In Proc. of the 12th International Conference on Hybrid Systems: Computation and Control, San Francisco, United States, April 2009.
Revised version.

Specification and Analysis of Network Resource Requirements of Control Systems

Gera Weiss[1], Sebastian Fischmeister[2], Madhukar Anand[1], and Rajeev Alur[1]

[1] Dept. of Computer and Information Science, University of Pennsylvania, USA
[2] Dept. of Electrical and Computer Engineering, University of Waterloo, Canada

Abstract. We focus on spatially distributed control systems in which measurement and actuation data is sent via a bus shared with other applications. An approach is proposed for specifying and implementing dynamic scheduling policies for the bus with performance guarantees. Specifically, we propose an automata-based scheduler which we automatically generate from a model of the controlled plant and the controller. We show that, in addition to ensuring performance, our approach allows adjustments to dynamic conditions such as varying disturbances and network load. We present a full development path from performance specifications (exponential stability) to a control design and its implementation using Controller Area Network (CAN).

1 Introduction

As control systems grow in both size and complexity, so does the need to spatially distribute control equipment such as sensors, actuator and computational devices. In recent years, implementations of distributed control systems are shifting from traditional hard-wired architectures, where each device is connected via a dedicated wire, to networked architectures, where control data is sent via shared communication buses (e.g., in the automotive [12] and aviation [20] industries, and for process control [24]). While, shared communication buses reduce costs and allow flexible architectures, they also introduce the problem of resource contention and require scheduling mechanisms to resolve them [19, 25].

Existing approaches to bus scheduling in control applications rely on static (periodic) schedules designed to assure performance in worst-case conditions [10, 18, 23]. The main disadvantage of static schedules, in our context, is that they lack a mechanism to adapt to changing conditions. This often leads to trading off average for worst-case performance.

In this paper, we propose a mechanism for generating schedules for shared buses such that a specified stability rate is guaranteed. We use guarded automata as a tool for formalizing the effect of bus scheduling on performance and as a mechanism for scheduling the network such that stability is guaranteed.

A scheduling approach is proposed that provides good performance both in average and worst-case conditions. We show that automata based scheduling allows the schedule to react to dynamic conditions such as the output of the plant or the load on the

This work was partially supported by NSF CNS 0524059, NSF CPA 0541149, and NSERC DG 357121-2008.

network and still guarantee high-level requirements such as stability. In particular, we demonstrate how, with our approach, the bus is only used when needed and, by that, good average performance can be obtained together with worst-case guarantees.

While our approach may apply also to other architectures, we concentrate on systems where one control loop shares a communication bus with background applications that use it for non–real-time communication. In this architecture, the network scheduler needs to assign communication bandwidth to the background applications while maintaining the specified stability performance. Focusing on this architecture allows us to present a holistic approach that begins with stability specification and ends with an implementation. The presented approach can be generalized to multiple control loops and to a star architecture where spatially distributed plants are controlled with a central controller.

The remaining of the paper consists of the following parts: in Section 2 we formally define the technical problem that this paper is about and sections 3-6 detail the steps towards its solution. Section 7 puts our work in context with related work. And, in Section 8, we draw conclusions from the results and outline potential next steps.

2 Shared Bus for Control and Background Traffic

Consider the system depicted in Figure 1 below, where the sensor and the actuator of a control system are spatially distributed and a shared communication bus is used to pass information from a computer processor near the sensor to a processor near the actuator.

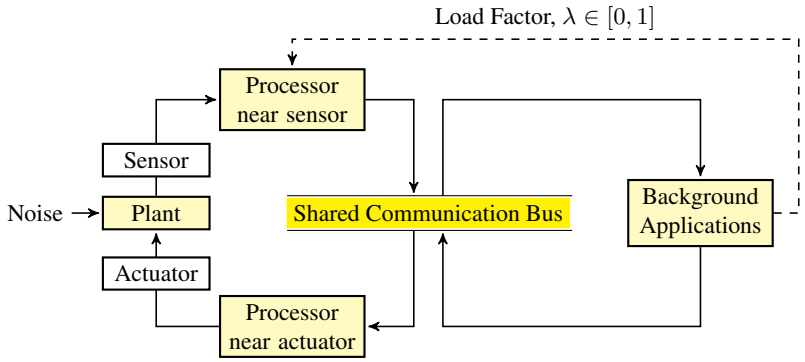


Fig. 1. A bus shared by a control loop and non real-time traffic.

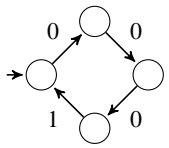
In this system, a communication bus is used both to close a control loop and for non real-time background applications. We assume TDMA (time division multiple access) arbitration, where messages are transmitted in separated time slots of fixed length (time-triggered message generation). The control loop can be modeled by a discrete-time control system where the sampling interval is the time slot of the network. To simplify notations and avoid orthogonal complications, we consider a single-input, single-output linear time-invariant plant.

Assume that the processor near the sensor has priority over the bus, i.e., when it decides to send data, all other messages are preempted. Suppose a time-varying number $\lambda \in (0, 1)$, called the load factor, is fed to the processor near the sensor. This number is an input parameter that models the fraction of bandwidth that the control system is asked to leave to background applications. The main problem addressed in this paper is how should the processor near the sensor decide when to send data, taking both λ and the output of the plant into account. Since decisions are taken online by devices with low computational power, we are especially interested in decision procedures with low online computational demands.

The scheduling problem can be solved by a static, time-triggered cyclic executive that assigns resources to either the control loop or the background applications [16, 10, 13]. This approach is depicted in Figure 2. Figure 2(a) shows an implementation of the approach using dispatch tables. The table describes, for specific time slots, which consumer uses the resource. This particular schedule shows that the background applications use the resource for the first three steps and the control loop uses the resource in the fourth step.

Time	*Consumer*
1	background
2	background
3	background
4	control loop

(a) Dispatch table



(b) Automaton

Fig. 2. Two ways for encoding a static schedule.

We can also encode such a cyclic executive using automata (see Figure 2(b)). The language of this automaton is the set of all sequences over $\{0, 1\}$ with 1 at every fourth position. The symbol 1 means that the control loop gets the network resource and 0 means that any of the background applications gets it. Note that implementing the scheduling policy of an automaton or a dispatch table can be done with a lightweight decision procedure requiring low (constant) time and memory.

This type of static scheduling is a common practice but it is also often wasteful. Static scheduling via dispatch tables and static automata is useful because of analyzability and ease of implementation. However, static scheduling often uses more resources than necessary, because many applications do not require a fixed sampling frequency to assure performance. For example, consider a system with sporadic disturbance bursts. In this case, a periodic sensor update often provides no additional information to the processor near the actuator and therefore is a waste of network resources. An improved version will only send measurements if the plant's output exceeds some threshold discrepancy, as we show in the following example.

On the other extreme, bus arbitration could also be decided by a tailored, fully dynamic software. The main problem with the latter approach is that it is not clear how

to analyze and systematically design such software. In this paper, we propose a mid-way between fully dynamic code and dispatch tables. Using guarded automata, we propose a scheduling mechanism that allows analyzability and lightweight implementation (as static scheduling) with adaptability and efficiency (as dynamic scheduling).

The following example illustrates how guarded automata can be used for scheduling. Note that, while the example is an ad hoc (manually designed) automaton, the methodology we are proposing is automatic generation of automata using the construction described in this paper.

For the system depicted in Figure 1, assume that the processor near the sensor maintains an estimate of the plant state denoted by x_{sen}, and the processor near the actuator maintains its own estimate (based on the information available to it) that we denote by x_{act}. Consider the scheduling scheme depicted in Figure 3. In words: the scheduling decision is based on the difference $|x_{\mathrm{sen}} - x_{\mathrm{act}}|$. If it lies above 1, then data is transmitted. If it lies below .5, then the processor near the sensor will not transmit and leave the slot for the background applications. If $|x_{\mathrm{sen}} - x_{\mathrm{act}}|$ lies between .5 and 1, the processor near the sensor will transmit a reading once and then pause in the next step.

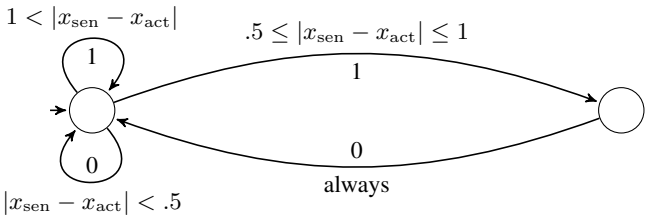


Fig. 3. A guarded (dynamic) schedule.

Clearly, this scheduling scheme is more expressive than cyclic scheduling using dispatch tables or unguarded automata. Still, unlike general dynamic techniques, the model is simple enough for formal analysis of system properties such as exponential stability. In this paper we focus on generating guarded automata of the form depicted in Figure 1 that guarantee high-level requirements of the control application. Specifically, we investigate automata generation for exponential stability requirements.

In the following sections we propose steps towards synthesizing a scheduler that guarantees exponential stability and uses the bus only when needed or when the parameter λ is small (low background traffic). The proposed methodology is described in four steps, each described in a separate section and summarized as follows. The first step, described is Section 3, is a construction of an automaton that specifies unstable runs of the control loop. The second step, described in Section 4, is to transform the specification automaton to an executable state machine that identifies when using the bus is critical for ensuring stability. The third step, described in Section 5, is to implement the scheduling scheme with a distributed bus arbitration mechanism. The fourth step, described in Section 6, is to test and validate the mechanism by implementing a switched control strategy and a scheduling scheme that combines the parameter λ with the executable state machine.

3 Step I: Specification Automaton

As a first step towards solving the problem presented in Section 2, we propose an automaton that specifies unstable runs, as follows.

We use switched systems (see e.g. [14]) to model the system depicted in Figure 1. The switched system has two modes: (1) a mode that models the transformation of the state variables when the processor near the sensor is using the bus and (2) a mode that models the transformation when the bus is not used by the processor near the sensor. See [1, 2] and the examples given in Section 6 for more details about this modeling approach.

Formally, let $A_0, A_1 \in \mathbb{R}^{n\times n}$ and $c_0, c_1 \in \mathbb{R}^{1\times n}$ be such that

$$\begin{aligned} x(t+1) &= A_{w(t)}x(t); \\ y(t) &= c_{w(t)}x(t), \end{aligned} \tag{1}$$

models the dynamics of the control loop depicted in Figure 1, where $x(t) \in \mathbb{R}^n, y(t) \in \mathbb{R}$ are the state and the observation at time t, respectively. The infinite word $w \in \{0,1\}^\omega$ (called the switching sequence) is such that the processor near the sensor sends data to the processor near the actuator at time t iff $w(t) = 1$. A run of the system is a solution of the equations.

As a performance measure we choose exponential stability. The standard definition of exponential stability requires that behaviours converge to the origin faster than a given exponentially decaying function. In [1], a system is defined to be (ρ, l)-exponentially-stable if in every l time units the distance to the origin (norm) decreases by a factor of ρ. In this paper we add two more parameters. Specifically, for the parameters $0 < \rho \leq 1, l \in \mathbb{N}$ and $0 < \varepsilon < \delta$,

Definition 1. A run of the system (1) is $(\rho, l, \varepsilon, \delta)$-exponential-stable if $\varepsilon < |x(t)| < \delta \implies |x(t+l)| < \rho|x(t)|$ for every $t \in \mathbb{N}$.

Namely, a run is exponentially stable if any state, in the δ-ball and not in the ε-ball around the origin, gets closer to the origin by a factor ρ, every l steps.

In the following definition, a regular language (over an infinite alphabet) is used to specify runs of the system that are not exponentially stable.

Definition 2. A language over the alphabet $\Sigma = \{0,1\} \times \mathbb{R}$ is a $(\rho, l, \varepsilon, \delta)$-safe-monitor for the system (1) if for every run that is not $(\rho, l, \varepsilon, \delta)$-exponentially-stable there exists $k \in \mathbb{N}$ such that the word $\langle w(1), y(1)\rangle \cdots \langle w(k), y(k)\rangle$ is in the language.

Namely, a safe-monitor is an automaton such that if an accepting state is not reached (when the outputs and modes of the plants are fed as inputs to the automaton) then the run is exponentially stable with the required parameters. If the accepting state is reached then the run may not be safe. The approach that we are proposing in this paper is to avoid accepting states and, by that, assure, for instance, exponential stability.

In the rest of this section, we give a construction of a non-deterministic automaton whose accepted language is a $(\rho, l, \varepsilon, \delta)$-safe-monitor called $(\rho, l, \varepsilon, \delta)$-specification-automaton (because it specifies $(\rho, l, \varepsilon, \delta)$-exponentially-stable runs). Note that, while we propose a specific construction, it is not the only $(\rho, l, \varepsilon, \delta)$-specification-automaton. However, we are not assuming the specific construction in the rest of the paper (only the properties given in Definition 2).

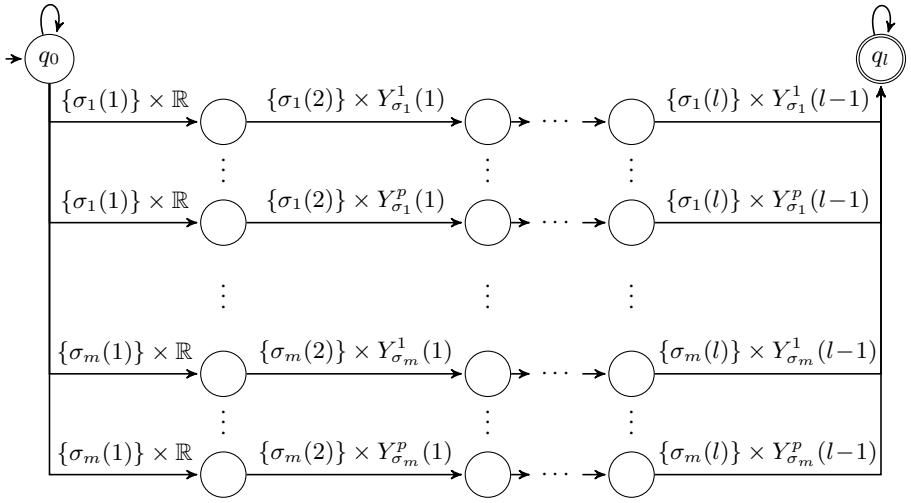


Fig. 4. A non-deterministic automaton over the alphabet $\{0,1\} \times \mathbb{R}$ such that if no prefix of $\langle w(1), y(1)\rangle, \langle w(2), y(2)\rangle, \ldots$ is accepted then any run of the system (1) with switching signal $w(1), w(2), \ldots$ and outputs $y(1), y(2), \ldots$ is exponentially stable.

Construction 3. Let $0 < \rho \leq 1, l \in \mathbb{N}$ and $0 < \varepsilon < \delta$ be the required exponential stability parameters.

- For each $\sigma = \sigma(1) \cdots \sigma(l) \in \{0,1\}^l$:
  - Let B_σ be the set of all $x \in \mathbb{R}^n$ such that $|A_{\sigma(l)} \cdots A_{\sigma(1)} x| \geq \rho|x|$ and $\varepsilon \leq |x| \leq \delta$. In words, B_σ is the set of all vectors, in the δ–ball but not in the ε–ball, whose distance to the origin does not shrink by a factor of ρ when the transformation $A_{\sigma(l)} \cdots A_{\sigma(1)}$ is applied. We focus on this set because, to guarantee exponential stability, we can make sure that whenever the current state is in B_σ the switching signal for the next l steps is not σ. Let $B^1_\sigma, \ldots, B^p_\sigma$ be a finite cover of B_σ by compact convex sets.
  - Let $Y^j_\sigma(k) := \{o(k)x : x \in B^j_\sigma\}$ where $o(k) := c_{\sigma(k)} A_{\sigma(k-1)} \cdots A_{\sigma(1)}$. In words, $Y^j_\sigma(k)$ is the set of possible outputs that we may observe at time $t+k-1$ if $x(t)$ is in B^j_σ and $w(t) \cdots w(t+l-1) = \sigma$. Note that $Y^j_\sigma(k)$ is an interval whose bounds can be effectively computed, because B^j_σ is compact and convex.
- Let $\sigma_1, \ldots, \sigma_m$ be the set of words of length l such that $|A_{\sigma_i(l)} \cdots A_{\sigma_i(1)}| \geq \rho$, i.e. the words such that $B_{\sigma_i} \neq \emptyset$.
- For the switched system (1), we define a non-deterministic automaton (depicted in Figure 4). The states of the automaton are $\{q_{i,j}(k) \colon i = 1, \ldots, m, j = 1, \ldots, p$ and $k = 1, \ldots, l-1\} \cup \{q_0, q_l\}$. The transition from q_0 to every $q_{i,j}(1)$ is guarded by the condition $w(t) = \sigma_i(1)$. For, $k = 2, \ldots, l-1$, the transition from $q_{i,j}(k-1)$ to $q_{i,j}(k)$ is guarded by the condition $w(t) = \sigma_i(k) \wedge y(t-1) \in Y^j_{\sigma_i}(k-1)$. The transition from every $q_{i,j}(l-1)$ to q_l is guarded by $w(t) = \sigma_i(l) \wedge y(t-1) \in Y^j_{\sigma_i}(l-1)$.

Finally, the self loops on states q_0 and q_l are unconditioned. The initial state is q_0 and the only accepting state is q_l.

– By construction, if no run of the automaton gets to $F = \{q_l\}$ at time t then the product of the last l matrices takes $x(t-l)$ closer to the origin by a factor of at least ρ; assuming that $\varepsilon < |x(t)| < \delta$. In particular, since this is true for every t, we get that the system (1) is exponentially stable.

The only non-constructive step in the above description is covering B_σ by a finite number of compact convex sets (where $\sigma = \sigma(1)\cdots\sigma(l)$ is an arbitrary word). Let $A_\sigma = A_{\sigma(l)}\cdots A_{\sigma(1)}$. Towards a cover, we explore the geometry of the set B_σ, as depicted in Figure 5. Consider the sphere $S_\delta = \{x \in \mathbb{R}^n : |x| = \delta\}$. The linear transformation A_σ maps this sphere onto an ellipsoid $E = \{A_\sigma x : x \in S_\delta\}$. The intersection $B_\sigma \cap S_\delta$ can be visualized as two symmetric arcs on the sphere (the part of the sphere that is mapped to the part of E that is outside of the $\rho\delta$-sphere). By linearity, $B_\sigma = \{\lambda x : \lambda \in [\varepsilon/\delta, 1), x \in B_\sigma \cap S_\delta\}$ which can be visualized as a pair of two symmetric trimmed cones. Let a be the largest semi-axes of the ellipsoid E and h be such that $A_\sigma h = a$. Then, the cover is $B^1_\sigma = \{x \in B_\sigma : h^T x \geq \gamma\}$ and $B^2_\sigma = \{x \in B_\sigma : h^T x \leq -\gamma\}$ where γ is the largest number such that these two sets cover B_σ. In practice, one can compute h using singular value decomposition of A_σ.

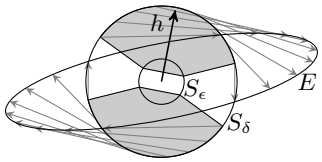


Fig. 5. A coverage of B_σ by two compact convex sets, B^1_σ and B^2_σ (grayed).

Example 1. Consider the system (1) where $A_0 = \begin{pmatrix} -1 & 1/2 \\ 0 & 1 \end{pmatrix}$, $A_1 = \begin{pmatrix} 1/2 & -1 \\ 1 & 0 \end{pmatrix}$ and $c_0 = c_1 = \begin{pmatrix} 1/2, 1/2 \end{pmatrix}$. Let $\rho = 1$, $l = 10$, $\varepsilon = 1/10$, and $\delta = 1$. Assume that the switching signal is zero for 10 consecutive steps. A stability monitor that can only base its decision on the switching signal (as the one considered e.g. in [1]) must accept the run as potentially unstable because the norm of A_0 to the power 10 is bigger than one (which means that there exists x such that $\|A_0^{10}x\| > \|x\|$). However, a closer examination reveals that we only have a problem if the polar angle $\alpha = \tan^{-1}(x_2/x_1)$ of the initial state satisfies $-\cos^{-1}\left(-38/\sqrt{1973}\right) \leq \alpha \leq -\cos^{-1}\left(18/\sqrt{2173}\right)$ or $\cos^{-1}\left(38/\sqrt{1973}\right) \leq \alpha \leq \cos^{-1}\left(-18/\sqrt{2173}\right)$. The first case, corresponding to the set $B^1_{\sigma_1}$ in the construction (considering also $\varepsilon < \|x\| < \delta$), is depicted in Figure 6. Figure 6(a) shows an over approximation of the bad set based on the first two observation, $y(1) = c_0x(0), y(2) = c_0A_0x(0)$. Figure 6(b) shows an over approximation of the bad set based on the first five observation, $y(1) = c_0x(0), \ldots, y(5) = c_0A_0^4x(0)$. In the automaton depicted in Figure 4, these are the initial states for which the automaton will get to the third and sixth states on the first horizontal path, respectively. More

generally, the first horizontal path of the automaton corresponds to the "non deterministic guess" that the next 10 values of the switching signal are going to be zeroes. The guards are designed such that the path is abandoned if this guess turns to be wrong or the observations show that the initial state was not in $B^1_{\sigma_1}$.

(a) Inequalities over $y(1)$ and $y(2)$ (b) Inequalities over $y(1), \ldots, y(5)$

Fig. 6. Over-approximations of bad initial states by linear inequalities over the observations.

4 Step II: Executable State Machine

The specification automaton, described in the preceding section allows to detect errors, but is not directly applicable for scheduling. In this section we use the specification automaton to obtain an executable state machine that identifies times where sending a message from the processor near the sensor to the processor near the actuator is essential for keeping the control-loop stable.

Towards such an executable state machine, we define some of the states of the specification automaton as `bad` states. Specifically, q is marked as a `bad` state if there is a path $q = q(1), \cdots, q(k)$ (of arbitrary length k) from q to an accepting state and a sequence $y_1, \cdots, y_{k-1} \in \mathbb{R}$ such that $q(i+1) \in \delta(q(i), \langle 1, y_i \rangle)$ for each $i = 1, \ldots, k-1$ (where δ is the transition function).

Since every accepting state is a `bad` state, avoiding `bad` states ensures exponential stability. But, unlike the case for accepting states, we now have also the following property: if q is not a `bad` state then all the states in $\delta(q, \langle 1, y \rangle)$ are not `bad`, for any y. This property is useful for scheduling because it means that we can always avoid a transition to a `bad` state by scheduling mode 1. Note that we are assuming that the initial state is not `bad`. This assumption makes sense because if the initial state is already `bad`, we will not get the required stability even if the bus is dedicated to the control loop. Practically, we are assuming that the control design is such that the requirements are met when the bus is always available.

An executable state machine is obtained by simulating the automaton. For a finite trace $T = \langle w(1), y(1) \rangle, \ldots, \langle w(t-1), y(t-1) \rangle$ of the system (1), the state of the state-machine, at time t, is the set $\delta^*(q_0, T) \subseteq Q$ consisting of the end states of all runs of the automaton up to time t. We say that the state-machine is in a `must` state if choosing $w(t) = 0$ will make that state contain a `bad` state of the automaton. As the transition from a state that is not `bad` to a `bad` state is conditioned on $w(t) = 0$ (otherwise the source is also `bad`), we are guaranteed that we can avoid `bad` states by scheduling mode

1 whenever the executable state machine (described in the previous section) is in a `must` state.

Note that our approach can be directly generalized to any number of concurrent control loops. Imagine, for example, a second control loop (another triplet of sensor, actuator and plant) that shares the same bus for communicating data from sensor to actuator. In this case, the scheduler of each loop is going to get to a `must` state if it must send in one of the next two communication slots.

5 Step III: Stateful Priority Assignment

The executable state machine, described in the previous section, detects times when sending a message is essential (when the machine is in a `must` state), but it does so in a centralized manner. In this section we discuss an implementation of it using priority based bus arbitration.

For implementation, we propose the use of priority based bus arbitration, but, instead of assigning priorities to nodes or to message types, we propose to assign priorities to states of the scheduler. Specifically, the priority of a message from the processor near the sensor to the processor near the actuator is high when the scheduler is in a `must` state and low otherwise.

Assigning priorities to messages based on system state has not been considered so far, because the developer must assure that, at any point in time, each priority level is used by at most one node. In this paper, a formal model (automaton) assist the developer in guaranteeing this property for large, complex systems.

Using CAN. Controller Area Network (CAN) [6] is the most widespread priority based networking technology for control applications. It provides eleven or 29 bits for encoding an unsigned integer priority level for individual messages and requires two wires for the physical communication layer. One wire implements the dominant bits while the other implements recessive bits. Using a logical-AND between those two bits, the bus implements a bit-wise arbitration mechanism, which allows the winning node to continue sending its message during the arbitration.

The first step, towards applying CAN, is to annotate the transitions of the scheduler with priorities (highest and lowest), as follows: if the scheduler is in a `must` state at time t, then it will have the highest priority. All other transitions in the scheduler get assigned some value other than the highest priority level. The second step is to assign priorities to background applications. The specific priorities assigned the background applications and the sensor can be tweaked to fit bandwidth defined by λ (see Section 6).

Note that implementing our approach with CAN requires no extra hardware. With eleven bits, the control engineer can assign 2046 priority levels (one for broadcast), which means that the control application uses the priority levels `0x07FF` and `0x001`. This leaves plenty of additional priorities for background applications. Using 29 bits raises this range even further.

Our approach assigns different priorities to the same message depending on the message context—high priority if the message is important, low priority if it is unimportant. In priority-driven arbitration such as CAN, nodes that simulatneously access the bus must use different priorities, otherwise the bus arbitration will fail and cause data corruption. Our approach still follows this rule, because instead of assigning one

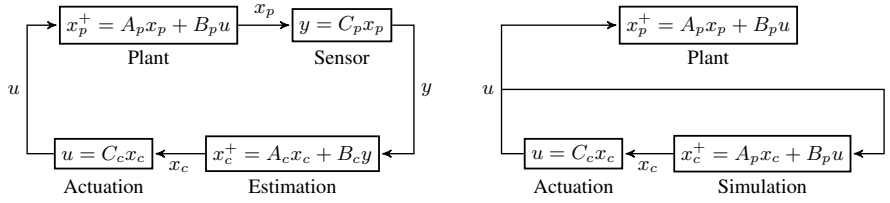
priority level to a message, we assign two. Both priority levels are exclusively used for this message. A straightforward way to extend our approach is to share priorities among messages and use formal verification of the schedulers to guarantee that two schedulers never use the same message priority at the same time.

6 Step IV: Testing and Validation

In this section we revisit the initial example, provide more technical details on controller and scheduler designs and present some performance analysis that illustrates the advantages and disadvantages of our approach.

Controller Design. As a specific example, we examine how an LQG controller can be implemented with the architecture depicted in Figure 1, above, where a shared communication bus separates the processor near the sensor from the processor near the actuator. The control loop is designed as a switched system with the following two modes.

Figure 7(a) shows the feedback mode, active when the processor near the sensor sends data on the bus. The matrices A_p, B_p, C_p model the controlled plant and the matrices A_c, B_c, C_c are computed using a standard technique for LQG design (e.g., by MATLAB's [17] `lqg` command).



(a) Feedback Mode

(b) Feedforward Mode

Fig. 7. Two modes of the control loop. The feedback mode, active when processor near the sensor sends data to the processor near the actuator, is a full LQG based feedback. In the feedforward mode, data is not sent. Instead, the processor near the actuator simulates the dynamics of the plant based on earlier data.

The second mode of the controller corresponds to times at which the processor near the sensor does not transmit its reading. In these times, the processor near the actuator simulates the dynamics of the plant. Figure 7(b) shows this second mode. A simulation block replaces the estimation block and the output of the plant remains unused (because data is not sent).

The composition of the system with the controller modes results in the closed-loop switched system described by equation (1), where $A_0 = \begin{pmatrix} A_p & B_p C_c \\ 0 & A_p + B_c C_p \end{pmatrix}$, $A_1 = \begin{pmatrix} A_p & B_p C_c \\ B_c C_p & A_c \end{pmatrix}$ and $x(t) = (x_p^T, x_c^T)^T$. The switching signal $w \in \{0,1\}^{\mathbb{N}}$ is such that

$w(t)$ is one iff the processor near the sensor sends data to the processor near the actuator at time t.

As an example, consider the plant $\dot{x}_p = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} x_p + \begin{pmatrix} 1 \\ 0 \end{pmatrix} u, y = (0,1)x_p$. In MATLAB, the matrices A_0 and A_1 can be computed as follows: (1) get a discrete-time model using `c2d`, (2) compute an LQG compensator using `lqg`. The closed loop matrices obtained by this procedure (using $Ts = 1$ and $QXU = QWV = .1$) are:

$$A_1 = \begin{pmatrix} 0.568 & 0.432 & -0.357 & -0.339 \\ 0.432 & 0.568 & -0.142 & -0.134 \\ 0 & 0.412 & 0.210 & -0.319 \\ 0 & 0.461 & 0.291 & -0.028 \end{pmatrix} \text{ and } A_0 = \begin{pmatrix} 0.568 & 0.432 & -0.357 & -0.339 \\ 0.432 & 0.568 & -0.142 & -0.134 \\ 0 & 0 & 0.210 & 0.094 \\ 0 & 0 & 0.291 & 0.433 \end{pmatrix}.$$

The network is scheduled based on the difference between the output and the estimated output, i.e., $c_0 = c_1 = (0, 1, 0, -1)$.

Scheduling Scheme. As described in Section 2, the parameter λ specifies the fraction of slots that the processor near the sensor should leave for background applications. To achieve this requirement, we propose the following scheduling scheme. The executable state machine, described in Section 4, is used in the following way. If the machine is at a `must` state, the processor near the sensor sends data unconditionally. Otherwise, a Bernoulli trial with $1 - \lambda$ probability of successes is conducted and a message is sent only if the experiment successes. The input to state-machine is the sequence of decisions of the processor near the sensor and the output of the plant.

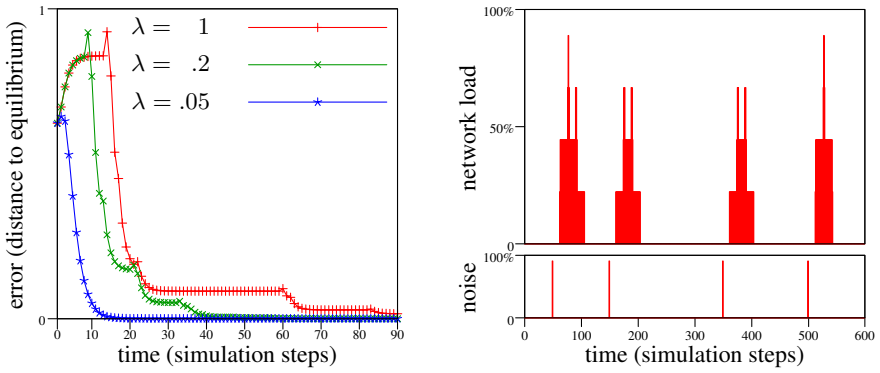
This scheduling scheme respects both λ and the needs of the control loop. The parameter λ determines the likelihood of using the bus when the conditions of the control loop allow not to. Note that an implementation of this scheme in a distributed environment means that decisions are made at the processor near the sensor. This may require the processor near the sensor to also compute the estimation as the processor near the actuator does, to get the estimated output.

Simulation Data. To test our approach, the automaton described in Section 3 is constructed with the parameters $l = 15, \delta = 1, \epsilon = .1, \rho = 1$ and the matrices A_0 and A_1 above.

The graphs in Figure 8(a) show how our approach allows dynamic adaptation of control performance. The plots demonstrate an improvement in control performance (faster convergence) when the competition for resources is lower (smaller values of λ). This type of adaptation is not achievable with static scheduling, when the schedule is planned only for the worst-case scenario.

In another experiment, we executed the scheduler with the parameter $\lambda = 0$, and injected random noise to the control system at irregular intervals. The plot in Figure 8(b) shows that the network is only used some time after each disturbance. The upper part shows the network bandwidth used by the control loop and the lower part shows the introduced disturbances. Static approaches (including the one described in [1, 2]) that do not use the output of the plant to direct scheduling decisions cannot achieve this type of dynamic adaptation.

The conclusion from the simulations is that our approach to scheduling gives best benefits for systems that operates in dynamic conditions. While static scheduling may



(a) Faster convergence for smaller λ.

(b) Higher load after noise injection.

Fig. 8. Simulation results showing how dynamic scheduling allows adjusting control performance to network availability and adjusting network usage to control needs.

give good results for systems with constant network load and evenly distributed disturbances, our approach delivers better performance when varying load and disturbances that come in irregular bursts are present.

Integration Into Simulink. Simulink is the de-facto standard for modeling and analysing control system. We integrate our guarded automata approach into Simulink via the Network Code Machine extension [9] in the TrueTime library. TrueTime [7] is a Simulink simulator library for embedded and networked control systems. It can simulate a number of different communication arbitration mechanisms. We extended the TrueTime library with a block for the Network Code Machine, which takes a node identifier as input and provide access to the network to the specified node. Our extension is available on the project web site and is planned to be part of the next major TrueTime release.

Figure 9 provides a sample model that shows how to run the original model shown in Figure 1 in this Simulink environment. The left part shows the control application with the actuator, plant, and sensor blocks. The middle shows the networking part consisting of the TrueTime Network, the messaging and reception blocks. The Guarded Automaton-based Scheduler block and the Network Code Machine block also belong to the network part. The former implements the $(\rho, l, \varepsilon, \delta)$-specification-automaton as a state machine and the latter is an S-function extension for TrueTime to schedule the networked as specified in the automaton. The right side of the figure shows background application.

7 Related Work

Methodology. The concept of automata-based scheduling was introduced in [1] and [2] in the context of CPU scheduling. The main contribution of the current paper over the

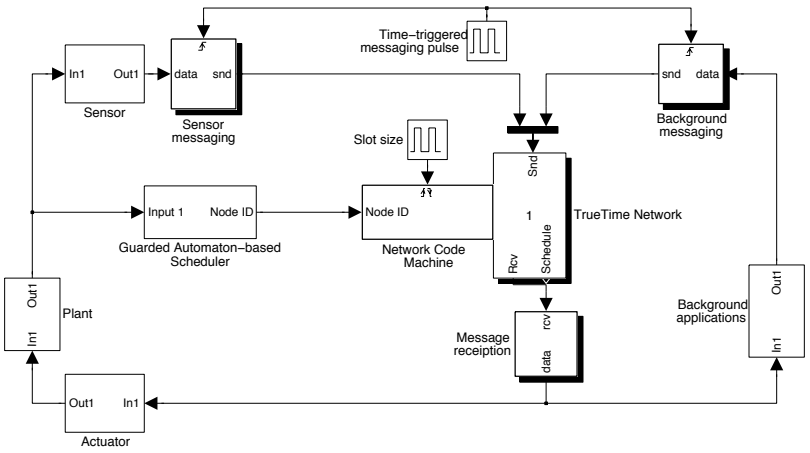


Fig. 9. Guarded automaton in Simulink with TrueTime and Network Code Machine.

previous work on automata based scheduling is that it studies the use of guards as a way to direct the schedules based on dynamic data.

Another, more technical, contribution is the elimination of the need to determinize the automaton. In [1] and [2], the proposed methodologies involve computing an automaton similar (in spirit) to the automaton depicted in Figure 4 and then determinizing it. As determinization does not scale, this paper proposes a direct way of using the non-deterministic automaton for scheduling. Determinization is especially problematic with guarded automata, because they usually have more states and because the formal language they induce is over an infinite alphabet (the real numbers).

The concept of automata-based scheduling generalizes tree schedules [9]. The main structural difference is that tree schedules require the underlying structure to be a tree resulting in periodic resets to the root location. While the work on tree schedules mainly concentrated on analyzing [3], composing [4], verifying, and implementing [9] them, this work concentrates on the generation of schedulers.

Network Control Systems. Form the control perspective, the insertion of communication networks in the control loop is usually viewed as a source of random time delays and information loss (see e.g., [7, 5, 11, 15, 22, 26]). For scheduling, this view leads to mechanisms such as [18, 23], where a periodic schedule that can cope with the worst case delays is proposed. Our approach views the network as a shared resource. Particularly, we do not view the other users of the network as introducing random delays but as components of the system that we need to take into consideration. This perspective allows more efficient use of the resource.

Communication Arbitration. In distributed control systems, nodes must access the network mutually exclusively. The dominating approaches use either temporal isolation or priority-based mechanisms. System such as the TTA [13], TTCAN [10], or FTT-CAN [8] provide temporal isolation where each node accesses the network at predefined time slots. Our approach follows a similar line in that it uses temporal isolation in its TDMA scheme. However, in contrast to works such as the TTA and TT-Ethernet [21],

our approach uses temporal isolation to synchronize steps in the automata and then uses priority-based arbitration for resource contentions instead of globally defined schedules.

Common architectures using priority-based mechanisms for resolving resource contentions either assign priorities to individual nodes or to individual messages. It is mandatory that each concurrent access must use a unique priority level, because otherwise the collision-avoidance mechanism will fail. Thus, the common architectures such as CANopen usually use a static global database assigning each message its unique priority. Our approach differs from such architectures in that we assign individual priorities to messages based on the context of the application; meaning as a message becomes more important to the application, its priority level changes. Although the method sounds intuitive and simple, prior approaches had to rely on a quasi-static assignments of priority levels to nodes or messages, because dynamic assignment must guarantee unique priority levels. Our system can guarantee this, because we can statically check whether two nodes on the same network will ever try to communicate simultaneously with the same priority level.

8 Conclusions and Future Work

We proposed a dynamic scheduling scheme for network control systems. The main idea is using automata with guards to decide resource assignments—in our case network slots. The work contains a full walk through the development process comprising of:

1. Specifying stability parameters as high-level performance requirements.
2. Generating an automaton that specifies traces of the system implying that the requirements are not met.
3. Constructing a scheduler state-machine that guarantees avoidance of the specified traces.
4. Implementing the scheduler with priority-based congestion arbitration.

The approach is demonstrated with experiments that show its advantages compared to standard approaches and previous work. Furthermore, the experimental data demonstrates the unique ability of our approach to adjust the priority level to its context *and* stay verifiable: after disturbances the control application requires more bandwidth to adjust the plant and therefore uses elevated priority levels. While during calm operations the priority levels remain low. Future work can look into multiple-output systems, more elaborate guards on the specification automaton and more sophisticated, possibly optimal, construction algorithms for the scheduler.

References

1. R. Alur and G. Weiss. Automata-based Interfaces for Control and Scheduling. In *Proc. 10th Conference on Hybrid Systems: Computation and Control*, 2007.
2. R. Alur and G. Weiss. Regular Specifications of Resource Requirements for Embedded Control Software. In *Proc. 14th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS)*, 2008.
3. M. Anand, S. Fischmeister, and I. Lee. An Analysis Framework for Network-Code Programs. In *Proc. of the 6th Annual ACM Conference on Embedded Software (EMSOFT)*, pages 122–131, Seoul, South Korea, Oct. 2006.

4. M. Anand, S. Fischmeister, and I. Lee. Composition Techniques for Tree Communication Schedules. In *Proc. of the 19th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 235–246, Pisa, Italy, July 2007.
5. P. Antsaklis and J. Baillieul. Guest editorial. Special Issue on Networked Control Systems. *IEEE Trans. Automat. Control*, 49(9):1421–1423, 2004.
6. Bosch. *CAN Specification, Version 2*. Robert Bosch GmbH, September 1991.
7. A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén. How Does Control Timing Affect Performance? *IEEE Control Systems Magazine*, 23(3):16–30, 2003.
8. J. Ferreira, P. Pedreiras, L. Almeida, and J. Fonseca. The FTT-CAN Protocol For Flexibility in Safety-critical Systems. *IEEE Micro*, 22(4):46–55, July-Aug. 2002.
9. S. Fischmeister, O. Sokolsky, and I. Lee. A Verifiable Language for Programming Communication Schedules. *IEEE Trans. on Comp.*, 56(11):1505–1519, Nov. 2007.
10. T. Führer, B. Müller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time Triggered Communications on CAN (Time Triggered CAN–TTCAN). In *Proc. 7th International CAN Conference*, Amsterdam, Netherlands, 2000.
11. D. Hristu-Varsakelis and W. S. Levine, editors. *Handbook of Networked and Embedded Control Systems*. Birkhäuser, 2005.
12. S. Kawamura and Y. Furukawa. Automotive Electronics System, Software, and Local Area Network. In *Proc. of the International Conference on Hardware/Software Codesign and System Synthesis*, 2006.
13. H. Kopetz. *Real-time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.
14. D. Liberzon. *Switching in Systems and Control*. Systems & Control: Foundations & Applications. Birkhäuser Boston Inc., Boston, MA, 2003.
15. H. Lin and P. J. Antsaklis. Stability and Persistent Disturbance Attenuation Properties for a Class of Networked Control Systems: Switched System Approach. *Internat. J. Control*, 78(18):1447–1458, 2005.
16. J. Liu. *Real-Time Systems*. Prentice-Hall, New Jersey, 2000.
17. `http://www.mathworks.com/products/matlab`.
18. H. Park, Y. Kim, D. Kim, and W. Kwon. A Scheduling Method For Network-based Control Systems. *IEEE Trans. on Control Systems Technology*, 10(3):318–330, May 2002.
19. A. Ray and Y. Halevi. Integrated Communication and Control Systems: Part II–Design Considerations. *ASME Journal of Dynamic Systems, Measurements and Control*, 110:374–381, 1988.
20. M. A. Sánchez-Puebla and J. Carretero. A New Approach for Distributed Computing in Avionics Systems. In *Proc. of International Symposium on Instrumentation and Control Technology (ISICT)*, pages 579–584. Trinity College Dublin, 2003.
21. K. Steinhammer, P. Grillinger, A. Ademaj, and H. Kopetz. A Time-Triggered Ethernet (TTE) Switch. In *Proc. of the Conference on Design, Automation and Test in Europe (DATE)*, pages 794–799, 3001 Leuven, Belgium, 2006. European Design and Automation Association.
22. G. Walsh, H. Ye, and L. Bushnell. Stability Analysis of Networked Control Systems. *IEEE Transactions on Control Systems Technology*, 10(3):438–446, 2002.
23. P. Wen, J. Cao, and Y. Li. Design of High-performance Networked Real-time Control Systems. *IET Control Theory and Applications*, 1(5):1329–1335, 20070901.
24. L. Yliniemi and K. Leiviskä. Process Control Across Network. In *Proc. of Parallel and Distributed Computing and Networks (PDCN)*, pages 168–173, Anaheim, CA, USA, 2006. ACTA Press.
25. W. Zhang, M. Branicky, and S. Phillips. Stability of Networked Control Systems. *IEEE Control Systems Magazine*, 21(1):84–99, Feb 2001.
26. W. Zhang and L. Yu. Output Feedback Stabilization of Networked Control Systems with Packet Dropouts. *IEEE Trans. Automat. Control*, 52(9):1705–1710, 2007.