_2]^T$ is the state of the system, with x_1 the angular position and x_2 the angular velocity of the point mass, m is the mass, l is the length of the rod, $g = 9.8\text{m/s}^2$ is acceleration due to gravity, ρ is the rotational friction coefficient, u is the applied force (control input), and w is the disturbance input.

Let us consider an instance of the above system where $\rho = 0.6$, $m = 0.4$ and $l = 0.6$ (all values are in S.I. units). We discretize the plant in (14) with sampling period of $20ms$. A stabilizing controller for the discretized plant is given by $K_1 = [4.8462\ 0.1800]$. The minimum successful transmission rate for this system to ensure stability is $r_{\min} = 0.6623$.

Now we plot how the upper bound on the $\mathcal{L}_\infty$ to RMS gain varies with the successful transmission rate between $r_{\min}$ and $r_{\max} = 1$. The figure is obtained by quantizing the successful transmission rates between $r_{\min}$ and $r_{\max}$ with a quantization factor 0.01, and then solving the optimization problem in (13) for each choice of the successful transmission rate. For a given controller, we refer to the plot of transmission rate vs. performance as the *performance profile* of the controller. The curve in Figure 3 shows the performance profile for the controller K_1.

Note that we use the upper bound on the $\mathcal{L}_\infty$ to RMS gain instead of the $\mathcal{L}_\infty$ to RMS gain for a particular disturbance pattern while constructing the performance profile. Though different successful transmission rates may be the best for different disturbance patterns, we seek to determine a successful transmission rate for which we can guarantee the minimum bound on the $\mathcal{L}_\infty$ to RMS gain, even if this successful transmission rate may not be the best for all possible disturbance patterns.

Now we come to the problem of choosing the operating successful transmission rate. Due to scheduling constraints, the value of $r_{\max}$ may be less than 1. The value of r_{opr} depends on the value of $r_{\max}$. In our present example, if $r_{\max} \leq 0.87$ then $\gamma_m(r_{\min} + r_{\text{net}}) \leq \gamma_m(r_{\max})$. In this case, the choice of operating successful transmission rate is $r_{\min} + r_{\text{net}}$, as in that case choosing $r_{\min} + r_{\text{net}}$ would give the optimal performance and optimal CPU time usage. If $r_{\max} > 0.87$ then $\gamma_m(r_{\max}) < \gamma_m(r_{\min} + r_{\text{net}})$. If $r_{\max}$ is permitted by the scheduling constraints to be greater than 0.87, $r_{\max}$ may be chosen as the operating successful transmission rate in order to get better performance. This example thus illustrates the effect that the scheduling constraints have on the choice of the operating successful transmission rate.

3. SCHEDULABILITY ANALYSIS IN THE PRESENCE OF PACKET DROPOUT

In this section, we introduce the effect of multiple control loops sharing the same CPU and the network. Since the CPU is shared, we have to schedule the execution of the controllers and ensure that each controller achieves an optimal successful transmission rate in the presence of scheduling constraints and network losses.

Let n denote the number of control loops. Let h_i denote the sampling period of the i'th control loop. For the i'th

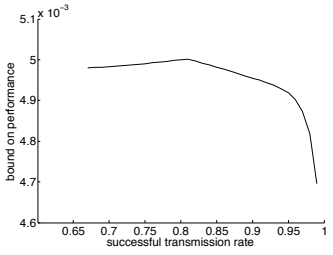


Figure 3: The upper bound of the $\mathcal{L}_\infty$ to RMS gain vs successful transmission rate for an inverted pendulum for $K_1 = [4.8462\ 0.1800]$

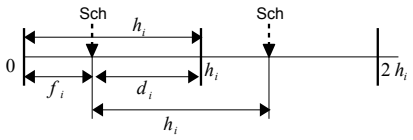


Figure 4: Periodic state transmission and control computation

control loop, the state of the plant is sampled at the instants $0, h_i, 2h_i, \ldots$, as illustrated in Figure 4. The period h_i is divided into two sub-periods f_i and d_i. In the first f_i time duration, the state of the plant is transmitted to the controller. At the end of the first sub-period f_i, the state of the plant is available for the computation of the control signal. At this moment the scheduler is invoked (denoted by Sch in Figure 4) to decide whether and how to schedule the computation of the control signal. If the scheduler decides to schedule the control computation, the control signal is computed during the second sub-period d_i, and is applied to the plant at the end of the sampling period. Let us refer to this computation of the control signal as a *job*. Note that these jobs arrive periodically with a period h_i, and each job has a deadline that occurs d_i time units after it is ready for execution. The execution time of the scheduler is considered to be negligible, however control signal computation requires some time. Though the computation in Equation 2 looks simple, the practical implementation of such controller requires some amount of sensor data processing [19] that contributes to the worst-case execution time of control signal computation. Let c_i denote the worst case computation time for the control signal of the i'th control system – c_i is thus the worst-case execution time (WCET) of each job of the i'th control system. We denote by r_i the successful transmission rate of the control signal to the plant for the i'th control system.

3.1 Computing Message Transmission Times

In order to determine the value of f_i for the i'th control system, we determine the worst case delivery time of the message from the sensor to the controller. The computation of worst case message delivery time depends on the nature of the protocol used in the transmission of message. We do not address the general problem in this paper, but use known results about the worst case message delivery time for the CAN protocol [17, 5]. In our experiments, we have used the CAN protocol to transmit messages from a sensor to a controller, and have used the recurrence relation in [5] to compute the worst case message delivery time. Note that the f_i's for different control systems may be different.

3.2 Schedulability of Control Computations

The jobs for the i'th control system arrive at the time instants $f_i, f_i + h_i, f_i + 2h_i, \ldots$. Given h_i, f_i, d_i, c_i and r_i for each control system, we seek to determine whether it is feasible to schedule the control computations on a single processor. Theorem 5 addresses this feasibility question. It follows from [2, (Lemmas 3.4 and 3.5)].

THEOREM 5. *A necessary and sufficient condition for n control systems to be feasible upon a shared preemptive processor is that*

$$\forall t_1, t_2 : t_1 < t_2 : \sum_{i=1}^{n} \eta_i(t_1, t_2)\, c_i \leq (t_2 - t_1)$$

where $\eta_i(t_1, t_2)$ denotes the number of jobs of the i'th control system that are scheduled for execution, that become available at some time $\geq t_1$ and have deadline $\leq t_2$.

However, it is known [2] that checking the condition of Theorem 5 is highly intractable – co-NP hard in the strong sense – even for the simple case where $r_i = 1$ for all i and there are no network losses ($r_{\text{net}} = 0$). We will therefore focus our attention on devising a *sufficient*, rather than exact, feasibility test. We start out adapting the notion of **demand bound function (dbf)** [2] to our specific situation: for any positive real number t,

$$\text{dbf}_i(t) = \max_{t'} \left(\eta_i(t', t' + t)\, c_i\right) \tag{15}$$

I.e., $\text{dbf}_i(t)$ denotes a tight upper bound on the cumulative execution requirement of computations of the i'th control system that are scheduled for execution with availability times and deadlines within some interval of duration t. Computing $\text{dbf}_i(t)$ for control tasks in the presence of network error turns out to be a very challenging problem; we defer a discussion on how we solve this problem to Section 5.

Applying the notion of demand bound function to Theorem 5 immediately yields the following sufficient feasibility condition for our collection of n control systems:

$$\forall t : t > 0 : \sum_{i=1}^{n} \text{dbf}_i(t) \leq t \tag{16}$$

The following theorem asserts that under suitable constraints, Condition 16 can be validated quite efficiently. The proof is similar to [2, Theorem 3.1].

THEOREM 6. *Let κ be a fixed constant, $0 < \kappa < 1$, such that $\sum_{i=1}^{n} r_i \frac{c_i}{h_i} \leq \kappa$. If Condition 16 is violated then it will be violated for some t that is $O(n\max\{h_i - d_i\})$, and is equal to $(\ell h_i + d_i)$ for some integer $\ell \geq 0$ and some $i, 1 \leq i \leq n$.*

We have designed an algorithm, `testFeasibility`, for determining the feasibility of collections of control systems satisfying the condition of Theorem 6. Algorithm `testFeasibility` takes as input vectors $\mathbf{h}$, $\mathbf{d}$, $\mathbf{c}$, and $\mathbf{r}$ for the control systems, and validates Condition 16 for each of pseudo-polynomially many different values of t at which the condition may be violated according to Theorem 6. We will see in Section 4.1 that each $\text{dbf}_i(t)$ can be determined in constant time; hence, Algorithm `testFeasibility` has pseudo-polynomial run-time.

3.3 Computation of Maximum Successful Transmission Rates

Algorithm 3.1: Computation of Maximum Successful Transmission Rates

```
1  function findMaximumRates(h, d, c)
2  begin
3      r := r_opt
4      while r ≥ r_min do
5          result := testFeasibility(h, d, c, r)
6          if result = feasible then
7              return r
8          end
9          for i = 1 ... n do
10             if r(i) > r_min(i) then
11                 r(i) := r(i) − ε
12             end
13         end
14     end
15     return "not feasible"
16 end
```

Algorithm 3.1 shows how to compute an upper bound on the rate of successful transmission for all control systems so that the tasks are schedulable. It takes as input the vectors $\mathbf{h}$, $\mathbf{d}$ and $\mathbf{c}$ representing the period, deadline, and computation time of the control tasks for all the control systems, respectively. The symbol $\mathbf{r}$ denotes the vector representing the successful transmission rates of the control systems. Initially, the components of $\mathbf{r}$ are set to $\mathbf{r_{opt}}$, where $\mathbf{r_{opt}}$ is a vector capturing the optimal successful transmission rate for the individual control systems when no scheduling constraint is present. The elements of $\mathbf{r_{opt}}$ are thus either minimum successful transmission rate for the corresponding control system, or 1. The algorithm runs in a loop. At each step, it calls Algorithm `testFeasibility` to check whether the control tasks with the rate vector $\mathbf{r}$ is feasible. If yes, vector $\mathbf{r}$ is returned as the vector containing the maximum successful transmission rates. Otherwise, if an element of $\mathbf{r}$ is greater than the minimum successful transmission rate of the corresponding control system, then the element is decremented by a constant ϵ. The loop continues to run when any element of $\mathbf{r}$ is greater than the minimum successful transmission rate of the corresponding control system.

4. SCHEDULER SYNTHESIS

In this section, we describe a dynamic scheduling strategy for the control tasks on a shared processor, so that the transmission rates for the control systems are maintained at their corresponding operating successful transmission rates. The requirement that a fraction r_i of the jobs of the i'th controller complete by their deadlines is an *asymptotic* one — it simply asserts that as the number ν of jobs generated by the controller approaches ∞, the number of these jobs that complete execution by their deadlines equals (or exceeds) $\nu \times r_i$. Hence if $r_i = 0.5$, for example, it is perfectly acceptable, according to this definition, to execute every other job; or to execute the first ten jobs and then skip the next ten; or to skip the first one hundred jobs and execute the next one hundred; etc.

While such asymptotic requirements on completion ratio arise naturally in control theory, they lead to scheduling problems that is not easily solved using current techniques from real-time scheduling theory. (In particular, we were unable to map such an asymptotic completion rate requirement into any of the known models for scheduling *soft* or

firm real-time systems.) Our scheduling strategy, presented in pseudo-code form in Algorithm 4.1, is more restrictive than mandated by such an asymptotic completion-ratio requirement: we seek to have the rate of executed control computations approach the specified operating rate r_i from below, while ensuring that it never exceeds r_i. (Such a strategy in essence does away with the "asymptotic" nature of the completion-ratio requirement, choosing instead to satisfy it as soon as possible while simultaneously minimizing the amount of execution that must be performed.) If a collection of control systems is deemed schedulable by Algorithm 4.1 (we discuss our schedulability test in Section 4.1 below), then it is guaranteed that the rates of control computations eventually reach $\mathbf{r_{opr}}$ and stay there if there is no packet drop by the network (here, $\mathbf{r_{opr}}$ is a vector of the individual desired successful completion rates – the r_i's). As we show in Section 4.1, under certain assumption on the occurrence of packet drop in the network, the schedulability analysis can be performed precisely. If the control tasks are schedulable, the scheduler can maintain the rate of control computation at $\mathbf{r_{opr}}$ even in the presence of packet drops.

In the pseudo-code, $\mathbf{r}(i)$ and $\mathbf{m}(i)$ denote the current successful transmission rate and the number of periods that have occurred thus far for the i'th control system. The scheduler runs in an infinite loop. Upon receiving the state of a plant for the control computation, the scheduler first checks whether completing the job would make the rate of scheduled jobs go above the operating rate $\mathbf{r_{opr}}(i)$. If not, then the control computation is scheduled based on earliest deadline first (EDF) strategy.

As pointed out above, Algorithm 4.1 makes no attempt to exploit the asymptotic nature of the completion-ratio requirements, and is therefore not an optimal algorithm for scheduling the control systems. Despite its sub-optimality, Algorithm 4.1 is clearly *correct* – any system that it does schedule in a manner that meets all the deadlines of all the jobs that are selected for execution, does indeed satisfy the properties desired by the control application (provided the network failure rate r_{net} is small enough to allow this to happen). In what follows, we present a schedulability analysis of Algorithm 4.1. Specifically, we will show that under the assumption $\mathbf{r_{opr}}(i) \leq 1 - r_{\text{net}}$, we can provide precise schedulability analysis for Algorithm 4.1.

Algorithm 4.1: Dynamic Scheduler

```
1  function scheduleControlComputation(r_opr)
2  r_opr is a vector: r_opr(i) is the desired successful completion
   rate for the i'th control system
3  begin
4      for i = 1...n do
5          r(i) = 0, m(i) = 0
6      end
7      time := 0
8      while true do
9          for i = 1...n do
10             if (time − f_i)%h_i = 0 then
11                 success_new_rate := (r(i)*m(i)+1)/(m(i)+1)
12                 failure_new_rate := (r(i)*m(i))/(m(i)+1)
13                 if The state of the plant is received then
14                     if success_new_rate ≤ r_opr(i) then
15                         Schedule based on EDF strategy
16                         r(i) := success_new_rate
17                     else
18                         // Scheduler drops the computation
19                         r(i) := failure_new_rate
20                     end
21                 else
22                     // Packet drop by network
23                     r(i) := failure_new_rate
24                 end
25                 m(i) := m(i) + 1
26             end
27         end
28         time := time + 1
29     end
30 end
```

4.1 Schedulability analysis of Algorithm 4.1

We start out studying the kinds of processor loads that are generated by Algorithm 4.1 when scheduling jobs of a particular control system. So let us consider a particular control system i, with parameters (h_i, d_i, c_i, r_i). It is evident that the computational demand by jobs of the i'th control system over an interval of duration t is maximized if a job that is selected for execution is released at the very start of the interval. In this case, the number of jobs that are released, and have their deadlines, within the interval of duration t is given by $(\lfloor \frac{t-d_i}{h_i} \rfloor + 1)$. However, not all these jobs may be selected for execution by Algorithm 4.1 – for any non-negative integer n, let $D_i(n)$ denote the *largest* number of jobs of the i'th control system that Algorithm 4.1 will select for execution, out of any sequence of n consecutive jobs. Given the function $D_i(n)$, we can easily compute the demand bound function for the task: for any $t > 0$:

$$\text{dbf}_i(t) = c_i \cdot D_i(\lfloor \frac{t - d_i}{h_i} \rfloor + 1) \tag{17}$$

Hence to compute the demand bound function $\text{dbf}_i(\cdot)$, it suffices to determine the function $D_i(n)$. We now discuss how $D_i(n)$ may be determined. In what follows, we refer a packet drop by the network as a *network fault.*

Job execution sequences. In the absence of network errors, Algorithm 4.1 selects jobs for execution according to a deterministic pattern. If r_i is a *rational* number[1] then this pattern is cyclic: if r_i is equal to a/b where a and b are integers and $\gcd(a, b) = 1$, the pattern will be of length b, out of which a jobs will be executed. E.g., if $r_i = 0.6 = 3/5$, the pattern is $(NYNYY)^\infty$, where a "N" denotes that a job is not executed; a "Y," that it is.

Note that computing $D_i(n)$ for a given n is equivalent to determining some job execution sequence of length n containing the largest number of "Y"'s (i.e., that correspond to jobs that are actually executed by Algorithm 4.1). We may restrict our search to job execution sequences beginning with a "Y". (To see why this is so, consider some sequence of length n that begins with a "N". The sequence of length n obtained by shifting rightwards by one job would yield a sequence of length n in which the number of "Y"'s does not decrease, and may in fact *in*crease if the new job added at the end of the pattern were a "Y".) Henceforth, therefore, we restrict our attention to sequence that begin with a "Y."

No faults. Let $D_i^{(o)}(n)$ denote the largest number of jobs that Algorithm 4.1 will end up executing for the i-th control system out of any sequence of n consecutive jobs, in the *absence* of any network fault. $D_i^{(o)}(n)$ is easily determined

[1]We do not deal with non-rational r_i's.

by inspection of the cyclic pattern of execution. Letting $r_i = a/b$ (as above), the general expression is

$$D_i^{(o)}(n) = a \times \lfloor \frac{n}{b} \rfloor + \left(a - \left\lfloor \frac{a}{b} \times (b - n \bmod b) \right\rfloor\right) \quad (18)$$

Here, the first term on the right-hand side represents the fact that a jobs are executed out of every consecutive b jobs, and the second term yields the amount of jobs executed from amongst those remaining after every group of b consecutive jobs is thus accounted for. Let us illustrate by revisiting our example of the control task i with $r_i = 3/5$. It is not difficult to show that

$$D_i^{(o)}(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2, & \text{if } n = 2 \text{ or } 3 \\ 3, & \text{if } n = 4 \text{ or } 5 \\ 3 \times \lfloor n/5 \rfloor + D_o(n \bmod 5) & \text{if } n >= 5 \end{cases}$$

A single network fault. Now, suppose that a ***single*** network fault, impacting upon Algorithm 4.1's ability to execute a single job of the i'th control system, were to occur. If this fault occurs during a job that Algorithm 4.1 would not have chosen for execution, then it has no effect on the computational demand. However, a fault during a job that would otherwise (i.e., in the absence of the fault) have been selected for execution prevents that job from getting executed; i.e., Algorithm 4.1 would not execute this job and instead seek to execute the *next* job that would not have been chosen for execution had the fault not occurred. Letting "Y" ("N," respectively) denote a job that would have been selected (would not have been selected, resp.) for execution in the absence of a fault, the fault on the "Y" effectively results in that "Y" becoming a "N,", and causes the next "N" to become a "Y".

As we had argued above, in determining $D_i(n)$ we may restrict our attention to sequences that begin with a "Y." Upon all sequences that have been hit with a single fault, note that if the sequence does not *begin* with the new "Y" resulting from the fault, the worst-case effect of this change is absorbed: a "YN" has simply become a "NY," and the total number of "Y"s in the sequence has not increased. Hence, we only need to consider sequences that begin with the new "Y." To do so, we perform the following steps.

1. We separately consider the possibility that the single fault occurred in each of the a "Y"'s in the cyclic pattern of length b characterizing the job execution sequences of the i'th control task
2. For each possibility so considered, we identify and encapsulate in an equation, the largest number of "Y"'s in any n consecutive jobs; once again, such an equation is easily determined by inspection of the pattern.
3. The desired value for $D_i(n)$ is then obtained by simply taking the maximum of the numbers of "Y"'s determined by each such individual equation.

We illustrate on our earlier example of $r_{\text{opr}} = 0.6$ for which we had identified the cyclic pattern $(NYNYY)^\infty$. If the single fault had occurred during the first "Y" in this pattern, the resulting sequence would have been of the form

$$\not{\mathbf{Y}}\mathbf{Y}YY(NYNYY)^\infty,$$

with the $\not{\mathbf{Y}}$ denoting the job that could not be executed due to the fault, and the following (bold-font) $\mathbf{Y}$ one denoting the previous "N" that has now become a "Y." Letting $D_i^{(1)}(n)$ denote the number of Y's from amongst the first n symbols for this sequence, we have

$$D_i^{(1)}(n) = \begin{cases} n, & \text{if } n \leq 3 \\ 3, & \text{if } n = 4 \\ 4, & \text{if } n = 5 \text{ or } 6 \\ 5, & \text{if } n = 7 \\ 3 \times \lfloor \frac{n-3}{5} \rfloor + D_i'(n - \lfloor \frac{n-3}{5} \rfloor \times 5) & \text{if } n \geq 8 \end{cases}$$

If the single fault had instead occurred during the *second* "Y" in the cyclic pattern $(NYNYY)^\infty$, the resulting sequence would have been of the form

$$\not{\mathbf{Y}}Y\mathbf{Y}YNYY(NYNYY)^\infty$$

This sequence can also be analyzed in a manner similar to the way we had analyzed the sequence above. And finally, we would also need to consider the possibility that the fault had occurred in the third "Y" in the pattern.

Multiple faults. Another disconnect between the worlds of control theory and real-time scheduling is highlighted in the consideration of the network error rate r_{net}. In the control-theoretic specification of the problem, this rate too is an asymptotic one which allows for the possibility that an arbitrarily long time interval with no faults will be followed by a very long one with multiple repeated faults. However, it is obvious that Algorithm 4.1 cannot make any non-trivial schedulability guarantees under such a fault model: we *must* make assumptions bounding the absolute number of faults that will occur within some specified duration in order to be able to guarantee schedulability by Algorithm 4.1. Such assumptions will limit the number of distinct jobs that may be affected by faults, within any specified window of successive jobs. We make the following assumption on the fault model. If the operating rate for a control system is given by $r = \frac{a}{b}$, then no more than $b - a$ faults can occur out of any b successive jobs. Given a bound on the rate of packet drop by the network to be r_{net}, this entails that the operating rate cannot be more than $1 - r_{\text{net}}$ for the schedulability test to hold. Our fault assumption ensures that the operating rate does not deviate from the desired rate for more than b cycles, as the scheduler has sufficient empty slots to compensate for the network faults.

It is easily seen that faults after the first sequence of contiguous faults (i.e., faults that effect consecutive jobs of a particular control system) cannot increase the $D_i(n)$ function, and consequently the demand bound function $\text{dbf}_i(\cdot)$. To see why, consider all the jobs that arrive in, and have deadlines within, some time interval. If a job that is not at the beginning of this interval and that would have been selected for execution in the absence of a fault is hit with a fault, it will not be selected for execution by Algorithm 4.1; a subsequent job, which would not have been selected for execution had the fault not occurred, will be executed instead. If both these jobs are within the interval, then the cumulative demand over the interval (as measured by the dbf function) is unchanged; if the second job (the "N" that becomes a "Y" due to the fault) is not within the interval, the demand has actually *de*creased and this interval cannot therefore be the one that defines the demand bound function for this particular interval-length (see Equation 15).

Hence in order to deal with multiple faults, we need only extend the form of analysis for a single fault, allowing for multiple faults rather than just one at the beginning of the sequence. The number of cases to be considered increases:

the analysis of a single fault required the consideration of a cases, where a is the numerator in the representation of the desired completion rate as a fraction. If our fault model allows for the possibility of up to w consecutive faults, then the total number of cases we need consider is bounded from above by $w \times a$, which remains pseudo-polynomial in the representation of the system. With our assumption on the fault model, $w = (b - a)$.

5. EXPERIMENTS

Implementation. We use the YALMIP modeling language [10] and SDPT3 semidefinite program solver [18] to solve the convex optimization problem to find the upper bound on the $\mathcal{L}_\infty$ to RMS induced gain for a specific successful transmission rate. We use the Truetime simulator [4] to implement the control tasks and the scheduler, and simulate the systems under different disturbance conditions. We assume that the plant state is transmitted by the sensor using the CAN protocol. The choice of CAN in our experiments is motivated by the fact that CAN is a widely used protocol in the domain of networked control systems and Truetime supports simulation using CAN protocol. The plant state is transmitted in a single precision floating point format. The plant has two states. Thus for 2 states the data packet from the sensor to the controller contains 8 bytes. We assume that the speed of the CAN bus is $250kbits/s$. As shown in [5], the transmission time of a CAN message m with 11-bit identifier and s_m data bytes is given by $(55 + 10s_m)\tau_{bit}$, where τ_{bit} is the time required to transmit 1 bit through the CAN network. With $250kbits/s$ speed of the CAN bus, $\tau_{bit} = 0.004ms$. Thus, the transmission of a message from the sensor to the controller requires $0.54ms$.

Case Study. We illustrate our results on the example of the five inverted pendulums modeled in Section 2.4, sharing a communication network and a processor. We assume that all pendulums have mass $m = 0.5$, and rotational friction coefficient $\rho = 0.6$. The pendulums differ from each other in their lengths, chosen as $[l_1, l_2, l_3, l_4, l_5] = [0.50, 0.50, 0.60, 0.50, 0.60]$, and their sampling times, chosen as $h = [h_1, h_2, h_3, h_4, h_5] = [15ms, 20ms, 20ms, 25ms, 25ms]$. We assume that the computation time for all the controllers is the same and equal to 4 ms. All constants and variables are expressed in SI units.

We use the algorithm provided in [5] to compute the worst case message delivery time for the individual control systems. The priorities for the CAN messages are assigned according to periods, with larger periods assigned lower priority. The worst case message delivery time for the control systems are given by $[f_1, f_2, f_3, f_4, f_5] = [0.54ms, 1.08ms, 1.62ms, 2.16ms, 2.70ms]$, and the deadlines by $[d_1, d_2, d_3, d_4, d_5] = [12.30ms, 17.30ms, 17.30ms, 22.30ms, 22.30ms]$. We assume that the rate of packet drop in the network is 0.05. Table 1 shows the controllers for the five pendulums, their minimum successful transmission rates $r_{\min}$, and the operating successful transmission rates r_{opr}. In all the cases, the operating successful transmission rate is $1 - r_{\text{net}}$.

We consider the following two disturbance scenarios in our simulation:

- **Disturbance Scenario 1:** A band-limited white noise with noise power 0.1 and sample time 0.01.

Systems	Controller	r_{min}	r_{opr}
System 1	[5.4395 -0.1315]	0.79	0.95
System 2	[5.8461 -0.0907]	0.59	0.95
System 3	[5.8843 0.2607]	0.62	0.95
System 4	[5.9949 -0.0750]	0.60	0.95
System 5	[5.5978 -0.0116]	0.68	0.95

Table 1: Parameters of the controllers

Disturbance Scenario	State Cost		Control Cost	
	Static	Dynamic	Static	Dynamic
white noise	331.10	330.76	328.00	323.00
pulse	186.92	186.94	335.00	339.00

Table 2: State cost and control cost for the static scheduler and the dynamic scheduler for System 3

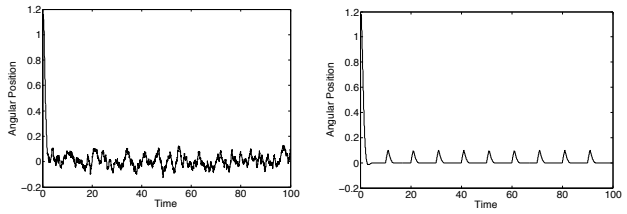
- **Disturbance Scenario 2:** A disturbance signal of pulse shape with amplitude 1 unit, period $10s$, pulse width $1s$, and zero phase delay.

Figure 5(a) and Figure 5(b) show the evolution of angular position of the plant for System 3 under the dynamic scheduler in the effect of the two disturbance scenarios, respectively.

Comparison with Static Scheduler. We compare our dynamic scheduler with the static scheduler presented in [11]. We first attempt to synthesize a static scheduler for the five systems in Table 1. However, the SMT solver Yices [6] could not generate a schedule even in 12 hours. The reason behind the failure in synthesizing the static scheduler is that the sampling periods are different for the control systems, thus the hyperperiod (i.e., the lcm of the periods) becomes large and the SMT solver needs to deal with many variables. We rather consider synthesizing a static scheduler for a group of four control systems where the individual control systems are an instantiation of system 3, having period to be $20ms$. We could easily synthesize a static scheduler for the four control systems using Yices. We compare the behavior of one control systems under both the static and the dynamic scheduler and under the two disturbance scenarios. Our comparison is based on the *state cost* and the *control cost*. The *state cost* is defined as the sum of the the Euclidean norm[2] of the state of the plant at the end of each sampling period for a time duration of 100s. The control cost is measured as the sum of the amplitude of the control signal at the end of each sampling period for a time duration of 100s. The state costs and the control costs are summarized in Table 2. The results show that the state cost and the control cost are comparable for the static and the dynamic scheduler. However, the benefit of our dynamic scheduling scheme is that unlike the static scheduler, the implementation of a dynamic scheduler does not have high computational overhead.

Another benefit of the dynamic scheduler over the static scheduler is that the dynamic scheduler is capable of operating with any rate of network packet drop r_{net} as long as the operating rates of the individual control systems are below $1 - r_{\text{net}}$. For example, in the case of synthesizing a scheduler for four control systems as above, suppose we want to maintain the computation rate of each control system at 0.70. If we want to synthesize a static scheduler for a network with

[2]The Euclidean norm of $x \in \mathbb{R}^n$ is given by $\|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$



(a) White noise (b) Pulse shaped disturbance

Figure 5: Evolution of angular position with time for System 3 from initial state $\langle 1, 1\rangle$ under the dynamic scheduler under the effect of (a)band-limited white noise and (b) a pulse shaped disturbance signal

r_{net}	State Cost		Control Cost	
	Static	Dynamic	Static	Dynamic
0.30	340.74	342.86	230.00	228.00
0.20	336.28	337.88	274.00	231.00
0.10	341.51	335.87	319.00	236.00
0.00	328.92	335.69	334.00	238.00

Table 3: State cost and control cost for the static scheduler and the dynamic scheduler for different values of r_{net} under Disturbance Scenario 1

packet drop $r_{\text{net}} = 0.30$, we have to choose the operating rate of the static scheduler to be 1. Now if we use the same static scheduler in another network with the rate of packet drop to be less than 0.3, the static scheduler will not be able to cope with the new situation and would schedule more computation than required. On the other hand, the dynamic scheduler will be able to adjust to the new requirement. Table 3 shows the state cost and the control cost of one of the four control systems for $r_{\text{net}} = 0.30$, $r_{\text{net}} = 0.20$, $r_{\text{net}} = 0.10$ and $r_{\text{net}} = 0$ under Disturbance Scenario 1. Our objective is to maintain the successful transmission rate at 0.70 and the static scheduler is synthesized for $r_{\text{net}} = 0.30$. The results show that both the static scheduler and the dynamic scheduler maintain almost the same state cost under different rate of packet drop. However, when the rate of packet drop decreases, the control cost under the static scheduler increases monotonically, whereas the dynamic scheduler maintains almost the same control cost for any rate of packet drop.

6. CONCLUSION

We have presented a dynamic scheduling methodology for achieving optimal disturbance rejection for a group of controllers in the presence of network packet drops and shared computational resources. Our scheduling algorithm is useful in integrated architectures of networked control systems, where multiple control loops share a single processor and communication medium. Although we have focused on a specific performance criterion ($\mathcal{L}_\infty$ to RMS induced gain), our methodology is applicable to other performance criteria as well, provided there is way to relate the rate of packet drops with performance.

7. REFERENCES

[1] K. J. Åström and B. Wittenmark. *Computer-controlled systems: theory and design*. Prentice-Hall, Inc., 2nd edition, 1990.

[2] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[3] M. S. Branicky, S. M. Phillips, and W. Zhang. Scheduling and feedback co-design for networked control systems. In *CDC*, pages 1211–1217, 2002.

[4] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

[5] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[6] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94, 2006.

[7] M. Garca-Rivera and A. Barreiro. Analysis of networked control systems with drops and variable delays. *Automatica*, 43(12):2054 – 2059, 2007.

[8] A. Hassibi, S. P. Boyd, and J. P. How. Control of asynchronous dynamical systems with rate constraints on events. In *CDC*, volume 2, pages 1345 –1351, 1999.

[9] M. Lemmon and X. S. Hu. Almost sure stability of networked control systems under exponentially bounded bursts of dropouts. In *HSCC*, pages 301–310, 2011.

[10] J. Lofberg. YALMIP: a toolbox for modeling and optimization in MATLAB. In *IEEE Symp. CACSD*, pages 284–289, 2004.

[11] R. Majumdar, I. Saha, and M. Zamani. Performance-aware scheduler synthesis for control systems. In *EMSOFT*, pages 299–308, 2011.

[12] R. Obermaisser, C. E. Salloum, B. Huber, and H. Kopetz. From a federated to an integrated architecture. *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, 28(7):956–965, 2009.

[13] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. SOSTOOLS: Control applications and new developments. *IEEE Symp. CACSD*, pages 315–320, 2004.

[14] A. Sangiovanni-Vincentelli and M. D. Natale. Embedded system design for automotive applications. *IEEE Computer*, 40(10):42–51, 2007.

[15] H. Shousong and Z. Qixin. Stochastic optimal control and analysis of stability of networked control systems with long delay. *Automatica*, 39(11):1877–1884, 2003.

[16] D. Soudbakhsh, L. T. X. Phan, A. Annaswamy, O. Sokolsky, and I. Lee. Co-design of control and platform with dropped signals. In *Proceedings of ICCPS*, 2013.

[17] K. W. Tindell, H. Hansson, and A. J. Wellings. Calculating controller area network (CAN) message response time. *Control Engineering Parctice*, 3(8):1163–1169, 1995.

[18] R. Tutuncu, K. Toh, and M. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming Ser. B*, 95:189–217, 2003.

[19] S. Vyas, A. Gupte, C. D. Gill, R. K. Cytron, J. Zambreno, and P. H. Jones. Hardware architectural support for control systems and sensor processing. *ACM Trans. Embed. Comput. Syst.*, 13(2):16:1–16:25, Sept. 2013.

[20] G. Walsh and H. Ye. Scheduling of networked control systems. *Control Systems, IEEE*, 21(1):57 –65, 2001.

[21] M. Yu, L. Wang, T. Chu, and G. Xie. Stabilization of networked control systems with data packet dropout and network delays via switching system approach. In *CDC*, volume 4, pages 3539–3544, 2004.

[22] F. Zhang, K. Szwaykowska, W. Wolf, and V. Mooney. Task scheduling for control oriented requirements for cyber-physical systems. In *RTSS*, pages 47– 56, 2008.

[23] L. Zhang, Y. Shi, T. Chen, and B. Huang. A new method for stabilization of networked control systems with random delays. *IEEE TAC*, 50(8):1177 – 1181, 2005.

[24] W. Zhang, M. S. Branicky, and S. M. Phillips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21:84–99, 2001.