

Self-Triggered Networked Control Systems: an Experimental Case Study

Antonio Camacho, Pau Martí, Manel Velasco,
Camilo Lozoya, Ricard Villà and Josep M. Fuertes
Automatic Control Dept., Technical University of Catalonia
Pau Gargallo 5, 08028 Barcelona, Spain
Email: antonio.camacho.santiago@upc.edu

Eulàlia Grifol
Dept. Statistics and Operational Research
Technical University of Catalonia
c/ Colom 11, 08222 Terrassa, Spain
Email: eulalia.grifol@upc.edu

Abstract—A self-triggered controller is characterized, in general, by a non-periodic sequence of job activations. And each job execution, apart from performing sampling, control algorithm computation and actuation, calculates the next job activation time as a function of the plant state. This paper describes the implementation of self-triggered controllers in networked control systems (NCS). The implementation corroborates that self-triggered control can be used for minimizing bandwidth utilization while providing similar control performance than periodic controllers.

I. INTRODUCTION

NCS [1], i.e. control loops closed over communication networks where sensors, controllers and actuators are physically distributed and exchange control data through the network, are gaining increased attention in many control application areas including for example motors and converters [2], [3], while the underlying required control theory is starting to offer mature and methodological results [4]. The most common design and implementation approach for NCS consists in the periodic execution of the control algorithm [5]. This approach makes a static use of the communication bandwidth regardless of the current load in the network and/or changes in the systems that are being controlled.

To overcome this periodicity limitation, two new trends for the analysis and design of NCS can be identified. The first one is to apply rate adaptation techniques where the period is selected according to the controlled system dynamics and/or to the bandwidth conditions, e.g. [6], [7], [8]. The main goal of these approaches is to improve the aggregated control performance for the set of control loops by efficiently using all the communication bandwidth. The second trend is to apply event-based sampling techniques which produce non-periodic executions of the control algorithm, and therefore, non-periodic messaging in the network, e.g. [9], [10], [11]. Its main goal is to minimize the bandwidth utilization while still guaranteeing stability and acceptable control performance.

In particular, the approach presented in [11] is based on the self-triggered control paradigm that was first introduced in [12]. A self-triggered controller, at each job execution, determines *when* the next job execution should occur. Focusing on

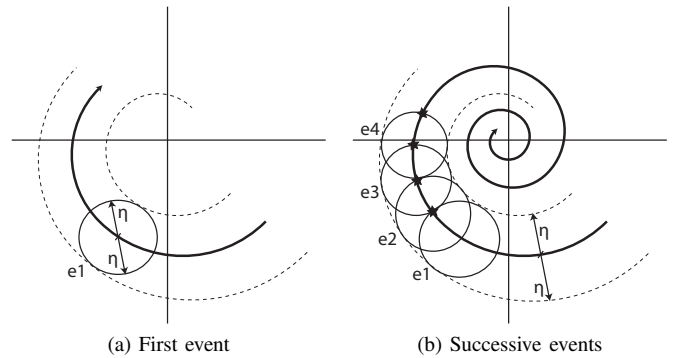


Fig. 1: The self-triggered control system concept

self-triggered controllers, this paper presents an experimental case study on the implementation of these type of controllers in NCS. The experiment i) shows the feasibility of implementing self-triggered controllers when control loops are closed over a network, ii) permits to identify the main software tasks implemented in each node, and iii) corroborates that self-triggered control provides similar control performance than periodic control while minimizing the number of messages, i.e., the required bandwidth.

The rest of this paper is organized as follows. Section II introduces the system model. Section III describes the experimental setup, with emphasis on the hardware, controlled plants, and software used in the experiment. Section IV explains the self-triggered control approach implemented in the NCS, as well as the periodic control strategy implemented for comparative purposes. Section V summarizes the experimental results. Finally, Section VI concludes the paper.

II. SYSTEM MODEL

A. Self-triggered Control Model

For various types of event-driven control approaches, the activation of controller jobs occurs when the system trajectory crosses *boundaries* that define a discretization in the state-space domain. These boundaries delimit the regions where, with some tolerated threshold, no control action is required.

The conceptual operation of an event-driven controller is illustrated in Figure 1. In both figures, the axes represent the

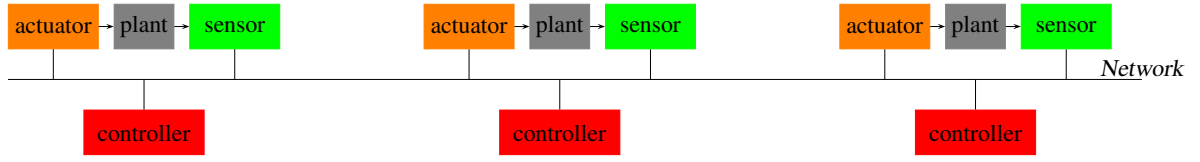


Fig. 2: Networked control system scheme

state-space domain. The thick curve represents the closed-loop system trajectory and circles represent boundaries. In Figure 1(a), from the sampled state (center of the circle), the control action is executed and a boundary is set with a tolerated error η . When the trajectory crosses the boundary, the state (plant) is sampled, a new control action is applied, and a new boundary is set. The sequence of sampled states (represented by little stars) and associated boundaries is represented in Figure 1(b).

A priori, the implementation of such systems would require using specific hardware for detecting the event condition “trajectory crosses the boundary” that would trigger the execution of each control job. However, as shown in [13], [14], [15], for these type of self-triggered controllers, at each job execution, it is possible to compute the time at which the system trajectory will cross the boundary. Therefore, no specific hardware is required. Next we formalize these concepts.

The plant to be controlled is modeled by the following continuous-time linear control system

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t)\end{aligned}\quad (1)$$

with $x \in \mathbb{R}^{n \times 1}$, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $u \in \mathbb{R}^{m \times 1}$, and $C \in \mathbb{R}^{1 \times n}$. Given the feedback matrix $L \in \mathbb{R}^{m \times n}$, let

$$u(t) = u_k = Lx(a_k) = Lx_k \quad \forall t \in [a_k, a_{k+1}[\quad (2)$$

be the control updates given by a linear feedback controller designed in the continuous-time domain using only samples of the state at the instants $a_0, a_1, \dots, a_k, \dots$. Between two consecutive control updates, $u(t)$ is held constant. In periodic sampling we have $a_{k+1} = a_k + h$, where h is the period of the controller. However here we do not set this limitation.

Let $e_k(t) = x(t) - x_k$ be the error evolution between consecutive samples with $t \in [a_k, a_{k+1}[$. For several types of event-driven control approaches, the event condition that triggers the sampling can be generalized by introducing a function $f(\cdot, \cdot, \Upsilon) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ that defines a boundary measuring the tolerated error with respect to the sampled state [15]. The condition that must be ensured is

$$f(e_k(t), x_k, \Upsilon) \leq \eta \quad (3)$$

where η is the error tolerance and $\Upsilon = \{v_1, v_2, \dots, v_p\}$, $v_i \in \mathbb{R}$ is a set of free parameters of f . Hence, we can define the complete dynamics of the event-driven system by the $n + 1$ order non linear discrete-time system

$$\begin{aligned}a_{k+1} &= a_k + \Lambda(x_k, \Upsilon, \eta) \\ x_{k+1} &= (\Phi(\Lambda(x_k, \Upsilon, \eta)) + \Gamma(\Lambda(x_k, \Upsilon, \eta))L)x_k\end{aligned}\quad (4)$$

where $\Lambda(x_k, \Upsilon, \eta)$ denotes the time separation between two consecutive activations a_{k+1} and a_k , assuming that $x_k = x(a_k)$ is the state sampled at a_k , and considering that $\Phi(t) = e^{At}$ and $\Gamma(t) = \int_0^t e^{As} ds B$. When an expression for $\Lambda(x_k, \Upsilon, \eta)$ solving (1), (2), and (3) can be found, the self-triggered mechanism is defined.

Looking at Figure 1, $\Lambda(x_k, \Upsilon, \eta)$ is the time that will take the system trajectory from the sampled state to the point where it crosses the boundary. And it has to be computed at each job execution.

B. NCS Model

The networked control system considered in this paper is illustrated in Figure 2. Several control loops, each one formed by a sensor, a controller and an actuator implemented in physically separated nodes, share a network to exchange the control data required for each control loop operation.

For a given networked closed loop system, each control loop operation would basically require sending the sensor reading from sensor to controller (sensor message) after sampling the plant, and sending the control signal from controller to actuator (control message) after computing its value in the controller node using the information contained in the incoming sensor message. Thus, in terms of bandwidth utilization, each control job simplifies to sending two messages, the sensor and the control message.

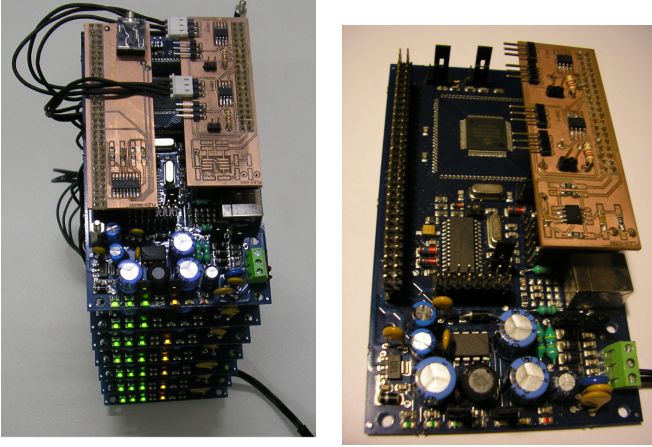
III. EXPERIMENTAL SETUP

As a prototype and performance demonstration, a proof-of-concept implementation consisting on the networked control of three plants, each of the form of a double integrator electronic circuit, has been built. The network is the Controller Area Network [16] that provides the basis for many cost-effective distributed embedded systems.

A. Hardware

The experimental setup is shown in Figure 3a. It consists on a tower of seven boards. Each board is a Flex board [17] equipped with a dsPIC micro-controller running the Erika [17] real-time kernel. Specifically, the micro-controller is a dsPIC33FJ256MC710 equipped with two CAN ports [18].

In Figure 3a, the three boards at the bottom are the controllers, and the following three boards are the sensors/actuators. All of them are connected with a single CAN bus. Though for simplicity each pair of sensor/actuator share the same physical board, any control loop operation requires using the CAN network as specified, that is, it requires sending



(a) Networked tower (b) A sensor/actuator board

Fig. 3: Implementation setup

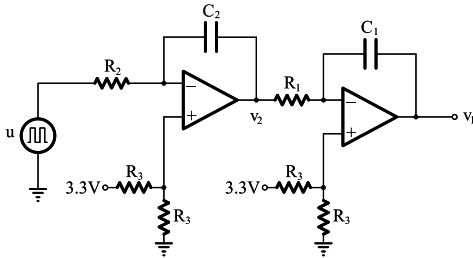


Fig. 4: Electronic double integrator circuit

the sample from a sensor/actuator board to a controller board, and sending the control signal from the controller board back to the sensor/actuator node. The board in the top is used for debugging purposes and is capable of extracting data from the tower to an external PC equipped with Matlab[®] via RS232 link. Figure 3b shows one single sensor/actuator board. It can be observed that it has plugged in an small board that contains a double integrator electronic circuit, as well as, CAN transceivers. All boards are also connected using a second CAN bus that is used for communicating debugging data.

B. Plants details

Each plant is an electronic double integrator, as illustrated in Figure 4. Note that in the integrator configuration, the operational amplifiers require positive and negative input voltages. Otherwise, they will quickly saturate. However, since the circuit is powered by the dsPIC, and thus no negative voltages are available, the 0V voltage (V_{ss}) in the non-inverting input has been shifted from GND to half of the value of V_{cc} (3.3V) by using a voltage divider R_3 . Therefore, the operational amplifier differential input voltage can take positive or negative values. The electronic components nominal values are shown in Table I.

An integrator implemented by an operational amplifier is

Component	Nominal value	Validated value
$R_{1/2}$	1 k Ω	1 k Ω
R_1	100 k Ω	100 k Ω
R_2	100 k Ω	100 k Ω
C_1	470 nF	420 nF
C_2	470 nF	420 nF

TABLE I: Electronic components nominal and validated values

modeled by

$$V_{\text{out}} = \int_0^t -\frac{V_{\text{in}}}{RC} dt + V_{\text{initial}} \quad (5)$$

where V_{initial} is the output voltage of the integrator at time $t = 0$, and V_{in} , V_{out} are the input and output voltages of the integrator, respectively.

From (5), and the scheme shown in Figure 4, the double integrator plant dynamics can be modeled by

$$\frac{dv_2}{dt} = \frac{-1}{R_2 C_2} u, \quad \frac{dv_1}{dt} = \frac{-1}{R_1 C_1} v_2.$$

or equivalently, following the notation of (1), as

$$\begin{bmatrix} \dot{v}_1 \\ \dot{v}_2 \end{bmatrix} = \begin{bmatrix} 0 & \frac{-1}{R_1 C_1} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{-1}{R_2 C_2} \end{bmatrix} u \quad (6)$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

Knowing the existing tolerances in the electronics components (5% for resistors and 25% for capacitors), the mathematical model used for controller design and simulation that best matches the electronic circuit dynamics is given by (6) with the validated values listed in table I. These values have been chosen after performing the following validation. A standard control algorithm with a sampling period of $h = 50$ ms has been applied to the plant as well as to the model (6) with the components nominal values. By comparing the theoretical results obtained from a Matlab[®] model with those obtained from the plant, the model values have been tuned until the simulated dynamics and those from the plant are the same. With the validated values for the components, the model is

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & -23.809524 \\ 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -23.809524 \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \end{bmatrix} x. \end{aligned} \quad (7)$$

Note that the plant is unstable because the eigenvalues of the system matrix are $\lambda_{1,2} = 0$.

The goal of the controller is to make the circuit output voltage (v_1 in Figure 4) to track a reference signal by giving the appropriated voltage levels (control signals) u . Both states v_1 and v_2 are read via the analog-to-digital converter (ADC) port of the micro-controller and u is applied to the plant through the pulse-width-modulation (PWM) port.

C. Software organization

Table II details the main software tasks in each board. It has to be pointed out that other tasks besides those listed in Table II have been also coded in each board for debugging purposes.

Node	Name of task	Type	Comments
Sensor/Actuator	Read_State	RT TASK	Read ADC
	Calculate_Next_Activation	RT TASK	Computes the next activation time
	CAN_Send	function	Sends sensor messages
	Send_Sample	RT TASK	Samples the plant, computes next activation time, and sends sensor message
	CAN_Receive1	RX Interrupt	Receive CAN controller message and reset synchronized actuation
	Apply_Control	RT TASK	Update PWM
Controller	Reference_Change	RT TASK	Generates perturbations in terms of reference changes
	CAN_Receive2	RX Interrupt	Receive CAN sensor message
	Compute_Control_Send	RX Interrupt	On reception of a sensor message, computes the control signal, and sends the controller message
	CAN_Send	function	Sends controller messages

TABLE II: Software tasks in each board

At each sensor/actuator board, the task period for the `Send_Sample` RT TASK is constant (h) for periodic control or varies for self-triggered control. In the latter case, the task resets its application timer (or alarm) with the calculated next activation time. The `Read_State` is not periodic but activated by the `Send_Sample` task. The task `Calculate_Next_Activation` is also not periodic and it is activated by the `Send_Sample` task in the self-triggered implementation strategy. The period for the `Reference_Change` RT TASK is several orders of magnitude higher than h . However, it is adjustable to permit set-point changes at random start times.

At each controller board, the `Compute_Control_Send` is activated by the reception of an upcoming sensor message. The two tasks `CAN_Receive1` and `Apply_Control` are further explained in next subsection.

The CAN communication is achieved by the `CAN_Send` functions which assemble each message and place them for transmission, and by the `CAN_Receive1/2` that are CAN interrupts handlers for reading the incoming messages.

Each circuit is constantly affected by perturbations in terms of step set-point changes. They occur at random times (within a given interval) in order to capture all possible different scenarios.

IV. CONTROL DESIGN

A. Preliminary Considerations

It is well known that a network within a loop introduces sampling-to-actuation (s/a) delays [1]. In the theoretical approach, delays were omitted to focus only on the self-triggered concept. However, varying s/a delays have to be considered, otherwise, performance degradation occurs.

To solve this problem, the approach adopted in this paper is to apply control updates at the actuator node at synchronized instants provided that synchronization between networked nodes is implemented. Synchronized actuation instants requires activating the `Apply_Control` task in the actuator node at a constant time with respect to the corresponding sampling time in such a way that s/a delays can be treated as being constant.

To do so, for a given bus baudrate, an upper bound to the s/a delays that appear in the implementation can be inferred from direct measurements, namely τ^* . Then, each time the `Send_Sample` task executes, the sampling time at the sensor node is recorded, namely t_k^s , and sent in the sensor message. The controller, after computing the control signal, sends it to the actuator node together with the sampling time. Upon reception at the actuator node by the task `CAN_Receive1`, the current time is recorded, namely t_k^a , and the `Apply_Control` task is programmed to fire at time $t_k^s + \tau^* - t_k^a$, time at which the control signal is applied to the plant, thus forcing a constant s/a delay.

The adopted strategy for dealing with varying time delays imposes a constant s/a delay within each control loop. In the implementation results presented next, the CAN bus baudrate was set to 1Mbps. For this baudrate, the measured s/a delay was very short, 0.26ms, and therefore, controller gains were computed considering the delay negligible.

Finally, it has to be stressed that for tracking step set-point changes, the control input specified in (2) is complemented by the N_x matrix that transforms the reference step r into a reference vector, and by the N_u matrix that is used for eliminating the error in steady state [19], as follows

$$u(t) = u_k = L(r_k N_x - x_k) + r_k N_u \quad t \in [a_k, a_{k+1}[.$$

In particular, for our plants, $N_x = [1 \ 0]^T$ and $N_u = 0$.

B. Periodic Control

The periodic control is designed using the standard linear quadratic regulator (LQR) method for discrete time systems [19] to minimize the continuous-time quadratic cost function

$$J_c = \int_0^\infty x(t)^T Q_{c1} x(t) + u(t)^T Q_{c2} u(t) dt. \quad (8)$$

In particular, the weighting matrices are the identity and the sampling period is $h = 20$ ms. With these settings, the control gain for each controller is $L = [0.6739 \ -1.3424]$.

C. Self-triggered Control

The self-triggered controller is designed in continuous time according to the model given in Section II-A where the

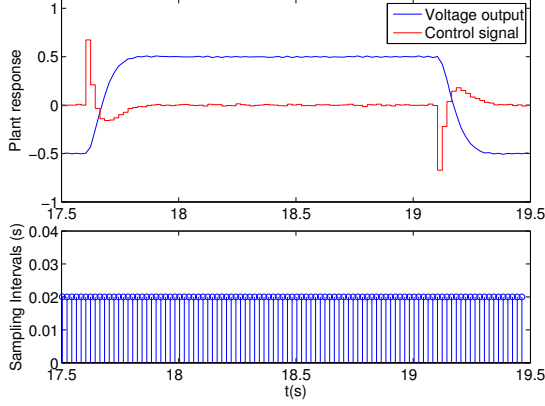


Fig. 5: Periodic controller response (top) and sampling periods (bottom).

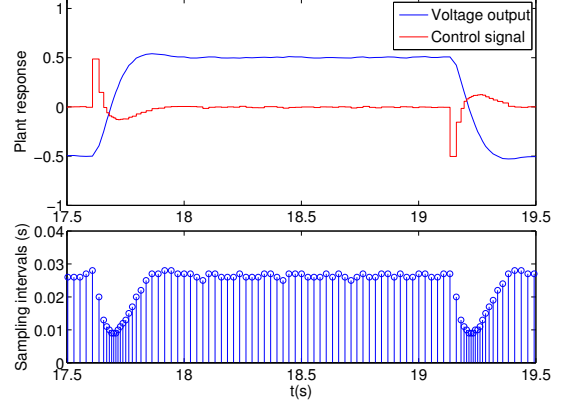


Fig. 6: Self-triggered controller response (top) and sampling periods (bottom).

sequence of sampled states obeys the event condition (3) given by

$$[x_{k+1} - x_k]^T M_1 [x_{k+1} - x_k] = \eta x_k^T M_2 x_k \quad (9)$$

with matrices M_1 and M_2 being the identity, and $\eta = 0.1$. Quadratic functions for event conditions of the form (9) are a typical choice [11], [14]. In addition, in [15] it was shown that for these type of conditions, an explicit solution to the problem of calculating the next activation time exists. In particular, from the event condition (9) and taking into account its parameters, the next activation time is the positive t of

$$t = \frac{\sqrt{-4[(A - BL)x_k]^T [(A - BL)x_k] (-\eta)x_k^T x_k}}{2[(A - BL)x_k]^T [(A - BL)x_k]}. \quad (10)$$

With these settings, the control gain for each controller is heuristically set to $L = [0.5 \quad -1.35]$.

V. EXPERIMENTAL RESULTS

This section describes the main results obtained using the periodic approach and the self-triggered approach.

A. Evaluation Metrics

Control performance for the three controllers is evaluated in terms of aggregated cost by adding for each networked control loop the continuous counterpart of the discrete cost function used for controller design,

$$J_{eval} = \sum_{i=1}^3 J_{c,i}. \quad (11)$$

In particular, each J_c was calculated taking values for x and u every 10ms, and accumulating the integral over intervals of 10ms as in

$$J_c = \sum_{j=0}^{t_{exp}} \int_{10j}^{10(j+1)} x^T(t) Q_{c1} x(t) + u^T(t) Q_{c2} u(t) dt$$

where the integral is computed using trapezoidal numerical integration, and t_{exp} is the experiment time.

The bandwidth utilization is evaluated by counting the number of messages required by each control strategy.

B. Evaluation

Figures 5 and 6 show for each implementation strategy and for a given plant, the type of dynamics that are achieved (top of both subfigures) and the activation patterns applied (bottom of both subfigures). In particular, for the bottom subfigures, the x -axis is time (note that only 2s are displayed), and the y -axis is the sampling interval in milliseconds. Each job activation time is represented by a vertical line, whose height indicates the next job activation time. Hence denser activations occur when the heights are shorter. As it can be seen in figure 5, the sequence of sampling intervals for the periodic strategy is constant, as it should be. However, as shown in figure 6, the sequence of sampling intervals vary between 0.010s and 0.030s.

Although not shown, in the self-triggered strategy, the type of activation pattern shown in figure 6 (bottom) repeats at each reference change. Hence, at each new set-point and during the transient, the self-triggered controller tends to apply shorter sampling intervals. While in the steady state the sampling intervals stabilize around 0.028s. This has a double benefit: it tends to correct errors using a fast sampling rate while it saves bandwidth when no errors are present.

Comparing the dynamics shown in Figures 5 and 6, no big difference can be appreciated. It is interesting to look at the control signal because it also reflects the fact that in the self-triggered strategy sampling periods vary.

Figures 7 and 8 provide a complete view of the network traffic generated by the three control loops for the periodic and self-triggered strategy, respectively, during 1.5s. The CAN traffic was captured using a USB-to-CAN sniffer module [20]. Each vertical bar corresponds to a pair of sensor/controller messages. As expected, for the periodic case, each plant generates periodic pairs of sensor and control messages. However, the self-triggered strategy generates non-periodic pairs

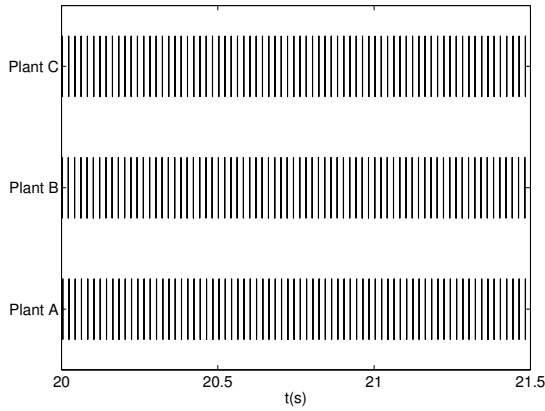


Fig. 7: Network traffic for the periodic strategy

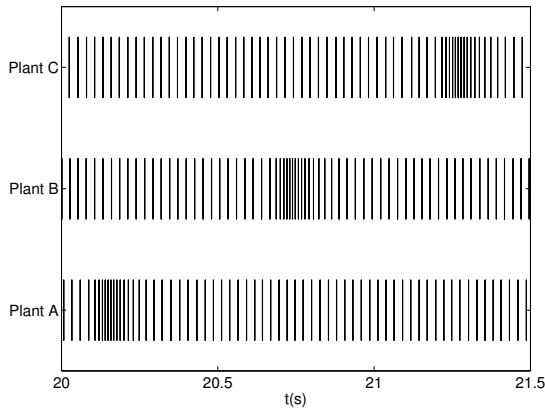


Fig. 8: Network traffic for the self-triggered strategy

of sensor and control messages. In particular, in figure 8, each time the density of messages increases marks a set point change for the corresponding plant.

To summarize the evaluation, for an experimental run of 1 minute, table III shows for both strategies the accumulated cost for the three control loops as specified in (11), the number of required messages to accomplish the control, and the average sampling interval. As it can be seen, while the control cost is very similar, the number of messages required by the self-triggered implementation is, by far, lower than those required by the periodic implementation.

VI. CONCLUSIONS

This paper has discussed the implementation of self-triggered controllers in networked control systems. The main conclusion that can be drawn is that self-triggered control is a serious candidate design strategy for minimizing the communication bandwidth required by each control loop in resource-constrained control systems. Future work will cope with two main problems detected in the implementation: when noise affects the plant and/or when the sensor resolution is

Strategy	Accumulated cost	Number of messages	Average sampling interval
Periodic	11.38	18000	0.020s
Self-triggered	11.75	15384	0.024s

TABLE III: Performance evaluation summary

limited, the computed next activation times differs from those obtained from theory.

REFERENCES

- [1] Y. Tipsuwan and M. Y. Chow, "Control methodologies in networked control systems", *Control Engineering Practice*, Vol. 11, pp. 1099-1111, 2003
- [2] S.K. Mazumder, M. Tahir, and K. Acharya, "MasterSlave Current-Sharing Control of a Parallel DCDC Converter System Over an RF Communication Interface," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 1, pp. 59-66, Jan 2008.
- [3] H. Li, M.-Y. Chow, and Z. Sun, "EDA-Based Speed Control of a Networked DC Motor System With Time Delays and Packet Losses," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 5, pp. 1727-1735, May 2009.
- [4] D. Nešić and D. Liberzon, "A unified framework for design and analysis of networked and quantized control systems," *IEEE Transactions on Automatic Control*, Vol. 54, No. 4, pp. 732-747, April 2009.
- [5] D. Hristu-Varsakelis and W. S. Levine, *Handbook of Networked and Embedded Control Systems*, Birkhäuser Boston, June, 2008.
- [6] Rehbinder, H. and Sanfridson, M., "Scheduling of a limited communication channel for optimal control," *Automatica*, vol. 40, n. 3, pp. 491-500, March 2004.
- [7] Ben Gaid, M.E.M.; Cela, A.; Hamam, Y., "Optimal integrated control and scheduling of networked control systems with communication constraints: application to a car suspension system," *IEEE Transactions on Control Systems Technology*, vol.14, no.4, pp. 776-787, July 2006.
- [8] M. Velasco, P. Martí, R. Castañé, J. Guardia and J.M. Fuertes, "A CAN Application Profile for Control Optimization in Networked Embedded Systems", *32th Annual Conf. of the IEEE Industrial Electronics Society*, Nov. 2006.
- [9] Hristu-Varsakelis, D., and Kumar, P.R., "Interrupt-based feedback control over a shared communication medium," *41st IEEE Conference on Decision and Control*, Dec. 2002.
- [10] X. Wang and M. Lemmon, "Decentralized Event-triggering Broadcast over Networked Systems," *Hybrid Systems: Computation and Control*, 2008.
- [11] A. Anta and P. Tabuada, "On the benefits of relaxing the periodicity assumption for networked control systems over CAN," *Real Time Systems Symposium*, December 2009.
- [12] M. Velasco, P. Martí and J.M. Fuertes, "The Self Triggered Task Model for Real-Time Control Systems," in *WiP of the 24th IEEE Real-Time Systems Symposium*, Decembre 2003
- [13] A. Anta and P. Tabuada, "Self-triggered stabilization of homogeneous control systems," in *2008 American Control Conference*, 2008.
- [14] X. Wang, and M. Lemmon, "Self-triggered Feedback Control Systems with Finite-Gain L2 Stability" *IEEE Transactions on Automatic Control*, v. 45, n. 3, pp. 452-467, March 2009.
- [15] M. Velasco, P. Martí, and E. Bini, "Control-driven Tasks: Modeling and Analysis," in *29th IEEE Real-Time Systems Symposium*, Barcelona, Spain 2008.
- [16] CAN Specification version 2.0. Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart, 1991.
- [17] Evidence S.r.l., Flex full base board, Erika real-time kernel, <http://www.evidence.eu.com>, [Online accessed 1-October-2009].
- [18] Microchip dsPIC33FJ256MC710, <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en024663>, [Online accessed 1-October-2009].
- [19] K.J. Åström and B. Wittenmark, *Computer-Controlled Systems*, Third Edition, Prentice-Hall, 1997.
- [20] USB-to-CAN module, IXXAT, http://www.ixxat.com/usb-to-can-compact-interface_en.html, [Online accessed 1-October-2009].