Punctual Cloud: Achieving Punctuality for Time-Critical Cloud Control Systems

Haorui Peng, Fatemeh Akbarian, William Tärneberg, Maria Kihl,
Department of Electrical and Information Technology, Lund University, Sweden

***Abstract*—Cloud Control Systems (CCSs) harness the power of cloud resources to carry out intense computational tasks, however, they face challenges in delivering time-critical control signals due to network and cloud-induced delays. In this paper, we introduce a novel framework, "Punctual Cloud", designed to enhance the timely delivery of control signals in CCS. This framework ensures that control signals are calculated in the cloud and delivered promptly, minimizing system-induced delays. The punctual cloud framework allows a basic controller to manage dynamic, time-sensitive processes over the cloud and is implemented in a microservice architecture to simplify CCS deployment and maintenance. Our evaluation indicates that our framework, while requiring less computational effort, is capable of tolerating longer delays compared to an optimal controller like Model Predictive Control (MPC), which is typically used in time-delay control systems.**

***Index Terms*—Cloud Control systems, Time-critical control, Delay compensation, Microservice architecture**

I. Introduction

Industry 4.0 has catalyzed the digital transformation of manufacturing systems, with cloud computing emerging as a crucial component, thanks to its seemingly unlimited computing resources and storage capacity. The cloud plays various roles within industrial systems, including data storage, centralized scheduling, and control task computing. Among all the cloud industrial applications, Cloud Control System (CCS) for real-time applications present the most stringent demands on cloud and network infrastructure.

CCSs are a type of Networked Control Systems (NCSs) where the controllers are deployed as services in either centralized or edge clouds. These systems typically close their control loops through IP networks and run their controllers in a virtualized environment with layered software management and access control in the cloud.

Furthermore, cloud computing has ushered the microservice architecture, a new software design paradigm that breaks a monolithic application into multiple microservices in the cloud. This approach enhances both the maintainability and deployability of an application, making it the most promising software paradigm for cloud web services. However, it also introduces inter-service communications, potentially extending service response times.

These factors may result in a longer and more variable wait time for plants to receive control signals from CCS compared to other control systems. This waiting time is termed as *response delay*. A lengthy response delay makes operating a CCS challenging, particularly for real-time plants with fast dynamics. These systems may fail if the correct control signal is not delivered promptly for actuation.

Research such as those presented in [1], [2] has demonstrated the impact of latency due to virtualization technology in cloud computing. Similarly, various studies on low-latency or real-time cloud applications have acknowledged the challenges posed by delays within CCS [3], [4].

Control systems characterized by response delays fall into the category of time-delayed control systems, a well-explored research area. Traditional approaches such as the Smith Predictor and its variants have been employed for decades [5], [6]. MPC, a type of controller that uses a model to calculate a sequence of future control signals [7], [8], has also been used to compensate for delay in time-delay systems. This method is especially useful in CCS, given the cloud's ability to handle the significant computational demands of MPC [9], [10].

CCS research has also been conducted for various applications and goals. For instance, [11] introduced a concept similar to CCS called Control as a Service (CaaS), which aimed at real-time adaptive cruise control over a wireless network using MPC. [12] presented a method to avoid collisions in a multi-robot scenario using an edge-based centralized MPC method and deployment. [13] proposed a security framework for CCS capable of detecting and mitigating cyber-attacks, thereby enhancing system resilience. A multi-tier industry control system combining local and edge controllers was explored in [14], while [15] resolved a mission-critical control task using an MPC-based cloud service. Lastly, [16] investigated a resource allocation problem within CCS.

The field of CCSs offers substantial potential to maximize the benefits of cloud computing for industrial control systems. Despite time-delay control systems being extensively researched over decades, the latency challenges associated with deploying CCS remain largely unexplored. Most literature on Networked Control Systems (NCSs) does not address response delays significantly longer than the control application's sample interval or the variance in delay. Moreover, while MPC has been widely considered as a solution for CCS, its computational time and resource demands cannot be overlooked. The capability of MPC to handle lengthy response delays is limited, as we demonstrate later in our evaluation. Additionally, prior CCS research has seldom capitalized on the microservice architecture when deploying on cloud or edge infrastructures. In this paper, we introduce a new framework, "Punctual Cloud", designed to deploy CCS that can mitigate delays between the cloud controller and the plant, caused by

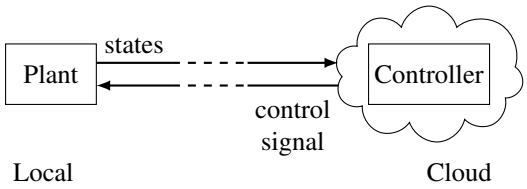


Fig. 1: Cloud control system, where the states and control signals are sent via the network between the plant and cloud controller.

network and cloud environments. This framework is relevant when the control signal's response delay exceeds one sampling interval of the control application, resulting in system performance degradation. This is a common occurrence in time-critical control systems characterized by fast plant dynamics and short sampling intervals.

Our contribution is threefold. Firstly, our framework is designed to ensure the 'punctuality' of control signals, meaning that a control signal generated by the cloud controller in CCS can be timely delivered and actuated at the intended plant state, even when system response delay exceeds one plant sampling interval. Secondly, our proposed framework offers a way to employ simple controllers to address the delays in CCS that affect system performance. Our approach exhibits superior control performance under long delay scenarios for mission-critical applications, compared to optimal controllers like MPC, which is commonly used in similar contexts. Lastly, we present a testbed of the framework deployed in a microservice architecture, which enables easy replacement or upgrades for control algorithms.

II. Targeted System and Problem Description

In this paper, we focus on the CCS depicted in Fig. 1. Here, the controller is positioned as a cloud service, either at the network's edge or within a centralized cloud. The control plant dispatches its states as feedback to the remote cloud service, which in turn returns the control signal back to the plant.

A. System Model

We consider a dynamic and observable control system with a nonlinear discrete physical model Eq. (1), with plant state $x(k)$ at time k, and control signal $u(k)$ that is actuated on the state at this moment.

$$x(k+1) = f(x(k), u(k)) \tag{1}$$

The linearized discrete state-space equation of the control system is presented as Eq. (2). The sampling interval of the discrete system is T_s.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{2}$$

In a CCS involving a dynamic plant, the controller functions as a cloud service, generating control signals remotely based on the plant states transmitted from the plant over a network. We define the *response delay* of the system as T_r, as

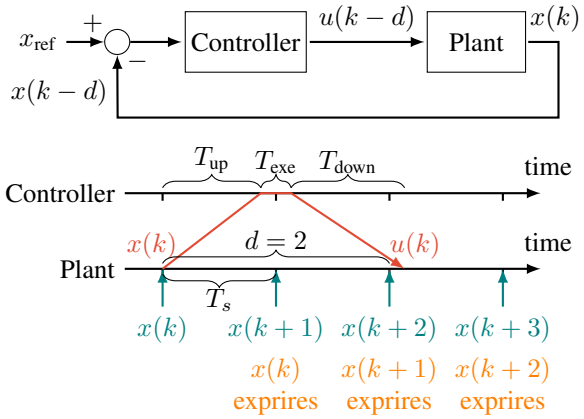


Fig. 2: Top figure: Control diagram of a time-delay system with d steps discrete response delay; Bottom figure: Time series example when $d = 2$ in CCS.

represented in Eq. (3). This delay is a stochastic variable with a mean value of μ and a jitter value of ϕ. It comprises the sum of three time durations: T_{up}, T_{down}, and T_{exe}. Here, T_{up} is the time required to send state information from the plant to the controller, T_{down} is the time to transmit a control signal from the controller to the plant, and T_{exe} is the execution time for computing the control signal in the cloud. We define the *discrete response delay* as d, as calculated in Eq. (3).

$$\begin{aligned} T_r &= T_{\text{up}} + T_{\text{down}} + T_{\text{exe}} \\ d &= \lfloor \frac{T_r}{T_s} \rfloor \end{aligned} \tag{3}$$

When the discrete response delay $d \geqslant 1$, the CCS transforms into a time-delay control system with a delay of d steps in discrete time. The system can then be modeled as shown in Eq. (4). The control diagram of this time-delay system is depicted in the top figure of Fig. 2.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k-d) \\ y(k) &= Cx(k) + Du(k-d) \end{aligned} \tag{4}$$

B. Problem description

As depicted in the lower illustration of Fig. 2, in a CCS, when the response delay $d \geqslant 1$, the control signal $u(k)$, calculated in the cloud based on $x(k)$, is returned to the plant when the state has progressed to $x(k+d)$, and when $x(k)$ is outdated.

A controller designed on a linear and delay-free model (Eq. (2)) might still be capable of actuating and stabilizing the plant. This suggests that $u(k)$ might still be effective when applied to $x(k+d)$, provided the plant dynamics are slow, and there is no substantial difference between $x(k)$ and $x(k+d)$. However, when the network delay significantly exceeds the sampling interval T_s, and the plant exhibits fast dynamics, such a delay-free model controller will no longer perform as expected in a CCS. This is because $u(k)$ is calculated based

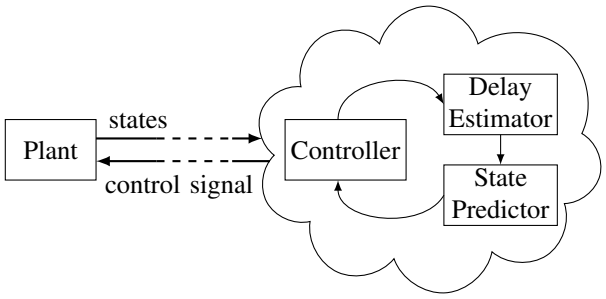


Fig. 3: Punctual cloud architecture

on the plant state $x(k)$, which could drastically differ from $x(k+d)$.

In the subsequent section, we introduce the "Punctual Cloud" framework, which mitigates system delays without modifying the existing delay-free controller. This ensures that $u(k)$ can be applied promptly at state $x(k)$, even if the network and the cloud system contribute to a significant response delay d between the cloud controller and the plant.

III. Punctual Cloud Framework

In this section, we introduce the "Punctual Cloud" framework for CCS, enabling a simple delay-free model-based controller to manage a remote plant despite substantial delays. The concept involves enhancing control signal punctuality by integrating delay estimator and state predictor services alongside the controller service in the cloud. They are collectively deployed in a microservice architecture, either in the cloud or at the edge, as shown in Fig. 3.

The framework, demonstrated as a control diagram in Fig. 4, predicts the plant state $x(k+d)$ when the controller receives state $x(k)$. The controller subsequently generates control signal $u(k+d)$. Unlike MPC which anticipates a sequence of future states, our method predicts the plant state only at the future time $k+d$, when the generated control signal will be actuated on the plant.

The delay estimator, state predictor, and controller operate independently, allowing individual algorithm changes without affecting other components, as long as the required inputs and outputs in Fig. 4 are furnished. In the following segment, we present the methods employed in each component of our CCS for managing a dynamic plant with arbitrary network delays operating at a sampling rate T_s.

A. Delay estimator

The delay estimator generates an estimate of the current discrete response delay, denoted as $\hat{d}$. This estimation is then utilized by the state predictor to forecast the state $\hat{x}(k+\hat{d})$. Accordingly, the controller generates a control signal $u(k+\hat{d})$ based on this predicted state, which is slated for actuation on the plant at time $k+\hat{d}$.

To estimate the continuous response delay $\hat{T}_r$ associated with each control signal, we employ the Exponential Moving Average (EMA) approach, as illustrated in Eq. (5). In this equation, ω signifies the weight of the EMA estimator, and

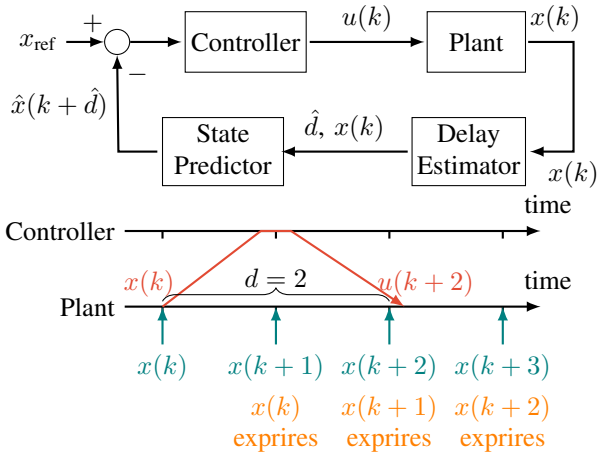


Fig. 4: Top figure: Control diagram of the punctual cloud for a CCS with d steps discrete response delay; Bottom figure: Time series example when $d=2$ in CCS with punctual cloud.

T_r^{input}—transmitted alongside $x(k)$ by the plant—provides information about the response delay related to previously delivered control signals. The discrete time horizon $\hat{d}$, output by the delay estimator, is depicted in Eq. (6).

$$\hat{T}_r = \omega \hat{T}_r + (1-\omega) T_r^{\text{input}} \tag{5}$$

$$\hat{d} = \lfloor \frac{\hat{T}_r}{T_s} \rfloor \tag{6}$$

B. State Predictor

With the estimated delay $\hat{d}$, the state predictor forecasts the state $\hat{x}(k+\hat{d})$ based on the input $x(k)$. Consequently, the controller is able to generate $u(x+\hat{d})$ using this predicted state. We utilize a Smith Predictor, which is based on a nonlinear model of the system. The Smith Predictor concept holds that if the model mirrors the plant, the controller of a time-delay system can be designed around its delay-free model [6]. However, models are not perfect and there is always a disparity between a plant and its model. The Smith Predictor addresses this by factoring in the error in the previous prediction $\hat{x}(k)$ and incorporating it into the next prediction $\hat{x}(k+1)$. As such, a control signal is generated based on $\hat{x}(k+1)+x(k)-\hat{x}(k)$.

Algorithm 1 Smith Predictor in d steps

Input: $x(k)$, d, $\{u(k), u(k+1), ..., u(k+\hat{d}-1)\}$
Output: $\hat{x}(k+\hat{d})$
  $i \leftarrow 1$
  $e(k) \leftarrow x(k) - \hat{x}(k)$
  while $i \leq d$ **do**
    $\hat{x}(k+i) \leftarrow f(x(k+i-1), u(k+i-1)) + e(k)$
    $x(k+i) \leftarrow \hat{x}(k+i)$
    $i \leftarrow i+1$
  end while

In our state predictor, we leverage the nonlinear model of the plant to reduce model error compared to a linearized model. We use Eq. (7) to predict one step ahead for discrete time. To predict the state $\hat{d}$ steps ahead, we run Eq. (7) for $\hat{d}$ times, as illustrated in Algorithm 1. It's noteworthy that the algorithm utilizes $u(k), u(k+1), ...u(k+\hat{d}-1)$, which are control signals generated from previous state inputs.

$$\begin{aligned} e(k) &= x(k) - \hat{x}(k) \\ \hat{x}(k+1) &= f(x(k), u(k)) + e(k) \end{aligned} \tag{7}$$

C. Controller

Our suggested system employs a controller meant for a delay-free plant model, for instance, a Linear Quadratic Regulator (LQR) [17]. Like the MPC, the LQR also uses an optimization object. However, unlike the MPC which requires online optimization during control process, the LQR's optimization is performed offline using the delay-free model (Eq. (2)). The control signal $u(k+\hat{d})$, set to act at time $k+\hat{d}$, is computed in the cloud control service as Eq. (8). This calculation uses the input reference state x_{ref} and a predicted state $\hat{x}(k+\hat{d})$ for the plant at time $k+\hat{d}$. Here, K is the LQR gain, calculated in advance based on the optimization result.

$$u(k+\hat{d}) = K(x_{\text{ref}} - \hat{x}(k+\hat{d})) \tag{8}$$

It's important to mention that while we've selected the LQR as our controller, due to its low computational complexity when deployed as a cloud service, our system is flexible and can easily accommodate other types of controllers.

IV. Testbed deployment

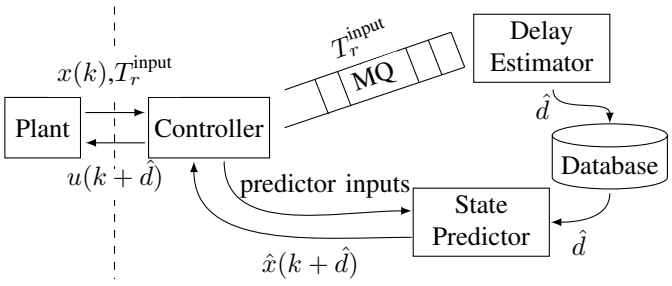
In this section, we provide a comprehensive description of our testbed setup for the punctual cloud framework. As demonstrated in Fig. 5, the testbed includes the plant under control, the network interconnecting the plant and cloud services, and the cloud services that is implemented using a microservice architecture. The entire source code for the testbed will be public after the publication of this paper.

A. The Ball and Beam plant

In our testbed, we emulate a Ball-and-Beam (BnB) plant as the subject of control in the CCS. This emulation, hosted on an Ubuntu server, is based on the source code provided by [18]. The controller's objective is to maintain the ball at a reference position on the beam through rotation, a time-critical task due to the plant's fast dynamics [15]. However, this plant can be substituted with others depending on the control goals.

The BnB plant periodically sends a state vector $x(k)$ at intervals of T_s to the cloud services. This vector includes ball position, beam angle, and ball speed. Along with the state vector, the plant transmits the response delay T_r^{input} of the most recently received control signal for delay estimation.

$$x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} = \begin{bmatrix} \text{ball position} \\ \text{beam angle} \\ \text{ball speed} \end{bmatrix} \tag{9}$$



predictor inputs: $x(k)$, $\{u(k), u(k+1), ..., u(k+\hat{d}-1)\}$

Fig. 5: Punctual cloud deployment. The left part of the dashed line is the plant to be controlled, and the right part is the cloud services in our framework.

B. Network between the plant and the cloud services

We employ HTTP with persistent connection for communication between the plant and our service cluster, given HTTP's widespread usage and support in cloud services [19]. The state vector $x(k)$ and response delay T_r^{input} are transmitted as an HTTP request to the cloud services, with the control signal sent back as an HTTP response.

To emulate varying network situations in the testbed, we use Netem [20] to introduce different delays on the network interface of the emulated BnB plant. The added network delay, T_{add}, subject to a mean value μ and jitter ϕ, represents the sum of T_{up} and T_{down} as per Fig. 2. Thus, the actual response delays, T_r, exceed T_{add} due to the inclusion of execution time in the cloud.

C. Punctual cloud services

Our punctual cloud includes three services: the controller, delay estimator, and state predictor, each relying on the output of another. To efficiently leverage microservice architecture's benefits, we deploy these services in a Kubernetes-orchestrated, bare-metal cluster consisting of seven Ubuntu nodes [21]. The architecture mirrors the system proposed in [16], but with more reliable inter-service communications via a Message Queue (MQ).

In this architecture, the controller service, acting as the "frontend," accepts states from the plant and decouples the received request's information. The controller publishes the response delay, T_r^{input}, to a MQ, to which the delay estimator service subscribes. The delay estimator, gauging recent network conditions, estimates the response delay and updates the *database* for the state predictor to read.

The controller also forwards $x(k)$ and the previously generated control signal, $u(k)$, to the state predictor via another HTTP request. Upon request, the predictor reads the potential response time, $\hat{d}$, from the database, and predicts $\hat{x}(k+\hat{d})$.

The controller subsequently generates the control signal $u(k+\hat{d})$, based on the predicted state $\hat{x}(k+\hat{d})$. The plant buffers the control signal if it arrives prior to time $k+\hat{d}$

and actuates it at the appropriate time, ensuring that the state $x(k+d)$ does not expire beforehand.

V. Evaluation

In this section, we detail the testbed's system parameters and the performance metrics for evaluating our punctual cloud framework. We benchmark our system against a cloud-deployed MPC and a LQR sans punctual cloud solution, each as a standalone service within the same Kubernetes cluster. The experimental results affirm the punctual cloud framework's efficacy in enhancing system performance when discrete response delay $d \geqslant 1$, without imposing additional computational overhead due to its microservice architecture.

A. Parameters of the BnB plant

In the experiment, we consider a BnB plant with beam length$= 1.1m$ and sampling time $T_s = 50$ms. The discredited linear and delay-free state space model written as a form of Eq. (2) has parameters as follows when considering the beam length as 1.1m.

$$\begin{aligned} A &= \begin{bmatrix} 1 & 0.05 & -0.008756 \\ 0 & 1 & -0.3502 \\ 0 & 0 & 1 \end{bmatrix} B = \begin{bmatrix} -6.421e^{-5} \\ -0.003853 \\ 0.022 \end{bmatrix} \\ C &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} D = 0 \end{aligned} \tag{10}$$

The nonlinear physical model is as Equation (11) by giving the control signal $u(k)$ and state $x(k)$, and where $g = 9.80665$ is the gravitational constant.

$$\begin{aligned} x_2(k+1) &= 0.44 T_s u(k) + x_2(k) \\ M &= \frac{5}{7} g \sin(x_2(k+1)) \\ N &= \frac{5}{7} 0.44^2 u^2(k) \\ Q &= T_s x_3(k) + x_1(k) \\ x_3(k+1) &= \frac{NQ - M}{1 - BT_s^2} T_s + x_3(k) \\ x_1(k+1) &= T_s x_3(k+1) + x_1(k) \\ x(k+1) &= [x_1(k+1), x_2(k+1), x_3(k+1)]^T \end{aligned} \tag{11}$$

In our evaluation, we generate a step response by varying the set point of the ball's position between 0 and 0.2m for the BnB plant, assessing system performance under diverse network scenarios. As the control signal $u(k)$ actuates on the beam's center to maintain the ball at its set point, if the ball's position $x_1(k)$ exceeds 0.55, the ball falls off the beam, indicating system instability.

B. Network parameters

In our experiments, we examine different network scenarios by establishing network delays with a mean value of μ, along with added jitter ϕ for a more realistic network experience when connecting to a centralized cloud.

We experiment with added network delays T_{add} having mean values μ ranging from 0 to 200ms - four times the BnB plant's sampling time T_s. Jitter values ϕ range from 0 to a maximum of 25ms, equivalent to $0.5T_s$. The controller's response delay T_r will exceed the added network delay T_{add} due to cloud controller execution time.

Our evaluation focuses on two main network scenarios. The first is a *changing delay scenario*, where the mean delay value μ varies during controller operation, testing the adaptability of the delay estimator. The second scenario comprises *fixed delay experiments*, where T_{add} follows a Pareto distribution with a specific (μ, ϕ) value. We'll present the following three categories of experiments for this scenario:

1) T_{add} has $\mu \in \{0, T_s, 2T_s, 3T_s, 4T_s\}$, jitter $\phi = 0$.
2) T_{add} has $\phi \in \{0, 0.1T_s, 0.2T_s, 0.3T_s, 0.4T_s, 0.5T_s\}$, delay mean $\mu = 3T_s$.
3) T_{add} has $\mu \in \{0, T_s, 2T_s, 3T_s, 4T_s\}$, jitter $\phi = 0.5T_s$.

C. Parameters in punctual cloud services

Here we present the parameters employed in evaluating the delay estimator and controller. It's important to mention that the state predictor is calculated based on the plant's nonlinear model and relies only on inputs from the controller and delay estimator, without requiring any other parameters.

In our testbed, we utilize a delay estimator weight of $\omega = 0.98$. The control signal $u(k)$, which affects the beam's angular speed, is calculated with an LQR gain K, targeting a set point x_{ref} for the ball's position on the beam.

$$\begin{aligned} u(k) &= K\left(\begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \end{bmatrix} - \begin{bmatrix} x_{\text{ref}}(k) \\ 0 \\ 0 \end{bmatrix}\right) \\ K &= \left[-31.0027, -13.9077, 20.4937\right] \end{aligned} \tag{12}$$

D. Performance metrics

For every network scenario detailed earlier, we assess the performance of the system with our punctual cloud framework, and compare it to the MPC and LQR using three different measures:

1) Control performance is evaluated through *Integral Absolute Error (IAE)*. A smaller IAE suggests improved control performance due to minimized deviation from the set point. The IAE for an experiment duration T is computed as Equation (13), where $x_1(k)$ is the position state of the ball, and $x_{\text{ref}}(k)$ is the set-point for the position of the ball on the beam.

$$\text{IAE} = \sum_{k=0}^{T} \left| x_1(k) - x_{\text{ref}}(k) \right|, \tag{13}$$

2) The computational overhead of each method is assessed through their respective execution time, which measures the time taken to compute a control signal. A lower execution time signifies reduced computational burden introduced by the services. Since our testbed implements persistent HTTP connections, we can calculate the execution time T_{exe} using the *response delay* T_r which can

TABLE I: Parameters in the experiments

T_{add}	Network delay added by Netem
u	Mean value of T_{add}
ϕ	Jitter value of T_{add}
T_s	Sampling time of the plant
T_{exe}	Execution time of the services in the cloud
$T_r, \hat{T}_r$	Response delay of control signals and its estimation
$d, \hat{d}$	Response delay in discrete time and its estimation

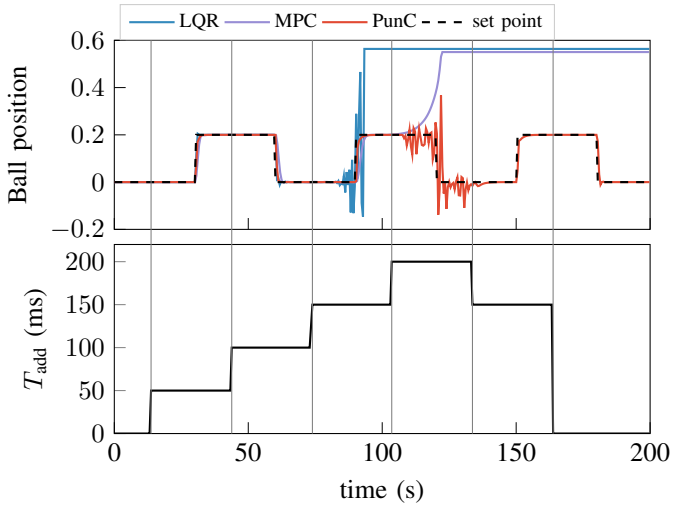


Fig. 6: The top figure is the control step response performances when the added network delay T_{add} changes by following the pattern in the bottom figure.

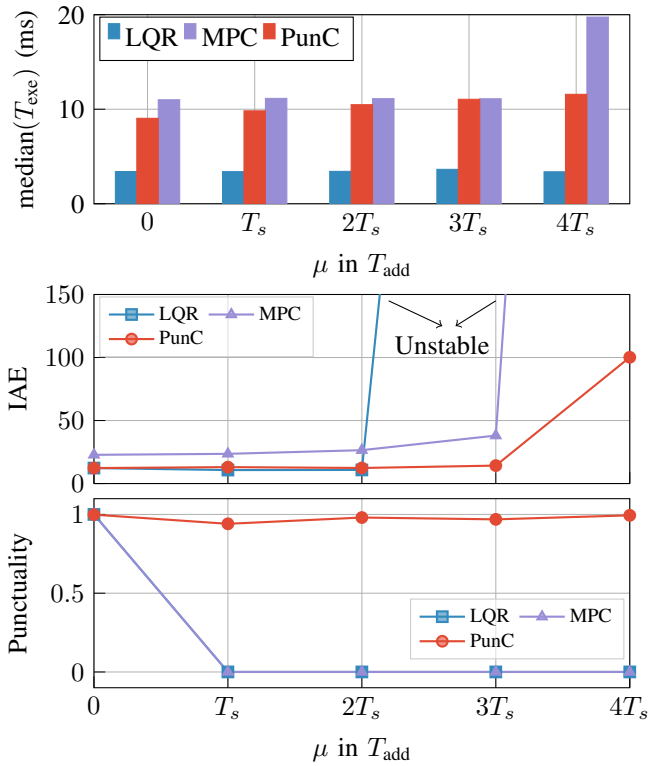


Fig. 7: Performances of three methods when delay increases and no jitter in T_{add}. From top to bottom, each figure presents the execution time, IAE, and punctuality with each method in the evaluation.

be measured by the plant for each pair of HTTP request and response:

$$T_{\text{exe}} = T_r - T_{\text{add}} \tag{14}$$

3) *Punctuality* is assessed as the ratio of control signals arriving "on time" during the experiment. A control signal $u(k)$ is considered "on time" if it reaches and actuates at the plant before the corresponding state $x(k)$ expires. Higher punctuality reflects a greater proportion of control signals being executed as intended and within the desired time frame.

In Table I, we give all the parameters and their notations that are employed in our experiment.

VI. Results

In this section, we analyze the performance of the MPC, stand-alone LQR, and our proposed punctual cloud framework (referred to as 'PunC' in the figures) under various scenarios.

A. Changing delay scenario

In the changing delay evaluation scenario, we conduct experiments where the mean value μ of the added network delay T_{add} varies from 0 to $4T_s$ according to a specific pattern depicted in the lower figure of Fig. 6. As shown in the figure, the control performance of each method is evaluated. It is observed that the stand-alone LQR can handle response delays up to $2T_s$, while the MPC can tolerate delays up to $3T_s$, indicating a response delay $d = 3$. However, when T_{add} exceeds $3T_s$, resulting in a discrete response delay $d \geq 3$, both the MPC and LQR fail to maintain the ball at its set point.

In contrast, our punctual cloud framework generates the control signal $u(k)$ based on the predicted state $\hat{x}(k)$ and ensures its timely arrival to stabilize the actual plant state $\hat{x}(k)$, assuming the network is lossless.

B. Fixed delay scenario

In the fixed delay scenario, we evaluate our system performance by fixing the mean and jitter of T_{add} in the Netem configuration for each experiment. We analyze the system performance in three cases.

1) Analysis on increasing μ and jitter $\phi = 0$: In Fig. 8, we observe the step response under different added delay values with zero jitter. The stand-alone LQR fails to stabilize the plant when $T_{\text{add}} = 3T_s$, causing the ball to fall off the beam. Similarly, when $T_{\text{add}} = 4T_s$, the MPC fails to maintain the ball at its set point. However, our proposed punctual cloud method successfully keeps the ball around its reference position until $4T_s$, albeit with some oscillation. This requires predicting the plant state four steps ahead in discrete time.

The error plot in Fig. 8 demonstrates the performance of the state predictor. It is evident that when T_{add} reaches $4T_s$, the state predictor starts to exhibit significant errors, leading to

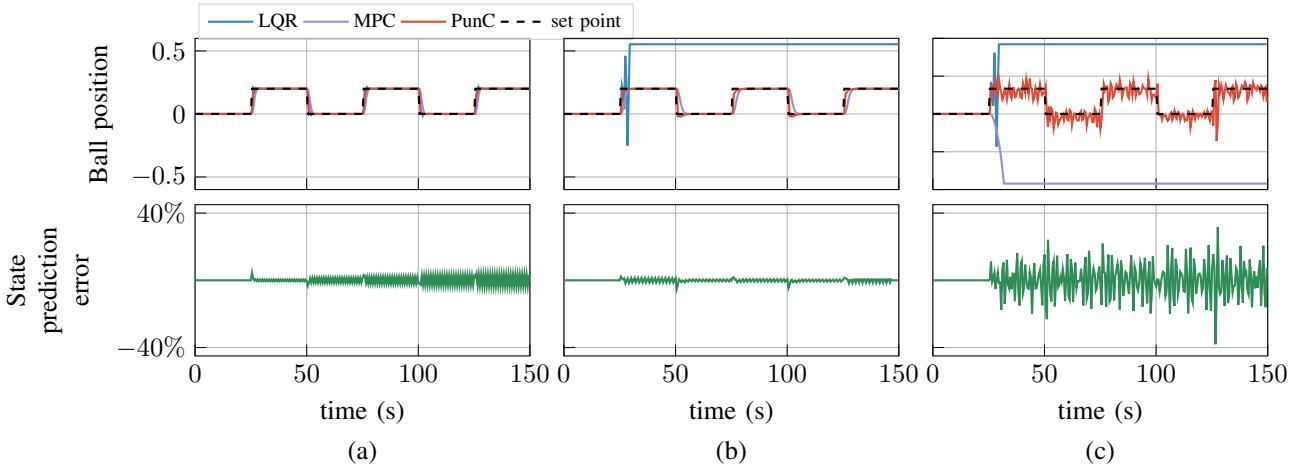


Fig. 8: The first row of the figure is the step response performance when (a) $T_{\text{add}} = 2T_s$ (b) $T_{\text{add}} = 3T_s$ and (c) $T_{\text{add}} = 4T_s$; the second row are the punctual cloud state prediction errors presented in percentage.

control performance degradation and oscillations in the step response. This highlights the impact of the accuracy of the state predictor on control performance, especially for response delays longer than $4T_s$. When $T_{\text{add}} = T_s$, Fig. 8 illustrates that the prediction error is larger than in the case where $T_{\text{add}} = 2T_s$. However, given the relatively shorter delay and its minimal impact on controller performances cross all three solutions, a state predictor is superfluous in this case.

Fig. 7 illustrates the performance of each method in terms of execution time, IAE, and punctuality as the added delay increases from 0 to $4T_s$. The stand-alone LQR has the shortest execution time due to its simple P-controller nature. While the MPC is also deployed as a single-service application without inter-service communication, its response delay is longer than that of our punctual cloud method. This is because MPC requires online optimization for each control signal generation, which is more computationally intensive. Despite the overhead introduced by inter-service communications, the punctual cloud method exhibits comparable or shorter response delays due to its lightweight microservice architecture.

The IAE of all methods increases as the mean of T_{add} increases. The time-delayed control signals of MPC and LQR become less effective as the delay grows, resulting in higher deviations from the set point. When T_{add} exceeds $3T_s$, both MPC and LQR exhibit significantly high IAE values above 1000, indicating system instability. However, our punctual cloud method maintains lower error levels compared to the other two methods.

From the punctuality plot in Fig. 7, it is evident that our framework significantly improves the punctuality of control signals. By incorporating the delay estimator and state predictor services, our method ensures that control signals are applied to the plant state $x(k)$ most of the time, leading to enhanced control performance.

2) Analysis on increasing ϕ when $\mu = 3T_s$: In this case, we focus on the performance of our punctual cloud framework with an added delay mean value of $\mu = 3T_s$ and increasing

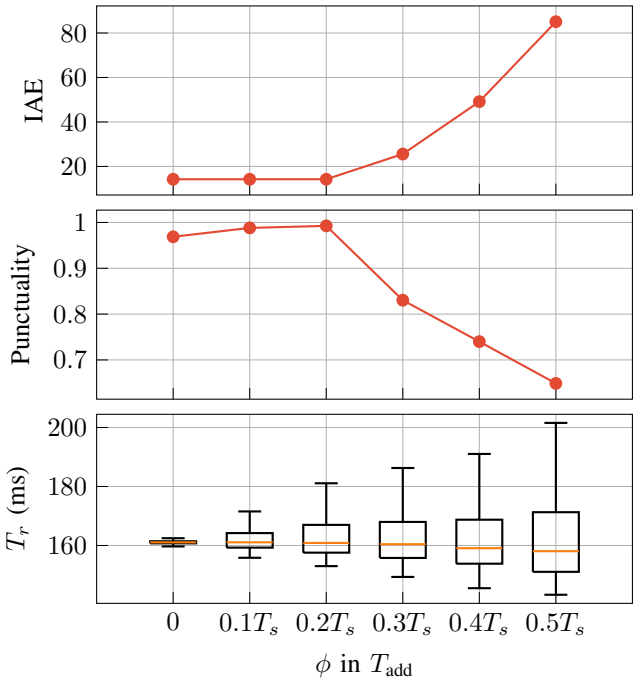


Fig. 9: The first two sub-figures are IAE and punctuality performances of punctual cloud when jitter increases and added network delay is $3T_s$. The bottom figure is the box plots of response delay in each experiment.

jitter from 0 to $0.5T_s$. Since the stand-alone LQR fails to stabilize the system even without jitter in the network, we only present the control performance of our proposed punctual cloud framework.

From the box plots of response time T_r in Fig. 9, we observe that as jitter increases, the median value of response delays remains relatively unchanged, but there is a larger variance and significantly higher $95th$ quantile. This makes achieving punctuality more challenging with the EMA algorithm used

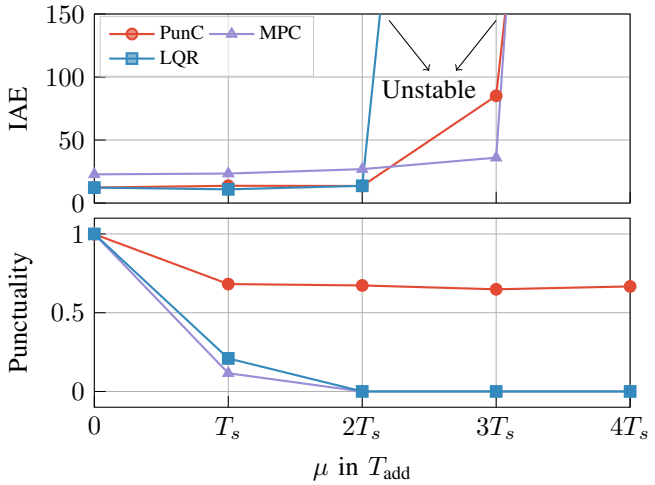


Fig. 10: IAE and punctuality performances of three methods when delay increases and jitter is $0.5T_s$

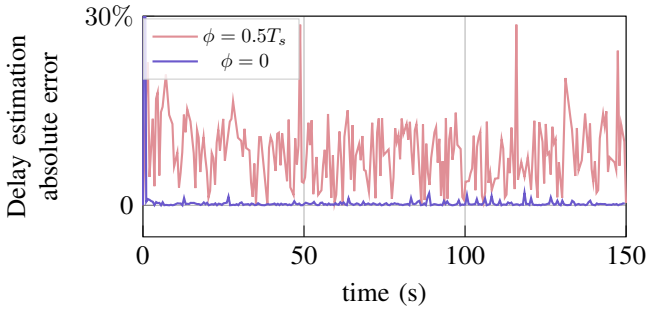


Fig. 11: The Absolute delay estimation error in punctual cloud when jitter $\phi = 0$ and $\phi = 0.5T_s$

in delay estimation.

Fig. 9 also reveals that the IAE increases as jitter increases, but the system can still be stabilized with our punctual cloud frame work under the maximum jitter value evaluated. The larger error in control performance can be attributed to the punctuality plot, where we observe that lower punctuality leads to larger IAE. The decreased punctuality has two main effects on control performance. Firstly, control signals may be applied to plant states that differ significantly from the predicted states on which the signals were based. Secondly, lower punctuality results in lower accuracy in state prediction since the predictor does not have knowledge of which control signals were actually applied between states $x(k)$ and $x(k+d)$.

3) Analysis on increasing μ with $\phi = 0.5T_s$: In Fig. 10, we present the performances of all three methods with increasing mean value μ of the added network delay T_{add} when jitter ϕ is set to $0.5T_s$ in Netem, following a Pareto distribution.

Similar to previous cases, neither MPC nor LQR can tolerate the added network delay when it results in a response delay larger than 3 steps in discrete time. However, in this specific scenario, even our punctual cloud framework fails to maintain system stability when $\mu = 4T_s$. The main reason can be observed in Fig. 11, where we see a significant increase in delay estimation error with EMA when there is larger jitter in the network. This leads to lower punctuality and inaccurate state prediction. Furthermore, in Fig. 8, we observe that when $\mu = 4T_s$, the ball starts oscillating around the reference position due to inaccurate state prediction caused by longer delays. A higher jitter in the network introduces more uncertainties and inaccuracies in the prediction, ultimately rendering the control system unstable.

VII. Conclusion

In this paper, we introduced the "Punctual Cloud" framework for controlling time-critical applications in the cloud, specifically addressing the timely delivery of control signals in cloud control systems with long response delays. Our framework enables the use of a simple controller designed based on a delay-free system model, without requiring any modifications to the controller itself.

The key components of our framework are the delay estimator and state predictor services, which work together to estimate the arrival time of control signals at the plant and compute the control signal based on the predicted plant state at that time. We evaluated our method using a BnB plant emulator with a sampling interval of 50ms and deployed it as microservice architecture in a Kubernetes cluster. The network delay between the plant and the cluster was emulated using Netem.

We compared our framework with two other cloud control services, an MPC and a stand-alone LQR. The results demonstrated that our punctual cloud framework could tolerate network delays up to 4 times the sampling interval ($4T_s$), providing accurate predictions for up to 4 steps in a discrete time. In contrast, MPC and LQR could only tolerate delays up to 2 times the sampling interval ($2T_s$). Our method also exhibited resilience to network jitter up to 25ms($0.5T_s$) under Pareto distribution, as long as the network delay remained below $4T_s$.

Our analysis suggests that improving the accuracy of delay estimation and state prediction algorithms can further enhance delay tolerance. The modular nature of our system allows for easy replacement and integration of alternative algorithms in these services. Our future work includes evaluating our framework on more time-critical industrial plants than BnB, as well as refining the accuracy of our delay estimator and state predictor.

Acknowledgment

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the SEC4FACTORY project, funded by the Swedish Foundation for Strategic Research (SSF), and the IMMINENCE project funded by Sweden's Innovation Agency (VINNOVA). The authors are part of the Excellence Center at Linköping-Lund on Information Technology (ELLIIT), and the Nordic University Hub on Industrial IoT (HI2OT) funded by NordForsk.

REFERENCES

[1] U. Drepper, "The cost of virtualization: Software developers need to be aware of the compromises they face when using virtualization technology." *Queue*, vol. 6, no. 1, p. 28–35, Jan. 2008.

[2] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *2010 Proceedings IEEE INFOCOM*. IEEE, Mar. 2010, pp. 1–9.

[3] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, pp. 726–740, Oct. 2014.

[4] P. Skarin, W. Tärneberg, K.-E. Årzén, and M. Kihl, "Control-over-the-cloud: A performance study for cloud-native, critical control systems," in *International Conference on Utility and Cloud Computing (UCC)*. IEEE, Dec. 2020.

[5] O. Smith, "Closer control of loops with dead time," *Chemistry Engineering Progress*, vol. 53, pp. 217–219, 1957.

[6] Q. C. Zhong, *Robust Control of Time-delay Systems*. Springer, 2006.

[7] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model predictive heuristic control: Applications to industrial processes," *Automatica*, vol. 14, no. 5, pp. 413–428, 1978.

[8] J. E. Normey-Rico and E. F. Camacho, "Model predictive control of dead-time processes," in *Control of Dead-time Processes*. Springer, London, 2007, pp. 271–308.

[9] P. Skarin, K.-E. Årzén, J. Eker, and M. Kihl, "Cloud-assisted model predictive control," in *2019 IEEE International Conference on Edge Computing*. IEEE - Institute of Electrical and Electronics Engineers Inc., Aug. 2019.

[10] K.-E. Årzén, P. Skarin, W. Tärneberg, and M. Kihl, "Control over the edge cloud - an MPC example," in *1st International Workshop on Trustworthy and Real-time Edge Computing for Cyber-Physical Systems*, Nashville, Tennessee, United States, Dec. 2018.

[11] H. Esen, M. Adachi, D. Bernardini, A. Bemporad, D. Rost, and J. Knodel, "Control as a service (CaaS)," in *Proceedings of the Second International Workshop on the Swarm at the Edge of the Cloud*. ACM, Apr. 2015.

[12] A. S. Seisa, B. Lindqvist, S. G. Satpute, and G. Nikolakopoulos, "E-CNMPC: Edge-based centralized nonlinear model predictive control for multi-agent robotic systems," *IEEE Access*, 2022.

[13] F. Akbarian, W. Tärneberg, E. Fitzgerald, and M. Kihl, "Detection and mitigation of deception attacks on cloud-based industrial control systems," in *2022 25th Conference on Innovation in Clouds, Internet and Networks (ICIN)*. IEEE, 2022.

[14] Y. Ma, C. Lu, B. Sinopoli, and S. Zeng, "Exploring edge computing for multitier industrial control," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, pp. 3506–3518, Nov. 2020.

[15] P. Skarin, W. Tärneberg, K.-E. Arzen, and M. Kihl, "Towards mission-critical control at the edge and over 5g." IEEE, Jul. 2018, pp. 50–57.

[16] H. Peng, W. Tärneberg, E. Fitzgerald, and M. Kihl, "Punctual cloud: Unbinding real-time applications from cloud-induced delays," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, 2021.

[17] H. Kwakernaak, R. Sivan, and B. N. D. Tyreus, "Linear optimal control systems," *Journal of Dynamic Systems, Measurement, and Control*, vol. 96, no. 3, pp. 373–374, Sep. 1974.

[18] "cotc-modsim," Last Accessed: September 13, 2023. [Online]. Available: https://www.thefuturenow.se/pypi/cotc-modsim/

[19] J. Dizdarevic, F. Carpio, A. Jukan, and X. Masip-Bruin, "Survey of communication protocols for internet-of-things and related challenges of fog and cloud computing integration," *ACM Computing Surveys*, vol. 51, Apr. 2018.

[20] "Netem," Last Accessed: September 13, 2023. [Online]. Available: https://wiki.linuxfoundation.org/networking/netem

[21] "Kubernetes," Last Accessed: September 13, 2023. [Online]. Available: https://kubernetes.io