Brief Industry Paper: Delay-Aware Control in Networked Systems Using Smart Actuators

Paolo Pazzaglia, Christoph Mark, Behnaz Pourmohseni, Fedor Smirnov, Kevin Schmidt and Laura Beermann
Robert Bosch GmbH, Corporate Research. Emails: *name.surname@de.bosch.com*

***Abstract*—The control applications of the future will increasingly rely on edge and cloud services to access enhanced computational resources, boosting efficiency and optimizing performance. Such networked systems are however sensitive to variable delays, which are inevitably introduced by the communication between the plant and the remote controller. To increase reliability, we present a delay-aware control design, based on the execution of multiple control modes optimized for different latency conditions, paired with a smart actuator that measures the control chain delay at runtime. The delay information is used by the actuator to select the best suited control input for the current delay, resulting in a higher performance of the control application. Additionally, the measured delay can be used in feedback to adjust the controller configuration for the subsequent iterations.**

I. Introduction

Distributing functionalities of systems over cloud and/or edge brings many advantages, such as a relatively cheap access to powerful and elastic computational resources, eased combination of information, centralized maintenance, and more flexible allocation. This design is already popular across multiple application domains such as streaming services, business, scientific computing, etc. On the other hand, the case of *safety-critical* applications executed over cloud/edge nodes, such as *networked control systems*, was mostly relegated to the domain of pure research. Enhanced connectivity solutions are only recently being explored for industrial control applications with real-time deadlines, e.g., in future factory or autonomous driving scenarios [1]. The push comes from the need of implementing complex control-related algorithms, such as Model Predictive Controllers (MPCs), object detection, or path planning, without the restrictions of embedded platforms.

The adoption of edge/cloud infrastructure comes nonetheless with its own challenges. The network, required to communicate between the control algorithm and the controlled plant, typically introduces uncertain communication delays [2], especially in complex environments and if wireless communication is involved. Moreover, to optimize costs, the distributed resources may be shared with other applications, widening the range of experienced delays. The resulting variable latency in the sensor-control-actuator chain could be detrimental for the stability and performance of the control systems [3].

To make the best of the proven advantages, while overcoming such limitations, a strong push towards making these distributed systems truly *reliable* is growing in the industrial world (see e.g., [1], for an overview of the potential of Reliable Distributed Systems). Multiple control designs exist in literature, mostly relying on the assumption that a probabilistic or worst-case model of the delays is available offline [2] and with often only stochastic robustness guarantees. However, a networked system for safety-critical applications is expected to work reliably in all delay conditions. In this context, knowing *in advance* the delay experienced in the control chain could open the door to better performing control designs, but in many use-cases a fail-safe knowledge or prediction of the latency is not realistically available. The most effective way to determine with certainty the chain delay is measuring it when the actuator has received the data, which unfortunately occurs only *after* the control input has been calculated.

In this paper, we present a piece of our work in progress for the connected industrial applications of the future. In our vision of a cloud/edge-enhanced distributed control system (see Figure 1), the actuator (on a low-end local node) features a limited amount of intelligence to work jointly with the remote control process (deployed on a powerful remote node). The distributed platform implements a *delay-aware* control strategy with multiple modes that are run jointly, each providing a control input that is optimized for a specific interval of sensor-to-actuator latency. All resulting inputs are then sent to the actuator. The delay information of the control chain, measured at the actuator, can then be used to (i) choose the best control input among the provided ones according to the current conditions, (ii) react in case of packet losses by applying instead a pre-determined (thus deterministic) actuation value, and (iii) provide an updated estimate of delays and enable dynamic reconfiguration of the execution of the active control modes for the subsequent iterations.

II. Related Work

The area of networked control systems has been explored for many years [2, 4]. While computational capabilities can be increased using cloud and edge devices, uncertainties in delays and packet losses represent the hardest obstacle for designing safely performing networked control applications [3, 5]. Precise information of delays at runtime can be gathered by measuring them directly at the actuator. This approach has been however overlooked since it allows only for late knowledge, after the control input has been computed. In this work, we argue that this information can be still useful if additional intelligence at the actuator level is introduced to take informed decisions at the latest step of the control chain, based on the measured delay values or on the detection of packet losses. Different designs of "smart" actuators have been proposed in literature, where intelligence added to actuators

2642-7346/24/$31.00 ©2024 IEEE
DOI 10.1109/RTAS61025.2024.00041

provides additional capabilities to the system. As examples, they have been proposed for self-diagnostic [6], online control re-tuning [7], or as local controllers for safe fallback [8].

In this paper, we combine a smart actuator with a multi-mode controller [9], to make use of the information of latency at the actuator level to choose, between the multiple computed ones, the control input that is optimized for the current chain latency. This approach provides better performance guarantees with respect to classic control designs, and to the best of our knowledge, it was previously unexplored in literature.

III. Model

A. Distributed System Setup

We consider a distributed setup consisting of a set of possibly heterogeneous compute nodes interconnected via a network, which may range from micro-controller-grade nodes to datacenter-style servers. The nodes offer various capabilities, e.g., the ones connected to physical plants provide sensing and actuation used by the applications deployed on them.

Each application executing on this infrastructure consists of a set of communicating tasks that collectively carry out the intended functionality. In general, the system can accommodate multiple applications, each with specific deployment requirements. A suitable deployment of an application involves considering these requirements and determining an assignment of the tasks to nodes and possibly updating the configuration of the resources or the exchange of messages. e.g., to reserve a dedicated budget for them. This paper focuses on control applications, encompassing sensing, control, and actuation tasks deployed in a distributed manner, as discussed next.

B. Distributed Control Application

We consider the simplified setup in Figure 1, consisting of a plant to be controlled, a sensor, a digital controller and a smart actuator, also called simply "actuator" hereafter. The plant is modeled as a continuous-time, linear time-invariant function

$$\dot{x}(t) = Ax(t) + Bu(t) \quad \text{and} \quad y(t) = Cx(t),$$

where $x(t)$ is the internal plant state, $y(t)$ the output of the plant, $u(t)$ the controlled input and A, B, and C the dynamic matrices of appropriate size. The sensor samples $y(t)$ periodically, with period T_s, producing the discrete values $y_k = y(kT_s)$, with $k \in \mathbb{N}$. The controller includes a set of N control functions f_1, $f_2, \ldots$ (called hereafter control "modes"), each optimized against a specific interval of sensor-to-actuator latency, namely $[\tau_i^{\min}, \tau_i^{\max}]$ for an arbitrary mode f_i. After receiving the output y_k from the sensor, the controller triggers the execution of the current set of modes, each producing a corresponding control input ($u_{i,k} = f_i(y_k)$ for any f_i). After receiving the set of values $\bar{u}_k = \{u_{i,k} \mid i = 1, 2, \ldots, N\}$ from the controller, the actuator selects one of such control inputs and holds it until a new update is received. The management of the modes and the selection mechanism are explained in more detail in Section IV.

The controller executes on a remote node (cloud, edge or a dedicated remote processor). The digital components of sensor and actuator run on local nodes, and sensor, controller, and actuator are connected via networked communication infrastructure as in Figure 1. We denote with $\tau_{sc,k}$ the communication delay from sensor to control experienced by the k-th sensor output y_k, and with $\tau_{ca,k}$ the delay from control to actuator of the corresponding control input $\bar{u}_k$.

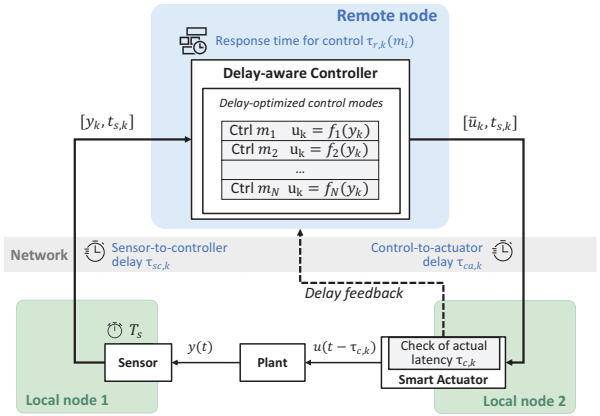

Figure 1: Networked control application with a smart actuator. Exchanged data is marked in black, delay values are in blue.

The response time of an arbitrary mode f_i depends on the complexity of the control function itself and on the amount of computational resources allocated for the control task. The response time of the overall control function(s) executed during the k-th iteration is denoted by $\tau_{r,k}$. We consider the case where communication delays are more prominent than computational delays for the overall chain latency. The overall control chain latency $\tau_{c,k}$ is thus computed as:

$$\tau_{c,k} = \tau_{sc,k} + \tau_{r,k} + \tau_{ca,k}. \tag{1}$$

Sensor, controller, and actuator agree on a common notion of time (synchronized clocks), which can be obtained with state-of-the-art mechanisms. The data is timestamped while passing through the control chain. The (absolute) timestamp $t_{s,k}$ is generated at the sensor level when the sensor data is sampled, and transmitted in the data packets throughout all the control chain, until the actuator. The actuator includes (possibly limited) computational capabilities, as well as some buffer space for input data. For the purposes of this work, we require that the actuator includes a limited computational capability which is sufficient to obtain the actual sensor-to-actuator latency from simple computations over timestamps, and take simple decisions based on that. A relative deadline D_c is defined for the control chain, and a timeout mechanism via a watchdog at the actuator level checks if $\tau_{c,k} \leq D_c$.

IV. Approach

A. Delay-Aware Control with Multiple Modes

In networked control systems, the control chain may be subject to delays that vary widely across different iterations. In this context, we argue that a *delay-aware* multi-mode control, where each mode is designed to optimally withstand a specific

(small) range of delays, provides better control performance properties than using a single controller which must be more conservatively tuned to withstand a wider range of delays. In its simplest form, this design can contain as few as two modes, where one is optimized for "normal" operating conditions, and one for the case of higher interference. As an example, it could include a complex controller such as an MPC that is required to be executed with low latency, paired with one or more simpler controllers optimized for larger delay values.

In general, we could design an arbitrary number N of control modes, spanning different delay ranges. Incorporating knowledge about the network topology (e.g., the number of hops) and interfering traffic can enhance the efficiency of the implemented controller, by restricting the design to handle only the range of delays that actually occur in the network. On the other hand, it may happen that a stabilizing controller is unrealizable for some large delay values. In this case, a maximum delay threshold for feasibility can be used as a requirement for the deployment on the distributed system.

The following example presents a numerical evaluation comparing a multi-mode design of Linear Quadratic Regulators (LQRs)[1] with respect to standard single-mode designs.

Example 1. *We consider an unstable dynamical system*

$$\dot{x}(t) = \begin{bmatrix} 2.0 & 3.0 \\ 0.0 & 0.1 \end{bmatrix} x(t) + \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix} u(t),$$

with sampling $T_s = 0.2\,\mathrm{s}$. The chain delay is assumed discrete and uniformly distributed in the range $\tau_{c,k} \in [0.05, 0.19]\,\mathrm{s}$.

A multi-mode LQR design (MM-LQR) with 15 modes, optimized for uniform intervals of delays in the range of 0.01 s, is compared against two single-control designs, namely a nominal LQR (N-LQR), i.e., a controller that assumes no delay, and a robust LQR (R-LQR), which always considers the worst-case time delay. In the MM-LQR design, at each iteration the actuator uses the control input associated with the closest delay to the current one. Figure 2 illustrates the closed-loop state evolution for each design, obtained through Matlab simulations. The R-LQR approach is overly conservative, resulting in a large state variance, while the N-LQR introduces oscillations due to the neglect of the time delay information. The MM-LQR allows for a delay-aware selection of control inputs, leading to a significant reduction in state mean and variance. In Table I, we quantitatively investigate the control performance of the three LQR approaches by using the closed-loop cost $J = \sum_k (x_k^\top Q x_k + u_k^\top R u_k)$, computed over the 8 seconds simulation window, where Q, R are the LQR weights: MM-LQR shows significantly lower (better) values.

In our approach, we make use of the powerful computational resources of the edge/cloud environment to execute all control modes at each iteration. The controller reads the sensor data y_k and computes for each mode f_i the corresponding control input $u_{i,k}$, obtaining a set of control inputs $\bar{u}_k$. Each value

[1]While we consider only LQR-based controllers here for ease of modelization, a similar scenario could also exist in a real setup, e.g., if it is the last step of a control application including other computationally expensive algorithms.

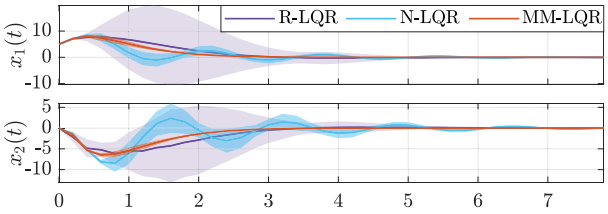

Figure 2: 1000 Monte-Carlo simulations of the closed-loop system with initial state $x(0) = [5, 0]^\top$. The solid lines denote the state mean and the shaded area the 1-σ confidence bands.

Table I: Closed-loop performance of each controller. $\mathbb{E}(J)$is the expected cost (mean); $\mathrm{std}(J)$ is the standard deviation.

Controller	R-LQR	N-LQR	MM-LQR
$\mathbb{E}(J)$	2686.5	3384.2	1218.9
$\mathrm{std}(J)$	4301.6	2107.6	318.1

$u_{i,k} \in \bar{u}_k$ is marked with the associated latency interval against which it is optimized. When all the control modes have completed their execution, the labeled set $\bar{u}_k$ of control inputs is sent to the actuator via the network, including the timestamp $t_{s,k}$ of the corresponding sensor value y_k, and the amount of delay for which each mode is designed.

When the smart actuator receives the set of control inputs, it computes the actual delay using the current reception time $t_{a,k}$ and the (sensor) timestamp from the label, i.e., $\tau_{c,k} = t_{a,k} - t_{s,k}$. Based on this value, the actuator selects the control input $u_{i,k}$ of the control mode that is designed for the time delay *nearest* to $\tau_{c,k}$. If the controllers are optimized for *intervals* of delay, the actuator will choose the control mode whose interval contains $\tau_{c,k}$. In case the control modes are designed for *discrete* values of chain delays, the mode associated with the nearest larger delay with respect to $\tau_{c,k}$ is used. In this case, the actuator may wait until the actual time delay coincides with the time delay associated with the selected control mode, and then applies it to the plant.

This approach guarantees that the controlled system behaves *as if it had full knowledge of the current delay*. The dynamics of the resulting closed-loop system resembles a switching linear system, and its stability may be established by design using, e.g., a switched Lyapunov function approach similar to [9], resulting in uniform asymptotic or mean-square stability (depending on the characteristic of the time delays).

B. Dealing with Packet Losses

The modes of the delay-aware controller can together cover a large range of delay values, but some upper limit (D_c) to the delay is required by design. Anyway, it may still happen that for some iterations the delay exceeds the maximum value D_c, or that the message sent through the network is corrupted or lost before reaching the actuator. From the perspective of the smart actuator, "too late" data and unreadable/lost data can be treated in the same way as *packet loss* events.

The timeout mechanism at the actuator level can be used to check if the k-th packet of the control chain arrives within the deadline D_c. Every packet that arrives after the deadline,

i.e., when $t_{a,k} - t_{s,k} > D_c$, is discarded. To compensate for dropped packets, a deterministic fallback mechanism is triggered. One simple choice can be to either hold the actuator value determined by the previous control input for another iteration, or to choose to actuate a default value (e.g., zero) [10]. This fallback condition must be taken into account when checking for stability of the switching closed-loop behavior, as an additional dynamics. Depending on the specific case, either holding or zeroing could prove to be most effective to withstand multiple consecutive packet losses [5].

More complex approaches are possible. For example, MPCs compute not only the current control input, but also the projected inputs of the next steps in the control horizon. If the whole current and projected values are sent to the actuator at each step in the message packet, in case of a lost update at step k, the actuator could use the estimated k-th control input that was computed at the $(k-1)$-th step, as best approximation of the correct k-th value. An analysis of this design is under investigation, and will appear in future works.

C. Feedback from Smart Actuator

A possible drawback of executing all control modes together at each iteration is that this may unnecessarily increase $\tau_{r,k}$ in the overall chain delay, while also limiting the flexibility of the resource management on the part of the distributed nodes. In the worst case, it might even be unfeasible to run all the modes together, if resources are too scarce.

To compensate these effects, we describe here a more advanced scenario where the information from the actuator regarding the measured delay (or notifying lost packages after a timeout) can be sent to an entity[2] capable of reconfiguring the control task. This information will be available at least one step later, i.e., the delay measured for the k-th iteration will affect at the earliest the $(k+1)$-th iteration. A related mechanism in literature is the so-called *feedback scheduling*, used instead to adapt the scheduling of the overall task set [11].

In our setup, we make use of the feedback delay information to steer the choice of which control modes to execute, with the goal of optimizing the computational resources. By implementing a *delay estimation* mechanism at the edge/cloud level, expanded with real delay measures from the actuator, it could be possible to execute at step $(k+1)$ only one control mode (or a subset of them) that is optimized for intervals of delays similar to the delay measured at step k. The estimator will compute at each step k an estimate of the chain latency of Eq. (1), before starting executing the control function. The overall *estimated* chain latency $\hat{\tau}_{c,k}$ is computed as follows:

$$\hat{\tau}_{c,k} = \tau_{sc,k} + \hat{\tau}_{r,k} + \hat{\tau}_{ca,k} + \varepsilon_{k-1}, \tag{2}$$

where $\hat{\tau}_{r,k}$ is the estimated response time of the chosen control mode, $\hat{\tau}_{ca,k}$ is an estimation of the control-to-actuator delay based on the network status, and ε_{k-1} is a correcting value based on the difference between the estimated delay at the previous step $\hat{\tau}_{c,k-1}$ and the measured delay $\tau_{c,k-1}$ at step $(k-1)$, sent by the actuator. Here, $\tau_{sc,k}$ is the only value that can be measured exactly (comparing the receiving time and $t_{s,k}$). The estimation process of $\hat{\tau}_{r,k}$ is iteratively performed for each control mode f_i, checking if the estimated overall delay $\hat{\tau}_{c,k}$ obtained considering the execution of f_i fits its corresponding delay interval. Then, only the best fitting control mode will be executed. The response time will also need to be properly inflated to account for the estimation overhead itself.

In case of delays evolving slower than the time characteristic T_s of the control chain, and thanks to the feedback influence of ε_k, the estimation can converge to a value close to the real chain delay, thus providing another important step to the reliability of the entire system. Tests on this setup are undergoing and will be part of future works.

[2]Depending on the implementation, the information could be sent to a dedicated component in charge of system configuration, e.g., an orchestrator, or to a sophisticated version of the control task capable of configuring itself.

V. Conclusion

In this paper we presented a novel control design architecture for networked control systems distributed over edge/cloud platforms. The proposed controller on a remote node consists of multiple control modes executed together, each mode optimized for different latency conditions. A smart actuator on a local node measures the control chain delay and chooses the most suitable control input at runtime, thus obtaining an optimized delay-aware control behavior. The smart actuator can also be closed in feedback with the distributed platform, using the measured delay information to adjust the controller configuration for the subsequent iterations.

Acknowledgments

This work was supported by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under the grant no. 13IPC021.

References

[1] P. Mundhenk, A. Hamann, A. Heyl, and D. Ziegenbein, "Reliable distributed systems," in *Proc. 2022 Design, Automation & Test in Europe Conf. & Exhibition (DATE)*. IEEE, 2022, pp. 287–291.

[2] A. Bemporad, M. Heemels, M. Johansson *et al.*, *Networked control systems*. Springer, 2010, vol. 406.

[3] M. B. Cloosterman, N. Van de Wouw, W. Heemels, and H. Nijmeijer, "Stability of networked control systems with uncertain time-varying delays," *IEEE Trans. Aut. Contr.*, vol. 54, no. 7, pp. 1575–1580, 2009.

[4] P. Park, S. C. Ergen, C. Fischione *et al.*, "Wireless network design for control systems: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 978–1013, 2017.

[5] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein, "Control-system stability under consecutive deadline misses constraints," in *Proc. 32nd Euromicro Conf. on Real-Time System*, 2020.

[6] J. C. Yang and D. W. Clarke, "The self-validating actuator," *Control Engineering Practice*, vol. 7, no. 2, pp. 249–260, 1999.

[7] D. Lee, J. Allan, H. A. Thompson, and S. Bennett, "Pid control for a distributed system with a smart actuator," *Control Engineering Practice*, vol. 9, no. 11, pp. 1235–1244, 2001.

[8] Y. Ma, Y. Wang, S. Di Cairano *et al.*, "Smart actuation for end-edge industrial control systems," *IEEE Trans. Autom. Sci. Eng.*, 2022.

[9] J. Daafouz, P. Riedinger, and C. Iung, "Stability Analysis and Control Synthesis for Switched Systems: A Switched Lyapunov Function Approach," *IEEE Trans. Automat. Contr.*, vol. 47, no. 11, 2002.

[10] L. Schenato, "To zero or to hold control inputs with lossy links?" *IEEE Trans. Automat. Contr.*, vol. 54, no. 5, pp. 1093–1099, 2009.

[11] M. S. Branicky, S. M. Phillips, and W. Zhang, "Scheduling and feedback co-design for networked control systems," in *Proc. 41st IEEE Conf. Decis. Control*, vol. 2. IEEE, 2002, pp. 1211–1217.