Using JitterTime to Analyze Transient Performance in Adaptive and Reconfigurable Control Systems

Anton Cervin
Department of Automatic Control
Lund University
Lund, Sweden
anton.cervin@control.lth.se

Paolo Pazzaglia
Real-Time Systems Laboratory
Scuola Superiore Sant'Anna
Pisa, Italy
p.pazzaglia@santannapisa.it

Mohammadreza Barzegaran
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Kgs. Lyngby, Denmark
mohba@dtu.dk

Rouhollah Mahfouzi
Department of Computer and Information Science
Linköping University
Linköping, Sweden
rohollah.mahfoozi@liu.se

***Abstract*—This paper presents JITTERTIME, a small Matlab toolbox for calculating the transient performance of a control system in non-ideal timing scenarios. Such scenarios arise in networked and embedded systems, where several applications share a set of limited and varying resources. Technically, the toolbox evaluates the time-varying state covariance of a mixed continuous/discrete linear system driven by white noise. It also integrates a quadratic cost function for the system. The passing of time and the updating of the discrete-time systems are explicitly managed by the user in a simulation run. Since the timing is completely handled by the user, any complex timing scenario can be analyzed, including adaptive scheduling and reconfiguration between different system modes. Three examples of how the toolbox can be used to evaluate the control performance of such time-varying systems are given.**

I. INTRODUCTION

Standard sampled-data control theory [1] assumes that measurement samples are taken at a fixed rate, that there is a constant delay between sampling and actuation, and that no measurements or controls are lost. These assumptions may be true as long as the controller is implemented as a single-task application in a dedicated CPU. Modern platforms, however, tend to be flexible, distributed, and reconfigurable, but at the same time also multitasking, elastic, and unreliable to some degree. For instance, control over 5G and the Cloud [2] offers the possibility of coupling low-level control loops with advanced learning algorithms, but wireless connections are susceptible to intermittent disturbances, and a remote data center may experience capacity dips during peak hours.

To understand the impact of the modern features described above, new analysis and simulation tools are needed. There is also a need to benchmark novel robust and adaptive control algorithms that can cope with time-varying computing resources. This paper presents one such new tool: JITTERTIME is a Matlab-based toolbox for performance analysis of time-varying cyberphysical systems. Combining elements of both analysis and simulation, JITTERTIME models the analog physical world and the digital filters/controllers as a set of connected linear systems driven by white noise. This allows the system performance for a given timing scenario, as measured by a quadratic cost function, to be calculated analytically. On the other hand, the timing of the execution of the digital subsystems and possible system mode switches must be carried out in a simulation. Using the tool it is possible to analyze such issues as delay and jitter due to CPU and network scheduling, lost samples or lost controls due to packet loss or execution overruns, and aperiodic behavior due to mode switches, transient failures, or unsynchronized network nodes.

The system model of JITTERTIME is largely inspired by JITTERBUG [3]. Both JITTERBUG and JITTERTIME evaluate a quadratic cost function for a mixed continuous-time/discrete-time linear system driven by white noise. The main difference is the timing model. In JITTERBUG, the timing of the discrete systems are governed by random delays with specified probability density functions. This allows the total system to be treated as a jump-linear system, and the stationary covariance can be calculated by solving a set of linear equations. In JITTERTIME, however, the timing is arbitrary and completely driven by the user. This allows for more complex timing scenarios to be analyzed, including scheduling algorithms with long-term timing dependencies and asynchronous execution in distributed control systems. For deterministic timing scenarios over a finite horizon (or a repeating hyperperiod), the performance is evaluated exactly. For stochastic timing scenarios, however, Monte Carlo simulations can be needed to obtain results with high confidence.

The timing simulation in JITTERTIME can be conducted in a script or be driven by a discrete-event simulator, such as the TRUETIME real-time control systems simulator [4]. The advantage of JITTERTIME over a full TRUETIME simulation is that the former does not require the process dynamics and disturbances to be simulated, since the control performance index is evaluated analytically.

978-1-7281-0303-7/19/$31.00 ©2019 IEEE

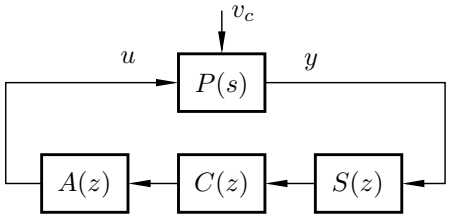

Fig. 1. A simple JITTERTIME model of a control loop with a continuous-time plant, $P(s)$, disturbed by white noise, v_c. The sampler, $S(z)$, the controller, $C(z)$, and the actuator, $A(z)$, can be executed at any points in time.

Outline

The rest of this paper is outlined as follows. The toolbox is described in Section II, followed by some theoretical background in Section III. Three use cases of the toolbox are reported in sections IV to VI, followed by a discussion of related work in Section VII and conclusions in Section VIII.

II. DESCRIPTION OF THE TOOLBOX

JITTERTIME[1] consists of a small number of functions and requires Matlab with the Control System Toolbox.

A. Creating JitterTime Models

A JITTERTIME model is created by adding and connecting any number of continuous-time and discrete-time linear systems. Throughout, multiple inputs and outputs are allowed, and a system may receive its inputs from several other systems. All noise sources in the model are assumed independent.

Continuous-time systems must be strictly proper and can be specified as state-space or transfer-function objects. Optionally, continuous-time white noise with a given intensity can be added to the system. A quadratic cost function can also be specified.

Discrete-time systems must be proper and can be specified as state-space or transfer-function objects. Optionally, discrete-time white noise with a given variance can be added to the system. When the system is executed, its inputs are read (sampled), noise is added, and its states and outputs are updated. Between executions, all states and output signals are held. A quadratic cost function can also be specified. Multiple versions of the dynamics for same system can be specified to allow for gain scheduling or other parametric behavior of the discrete-time system during simulation.

An example of a simple JITTERTIME model is given in Fig. 1. It models a sampled-data control loop with a continuous-time plant, $P(s)$, an ideal sampler, $S(z) = 1$, and discrete-time controller, $C(z)$, and zero-order hold actuator, $A(z) = 1$. When the sampler executes, it reads the measurement signal y. After executing both the controller and the actuator, the control signal u is updated and fed back to the plant. Assuming that the systems P, S, C and A and the noise and cost parameters have already been specified, the lines of code needed to construct the model are:

[1]JITTERTIME is freeware; documentation and the toolbox can be downloaded from http://www.control.lth.se/jittertime.

```
% Initialize JitterTime
N = jtInit;
% Add system 1 (P), input from system 4
N = jtAddContSys(N,1,P,4,R,Q);
% Add system 2 (S), input from system 1
N = jtAddDiscSys(N,2,S,1);
% Add system 3 (C), input from system 2
N = jtAddDiscSys(N,3,C,2);
% Add system 4 (A), input from system 3
N = jtAddDiscSys(N,4,A,3);
% Calculate the internal dynamics
N = jtCalcDynamics(N);
```

The variable `N` is a data structure that contains all the added systems. Every system is identified by a unique number. `jtCalcDynamics` checks that all system connections are correct and creates a large state-space model of the total system. Each continuous-time system requires n states, and each discrete-time system requires $n+p$ states, where n is the system order and p is the number of system outputs.

B. Simulating a JitterTime Model

The model is simulated by repeated calls to the functions `jtPassTime` and `jtExecSys`, in any order. `jtPassTime` is used to simulate the passing of time and integrates the covariance of all continuous-time systems. It also accumulates the cost of all systems. `jtExecSys` executes a given discrete-time system, which is assumed to take zero time. An optional argument can be used to control what version of the system dynamics should be applied. In the following example, the simple control loop model described above is simulated for 1000 periods of length T. The sampler executes at the start of each period, while the controller and the actuator execute after a random delay, uniformly distributed in $[0, T]$.

```
for i = 1:1000
   % Execute system 2 (S)
   N = jtExecSys(N,2);
   % Generate random delay
   delay = rand*T;
   % Pass time until control/actuation
   N = jtPassTime(N,delay);
   % Execute systems 3 and 4 (C and A)
   N = jtExecSys(N,3);
   N = jtExecSys(N,4);
   % Pass time until end of period
   N = jtPassTime(N,T-delay);
end
```

During a simulation, the internal model variables `N.P`, `N.J`, and `N.Tsim` are updated after each call to `jtPassTime` and `jtExecSys`. `N.P` contains the covariance matrix of all the states in the model. `N.J` holds the accumulated cost, and `N.Tsim` keeps track of the simulation time. All of these variables are initialized to zero in `jtCalcDynamics`. `N.J` and `N.Tsim` may be reset by the user at any time during a simulation. This can be useful for, for example, skipping the transient behavior at the start of a simulation.

C. Obtaining the Results

Depending on the purpose, the model variables `N.P`, `N.J`, and `N.Tsim` can be logged by the user during a simulation

and analyzed afterwards. If the purpose is to calculate the average cost per time unit, this can simply be done as follows:

```
Javg = N.J / N.Tsim
```

Further details about the various commands of the toolbox are available in the reference manual [5].

III. Theory

JitterTime is based on well-known theory for linear stochastic systems (e.g., [6]). The toolbox aids the user in setting up a mixed continuous/discrete linear system model driven by white noise and calculating the evolution of its total state covariance. The calculations themselves are quite trivial. At time zero, the state covariance P of the model is assumed to be zero. Between events (i.e., executions of discrete-time systems), the covariance evolves according to the matrix linear differential equation

$$\dot{P}(t) = AP(t) + P(t)A^T + R_c,$$

where A describes the total continuous dynamics and R_c is the intensity of the total continuous noise. All discrete system states are kept constant by corresponding zeros in the A matrix. When a discrete-time system k is executed at time t_k, the covariance is immediately updated according to

$$P(t_k^+) = E_{dk}P(t_k)E_{dk}^T + R_d,$$

where E_{dk} describes the discrete state transition for system k and its connection to other systems, while R_d is the variance of the discrete noise. The increase in cost between two events is given by

$$\Delta J = \int_{t_k}^{t_{k+1}} \text{tr } Q_c P(t)\, dt,$$

where Q_c is the cost matrix for the total model.

At each call to `jtPassTime`, the continuous dynamics, noise, and cost are internally sampled using the helper function `calcc2d` from JitterBug (see [3] for details). Any finite-dimensional linear system dynamics may be simulated, and the tool does not check for stability. If the model is indeed unstable, the state covariance P will grow unbounded.

A Simple Example

Consider the control loop in Fig. 1, where the process is assumed to be an integrator driven by unit-intensity white noise,

$$\dot{y}(t) = u(t) + v_c(t). \tag{1}$$

The control objective is to minimize the following cost function:

$$J(t) = \int_0^t y^2(\tau)\, d\tau. \tag{2}$$

Assuming periodic sampling with the interval T and zero delay between sampling and actuation, the stationary minimum-variance controller (see [1]) is given by the proportional feedback

$$u(t_k) = -\frac{1}{T}\,\frac{3+\sqrt{3}}{2+\sqrt{3}}\, y(t_k).$$

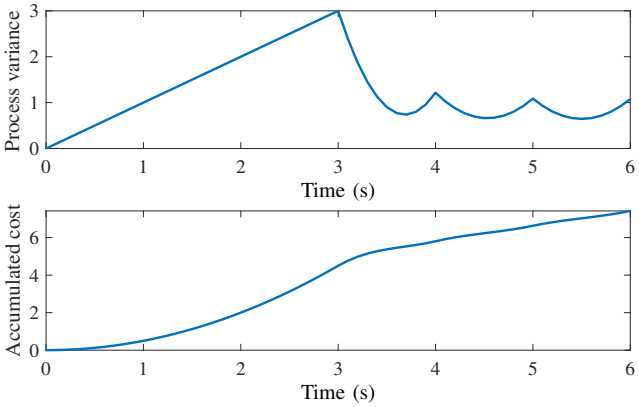

Fig. 2. Simple example where the controller is activated at $t = 3$ s and then executes once per second.

Setting $T = 1$ s and assuming the controller is activated at $t = 3$ s, the process variance, $P(t) = \mathrm{E}\, y^2(t)$, and the accumulated cost, $J(t)$, are calculated using JitterTime. The variance and cost are logged every 0.1 s to show the inter-sample behavior, and results are plotted in Fig. 2. It is seen that the process variance grows linearly when the integrator runs in open loop, as expected. When the controller is activated at $t = 3$ s, the variance decreases and soon reaches a stable periodic behavior. The average cost per time unit approaches $J_{avg} = \frac{3+\sqrt{3}}{6} \approx 0.79$, as predicted by the theory.

In the following three sections, some more complex examples of usage of the toolbox are presented.

IV. Example 1: Deadline Overrun Handling in Real-Time Control Tasks

In real-time embedded control systems, the computational resources are limited and shared between many concurrent tasks. Each real-time task, τ, is typically characterized by a minimum inter-arrival time, T, a worst-case execution time, E, and a relative deadline, D,, i.e., a maximum time limit by which the task instance (job) should be completed [7]. A scheduler is then used to arbitrate the execution of the tasks, trying to complete all jobs before their deadlines. Often it is assumed that $D = T$, so called implicit deadlines, and we will assume that here as well.

In case of a temporary overload, it may happen that some jobs are not completed by the deadline; we refer to this as a *deadline miss*. Depending on the strategy implemented in the real-time operating system, the job missing the deadline may be terminated, thus not producing the output at all, or could be allowed to continue execution and producing a late output, but potentially impacting the next jobs. Three such overrun handling methods were studied in [8], [9], see Fig. 3:

- *Kill strategy*: If a job misses its deadline it is immediately terminated and no output is produced.
- *Skip-next strategy*: If a job misses its deadline, it is allowed to continue executing but the next job is *skipped*, i.e., it is not executed at all.
- *Queue(1) strategy*: A job that misses its deadline will continue to execute until completion. Successive pending

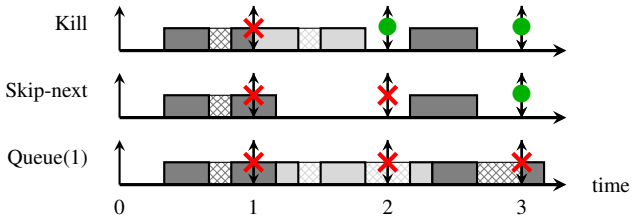

Fig. 3. Illustration of three different strategies to handle deadline misses in Example 1. A red cross at the deadline marks a job that missed that deadline (or a skipped job), while a green dot identifies a deadline hit.

jobs are appended to a queue with length 1 and executed after the completion of the previous job. The queue stores only the freshest pending job, and jobs that are removed from the queue are not executed.

If the overrunning task in question is a control task, it is clear that the overrun handling strategy can have a great impact on the control performance. Kill means that no new control signal is produced, while Skip-next and Queue(1) will delay the output until the next period (or further). Understanding exactly what the consequences will be is difficult, because of the intricate interplay between the scheduling algorithm, the overrun handling method, and the control loop. Here, JITTERTIME can be used to analyze the precise effects under various timing scenarios.

Following the system model of [9], we here consider a physical plant $P(s)$ and a control task implemented with period T and implicit deadline $D = T$, implemented on a real-time platform together with other concurrent tasks. A time-triggered I/O unit, with the same period T and synchronously released with the control task, manages the data exchange between sensors, controller and actuators. At the starting instant of each job, measurements from the sensors are copied in the platform memory, and the control command stored in the local memory is transmitted to the actuator. In nominal conditions (no deadline misses) the control task works with a fixed input-output delay equal to T. This means that even if the job finishes its execution before the deadline, the transmission of the control output is done only at the deadline instant. On the other hand, if a job does not complete its execution at the deadline instant, the actuator will be fed with the old value.

A JITTERTIME model of the above system setup is presented in Fig. 4. $P(s)$ is the continuous-time plant dynamics,

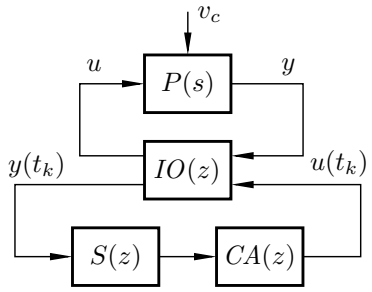

Fig. 4. JITTERTIME model of a control loop in Example 1 with time-triggered IO unit $IO(z)$ and scheduling-driven sampler $S(z)$ and controller/actuator $CA(z)$.

$IO(z) = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ represents the time-triggered IO unit, while the controller is split into the two parts $S(z) = 1$ (reading the input from the IO) and $CA(z)$ (calculating the control signal and writing the output to the IO). The model is constructed in Matlab code as follows:

```
% Create JitterTime model
N = jtInit;
% Add plant system
N = jtAddContSys(N,1,P,2,R,Q);
% Add IO unit
N = jtAddDiscSys(N,2,IO,[1 4]);
% Controller input part
N = jtAddDiscSys(N,3,S,2);
% Controller calculation and output part
N = jtAddDiscSys(N,4,CA,3);
% Calculate the internal dynamics
N = jtCalcDynamics(N);
```

Experiments

To illustrate the analysis, we study a specific scenario, where three periodic tasks should be executing on the same CPU using fixed-priority scheduling. The task set is summarized in Table I. The nominal execution time of each task is given in the table, but each instance also randomly experiences prolonged execution with 10% probability (Bernoulli distributed). The execution time is extended by 0–100% (uniformly distributed) in case of prolongation. This is a very crude model of the "tail" of the execution time distributions, capturing such phenomena as cache misses or unmodeled hardware interrupts.

Task τ_3, which is the lowest-priority task, implements a minimum-variance controller that should be used to regulate the integrator process (1) with the cost function (2). The controller is designed to compensate for a fixed input-output delay of one sampling period. The controller task period, T_3, is left as a design parameter. Choosing a small sampling period T_3 can improve the disturbance rejection but at the same time increases the risk of missed deadlines (and hence missing or delayed outputs).

To investigate how the controller cost depends on the control task period and the overrun handling method, the fixed-priority scheduling algorithm is simulated using TRUETIME [4] and the timing results of task τ_3 (the controller) are fed into JITTERTIME. (Since TRUETIME is also Matlab-based, it is in fact possible to run the JITTERTIME analysis from within TRUETIME as a co-simulation. The reader is referred to the examples supplied with the JITTERBUG toolbox for further details on how this can be implemented.)

A short example run with $T_3 = 120$ ms and the Skip-next strategy is shown in Fig. 5. The schedule reveals a large jitter

TABLE I
TASK PARAMETERS IN EXAMPLE 1.

Task	Priority	Period (ms)	Nominal exec. time (ms)
τ_1	High	80	25
τ_2	Middle	140	40
τ_3	Low	T_3	30

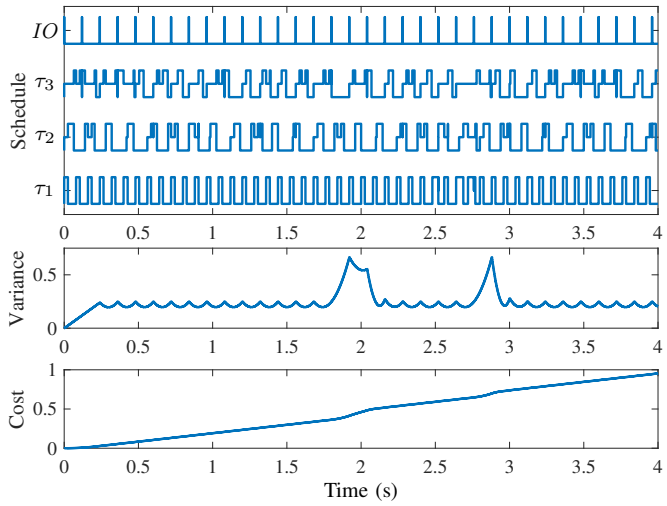

Fig. 5. Short simulation run of Example 1 with the controller period $T_3 = 120$ ms and the Skip-next overrun strategy. The overruns at $t = 1.8$–2 s (double) and $t = 2.8$ s (single) generate spikes in the process variance.

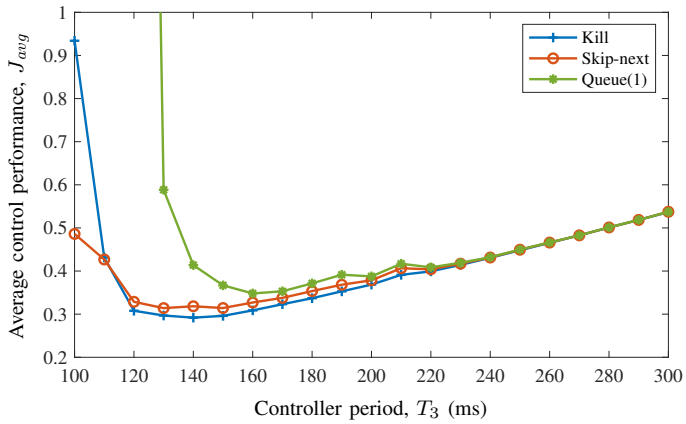

Fig. 6. Choice of controller period in Example 1. A task period of $T_3 = 140$ ms and the Kill strategy is the best design combination in this case.

for τ_3 due to the preemption from the two higher-priority tasks. Each overrun is visible as a transient increase in the variance of the controlled process, which in turn yields an increased accumulated cost.

To find the best controller period, T_3 is varied between 100 ms and 300 ms in steps of 10 ms. In each case, the scheduling algorithm and the JITTERTIME model are simulated for $T_{sim} = 1000$ time units, and the average controller cost is recorded. The performance results are reported in Fig. 6. It is seen that the Kill strategy seems to work the best overall, although it can be sensitive to multiple missed deadlines in a row. The Skip-next strategy is better at handling more severe overload, e.g., when $T_3 = 100$ ms in this example. The Queue(1) strategy performs the worst, since a missed deadline often leads to subsequent ones also being missed.

V. EXAMPLE 2: SCHEDULE OPTIMIZATION FOR CONTROL APPLICATIONS ON FOG PLATFORM

Fog computing is envisioned as an architectural means to realize the convergence of information technology and operation technology [10]. According to the OpenFog consortium, fog computing is a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things". Fog computing brings computing and deterministic communication closer to the edge of the network. Also, it virtualizes and integrates equipment such as programmable logic controllers and industrial PCs, which are used to run control applications.

A *Fog node*, being the main component of fog computing platform (FCP), has some computational resources, which are used for execution of applications with different criticality levels. The fog nodes should be configured such that not only functional but also extra-functional properties of applications are guaranteed. From a control perspective, the way the FCP is configured has an impact on the control performance of the control applications. This is especially true for the scheduling design.

In this example, we want to show the importance of proper schedule optimization on the FCP. Since control applications are executed alongside non-control applications, the scheduling algorithm should consider the extra-functional properties of the controllers, i.e., the performance of the associated control loop. The example is inspired by [11], where a quality-of-control aware algorithm for static scheduling of controllers alongside non-controllers on a FCP is proposed. The algorithm is based on a simulated annealing metaheuristic, which changes the parameters of the previous solution and finds a better one. Deadlines, offsets and activation times are the properties that the algorithm uses to find a solution. The algorithm determines the mapping of tasks to the cores as well as a static cyclic schedule for each core.

A starting point for the scheduling optimization is obtained by simulating all tasks using the earliest-deadline-first policy [7]. The scheduling algorithm then uses a cost function V to evaluate the current solution and for comparison with other solutions. Assuming a mix of control and non-control applications, the cost function can be written in the form

$$V = \sum_{i=1}^{n} D(h_i) + \sum_{j=1}^{m} (E(A_j) + O(A_j)) + \sum_{k=1}^{l} J(B_k),$$

where the cost term D checks for deadline violations in the subtasks h, E is the end-to-end response of a given application A_j, and O checks the order of execution of subtasks in a given application. These cost terms place a large penalty on the cost function V if there are any violations. Finally J represents the control performance of a given control application B_k, evaluated using JITTERTIME.

As an example, we assume that an inverted pendulum process

$$P(s) = \frac{200}{s^2 + 400}$$

should be controlled via the fog computing platform. An LQG controller with period $T = 12$ ms has been designed using JITTERBUG [3]. A summary of all the applications that should be scheduled on the FCP are given in Table. II. They are

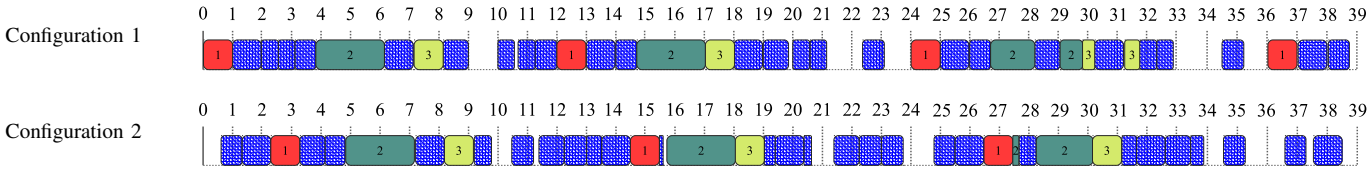

Fig. 7. Schedule configurations in Example 2. Only the first 39 ms out of the 60 ms long hyperperiod are shown.

TABLE II
CONTROL AND NON-CONTROL APPLICATIONS IN EXAMPLE 2.

Application	Type	Subtasks	Exec. time (μs)	Period (ms)
1	Ctrl	1	1000	12
		2	2500	12
		3	1000	12
2	Non-ctrl	4	500	12
3	Non-ctrl	5	1000	6
		6	500	10
4	Non-ctrl	7	750	10
		8	750	10
		9	1000	12

both control applications and non-control applications. Each application has number of subtasks and each subtask has a fixed execution time and a period. The control application consists of three subtasks: h_1 is the sensor task, h_2 is the control calculation task and h_3 is the actuation task.

During the simulated annealing execution, a large set of schedule configurations are evaluated, and for each case the controller cost $J(B_k)$ is evaluated using JITTERTIME. Two schedule tables generated by the scheduling algorithms in [11] are shown in Fig. 7. Configuration 1 represents a case where the tasks are scheduled with regards only to their deadlines, while Configuration 2 has focus on a short input-output delay on average and elimination of jitter of control tasks. We analyzed the schedule tables with JITTERTIME using the simple model given in Fig. 1. The subtasks have a hyperperiod of 60 ms. During each hyperperiod, the execution is completely deterministic, and the control cost can be evaluated exactly using JITTERTIME.

The resulting accumulated cost over a total simulation run equivalent to 10 times the hyperperiod for both configurations are shown in Fig. 8. Even though all deadlines are met in both configurations, it is seen that Configuration 2 has better control performance (smaller cost). For this application, a short input-output delay and small jitter are both critical for providing good performance. Hence, this solution will be preferred by the optimization algorithm.

VI. EXAMPLE 3: ROUTING AND SCHEDULING FOR REAL-TIME CONTROL APPLICATIONS ON ETHERNET

The problem of synthesizing network routing and scheduling for real-time control applications on time-triggered Ethernet networks has been studied in recent years [12]–[14]. Researchers have investigated various design-space exploration problems to synthesize Ethernet schedules and routes in the context of hard deadlines and worst-case latencies. The provided static route and schedule in these models result in

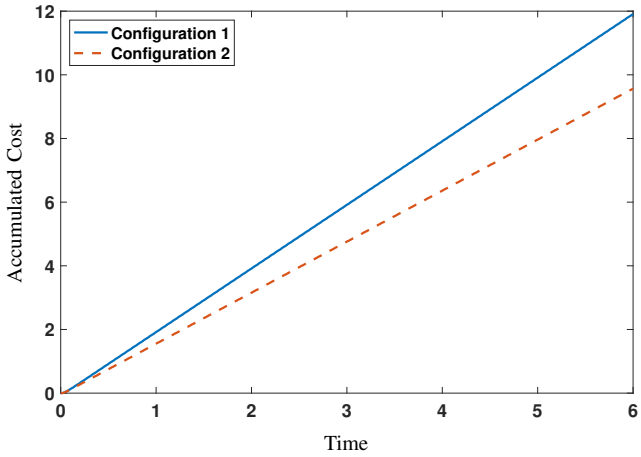

Fig. 8. Evolution of cost for the two different configurations in Example 2.

varying delays in different periods of a control application. Mahfouzi et al. [14] showed that considering only deadlines in providing route and schedule yields non-ideal timing scenarios that could potentially lead to instability of control applications. The authors used the jitter margin analysis from [15] to take into consideration average delay and jitter in providing route and schedule. In this example we show that JitterTime provides much less pessimistic results when the actual schedules are known. We consider 20 concurrent control applications that are randomly selected from a process database with inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators, represented by the following transfer functions:

$$P_1(s) = \frac{9}{s(s+1)} \qquad P_2(s) = \frac{19.6}{(s-4.43)(s+4.43)}$$
$$P_3(s) = \frac{3}{s^2} \qquad P_4(s) = \frac{9.81}{(s-3.13)(s+3.13)}$$
$$P_5(s) = \frac{3}{s^2+3}$$

These plants are considered to be representative of realistic control applications and are extensively used for experimental evaluation in the literature [1]. For each plant, an LQG controller with a given period is synthesized.

We assume that the sensors are connected through a network of 8 Ethernet TSN switches to the controllers. (See Fig. 9 for an illustration). For each control application, at the beginning of each period, the sensor samples data and sends it to its connected switch. The sampled data is served as a Time-Triggered message in the switches. Therefore, the static schedule for each message is determined using the timed gates described in

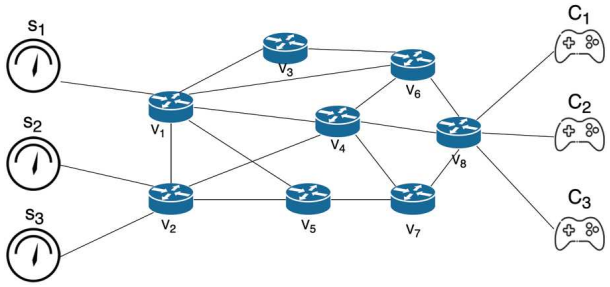

Fig. 9. Topology of Ethernet TSN network in Example 3.

TABLE III
ANALYSIS OF STABILITY AND PERFORMANCE IN EXAMPLE 3.

Application	Process	Guaranteed stability [15]	Worst-case cost [17]	JitterTime exact cost
1	P_1	Yes	1.318	1.039
2	P_2	Yes	1.005	1.002
3	P_3	Yes	1.024	1.002
4	P_4	No	6.337	1.157
5	P_5	Yes	1.136	1.008
6	P_3	Yes	1.157	1.011
7	P_4	Yes	1.187	1.024
8	P_3	Yes	1.027	1.001
9	P_4	No	29.95	1.206
10	P_4	No	∞	1.321
11	P_2	No	1.444	1.041
12	P_1	Yes	1.001	1.001
13	P_3	Yes	1.475	1.034
14	P_4	Yes	1.241	1.025
15	P_3	Yes	1.501	1.034
16	P_4	No	1.824	1.097
17	P_1	No	39.638	1.294
18	P_5	Yes	1.174	1.010
19	P_4	No	∞	4.835
20	P_2	No	∞	∞

IEEE TSN standard [16]. The routing of each message is also determined by the look-up tables in the Forwarding Engine of each switch. As a result, the sampled messages from each sensor are directed to the corresponding controller according to the static route and schedule that is hard-coded in the switches.

The static route and schedule is decided according to several constraints that comes from the characteristics of the switched fabric and the real-time constraints of the controllers, e.g. period and deadline. From the static routing and scheduling framework we extract the end-to-end delays in each period of all 20 control applications. The end-to-end delay and the jitter in turn determine the control performance of each loop.

Using the delay data from one optimization run, we investigate the stability and performance of each control loop using three different analysis tools. The results are reported in Table III. All cost values have been normalized with respect to the cost under ideal circumstances (minimum delay and zero jitter). For 8 out of the 20 control loops, stability cannot be guaranteed using Kao and Lincoln's jitter margin analysis [15]. This simple stability criterion is however only sufficient and can be very conservative. A more detailed performance and stability analysis was proposed in [17], which also enabled the computation of an upper bound of the worst-case relative performance degradation due to jitter. A value smaller than ∞ means that the loop is stable. With this tool, stability can be guaranteed for 17 of the loops. Finally we apply the JITTERTIME analysis to each control loop and compute the exact value for the relative performance degradation due to jitter given the entire distributed system schedule over the hyperperiod. We can now conclude that all loops are indeed stable except the last one (which is indeed unstable even for zero jitter), and the performance degradation is very modest in most cases. In the example we have seen that, by analyzing a particular timing scenario, we can get more detailed answers with regard to both stability and performance.

VII. RELATED WORK

The linear stochastic state-space analysis utilized in the toolbox hails back to Kalman's seminal work on optimal filtering [18]. Further background on stochastic filtering and control processes can be found in, e.g., [6], [19]. Linear covariance analysis for time-varying systems has frequently been applied in the field of space navigation, see, e.g., [20].

Co-simulation of real-time/embedded/networked control systems has been an active research topic for the past twenty years. Often the simulators have been implemented as Simulink libraries to allow integration with already existing plant and control system models. The aforementioned TrueTime simulator [4] has a focus on task scheduling but also includes simple models of wired and wireless networks. PiccSim [21] allows detailed wireless network models (using ns-2) to be co-simulated with the feedback control system. T-Res [22] is another recent tool that focuses on the scheduling of shared resources in embedded systems. JITTERTIME can be coupled to each of the simulators mentioned above, or to a pure discrete-event simulator such as SimEvents [23].

It is well known that direct covariance calculations as performed in the toolbox are not numerically robust for systems of very large dimension; there it is better to use a square-root representation or UD factorization [19]. This will be implemented in a future version of the toolbox.

VIII. CONCLUSION

Real-time control systems implemented on top of modern computing platforms are very complex systems. In this paper we have presented the toolbox JITTERTIME, which facilitates fast analysis of simple linear control systems under arbitrarily complex timing patterns. The analysis is fast enough to be used for instance within a larger optimization framework for synthesis of distributed real-time control systems. As always, there is a balance between the analytical power of a tool and its generality and applicability. A detailed full-system simulator such as TRUETIME [4] can arbitrarily complex process and timing models but does not give any performance guarantees. A purely analytical tool such as the jitter margin [15] can give stability guarantees but is very limited in its modeling capabilities. JITTERTIME strikes a middle path by joining linear stochastic systems analysis and arbitrary timing simulation. Three different examples of how the toolbox can be used in the analysis and design of complex real-time control

systems were given. Further examples of how the toolbox can be used are found in the reference manual [5].

ACKNOWLEDGMENTS

A. Cervin and R. Mahfouzi are part of the ELLIIT Excellence Center at Linköping–Lund in Information Technology. A. Cervin and M. Barzegaran are part of the Nordic Hub on Industrial IoT, funded by NordForsk. The work was partially supported by the European Union Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant No. 764785, FORA–Fog Computing for Robotics and Industrial Automation.

REFERENCES

[1] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems (3rd Ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[2] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Towards mission-critical control at the Edge and over 5G," in *IEEE International Conference on Edge Computing (EDGE)*, July 2018.

[3] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proc. 41st IEEE Conference on Decision and Control*, Las Vegas, NV, 2002.

[4] D. Henriksson, A. Cervin, and K.-E. Årzén, "TrueTime: Simulation of control loops under shared computer resources," in *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, Jul. 2002.

[5] A. Cervin, "JitterTime 1.0—Reference manual," Department of Automatic Control, Lund University, Sweden, Tech. Rep. TFRT-7658, 2019.

[6] K. J. Åström, *Introduction to Stochastic Control Theory*. New York: Academic Press, 1970.

[7] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer Publishing Company, Inc., 2011.

[8] A. Cervin, "Analysis of overrun strategies in periodic control tasks," in *16th IFAC World Congress*, 2005.

[9] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin, "DMAC: Deadline-miss-aware control," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, Stuttgart, Germany, 2019.

[10] W. Steiner and S. Poledna, "Fog computing as enabler for the industrial internet of things," *Elektrotechnik Und Informationstechnik*, vol. 133, no. 7, pp. 1–5, 2016.

[11] M. Barzegran, A. Cervin, and P. Pop, "Towards quality-of-control-aware scheduling of industrial applications on fog computing platforms," in *Workshop on Fog Computing and the IoT (IoT-Fog19)*. ACM, 2019.

[12] D. Tămaş-Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-Time Systems*, vol. 51, no. 1, pp. 1–35, 2015.

[13] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. ACM, 2017, pp. 8–17.

[14] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in ethernet networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 682–687.

[15] C.-Y. Kao and B. Lincoln, "Simple stability criteria for systems with time-varying delays," *Automatica*, vol. 40, no. 8, pp. 1429–1434, 2004.

[16] LAN/MAN Standards Committee of the IEEE Computer Society, *IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks Amendment 25 : Enhancements for Scheduled Traffic, IEEE Std. 802.1Qbv-2015*, 2015.

[17] A. Cervin, "Stability and worst-case performance analysis of sampled-data control systems with input and output jitter," in *2012 American Control Conference (ACC)*. IEEE, 2012, pp. 3760–3765.

[18] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transaction of the ASME-Journal of Basic Engineering*, pp. 35–45, March 1960.

[19] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. New Jersey: Prentice Hall, 2000.

[20] N. B. Stastny and D. K. Geller, "Autonomous optical navigation at Jupiter: A linear covariance analysis," *Journal of Spacecraft and Rockets*, vol. 45, no. 2, pp. 290–298, 2008.

[21] T. Kohtamaki, M. Pohjola, J. Brand, and L. M. Eriksson, "PiccSIM Toolchain: Design, simulation and automatic implementation of wireless networked control systems," in *2009 International Conference on Networking, Sensing and Control*, March 2009, pp. 49–54.

[22] F. Cremona, M. Morelli, and M. Di Natale, "TRES: A modular representation of schedulers, tasks, and messages to control simulations in Simulink," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1940–1947.

[23] M. Clune, P. Mosterman, and C. Cassandras, "Discrete event and hybrid system simulation with SimEvents," in *Proceedings of the 8th international workshop on discrete event systems*, 2006, pp. 386–387.