# Control-over-the-cloud: A performance study for cloud-native, critical control systems

Skarin, Per; Tärneberg, William; Årzén, Karl-Erik; Kihl, Maria

*Published in:*
International Conference on Utility and Cloud Computing (UCC)

2020

*Document Version:*
Publisher's PDF, also known as Version of record

# Control-over-the-cloud: A performance study for cloud-native, critical control systems

Per Skarin*†‡, William Tärneberg*§, Karl-Erik Årzén‡, Maria Kihl§
†Ericsson Research, Lund, Sweden
‡Dept. of Automatic Control & §Dept. of Electrical and Information Technology, at Lund University, Lund, Sweden
*Equal contributors

***Abstract*—In the Industry 4.0 era, time-sensitive and mission-critical control applications still have a long way to go, from being tied down and co-located with the systems they control, to taking full advantage of the cloud. Conservatively keeping applications local will deprive these complex applications of abundant compute capacity, wider system integration, and the potential for collaborative control efforts. Feedback control systems are unlike other cloud applications - their performance and objectives can be formally defined, they require timely feedback, and they are sensitive to variations in system performance and noise. Although resources are plentiful, the cloud is a noisy and latency prone execution environment, detrimental to feedback control system. In this paper, we evaluate a set of cloud platforms and infrastructures with the intention of hosting feedback control systems. In lower levels of the software stack, we observe differences between clouds. Further up in the stack we see the disadvantages of applying cloud native platforms. With an understanding of expected performance we proceed to evaluate a simple control strategy and show how the sensitive nature of control can cause a seemingly adequate cloud platform to pose a high risk, while a seemingly inadequate platform can positively affect the performance of our proposed controller.**

***Index Terms*—Cloud-native, feedback control systems, cyber physical systems, control over the cloud, Control-as-a-Service**

## I. Introduction

In the industry 4.0 era, Industrial Internet-of-things (IIoT) will enable time-sensitive production processes, that have previously been shackled to production floors on immutable software and hardware, to contribute to and take part of larger site-wide networked systems. Aggregating data and decisions to a central point, such as the cloud, can enable, for example, system-wide performance improvements and early failure detection while also minimizing the infrastructure at the production floor. We heed the call from [1] and argue that without compromising robustness and resiliency [3], industrial feedback control systems can be designed with basic software and hardware on the production floor and augmented by more capable feedback controllers in the cloud.

Some control problems will be implemented in the cloud out of necessity, such as, to enable collaborative control and satisfy data availability, others because of practicalities, and yet others to gain access to resource abundance. One path towards deploying ever more challenging control tasks in the cloud is to mitigate the issue of uncertainty, latency and jitter [8]. A seemingly competing, but rather complementary, challenge to control theory and computer engineering, is to identify a useful trade-off between added latency and the gains from using cloud technology. Through technologies such as Time Sensitive Networks (TSN), industry is working towards the implementation of real-time support in the clouds. Further, enabling mobility requires unprecedented reliability in the networks, which is promised by 5G Ultra-Reliable and Low-Latency Communication (URLLC), network slicing etc. Techniques are being studied and developed [13], [14] to bridge the gap between reliability and efficiency gains. Such *quality elastic* control is aimed at ways to use resources on-demand but can also provide relief to the real-time problem.

In a feedback control system, the effects of a lost or abnormally delayed message depends heavily on the state of the process and the constraints of the feedback loop. Inter-message latency is therefore as great of a challenge as overall reliability. In addition, to realize the cloud-based systems aimed for the Industry 4.0 era, at reasonable costs and development effort, a design criteria is that they rely on tried and trusted Commercial off-the-shelf (COTS) components. Today, cloud-native is the culmination of COTS, cloud, and no-ops software development and deployment. We therefore consider COTS components, services, and platforms as referring to, for example, public and private cloud providers (e.g. Amazon Web Services (AWS), Azure), open-source orchestration and service abstraction platforms (e.g. Kubernetes (K8S), AWS Lambda, Fission) and the connecting general-purpose networking infrastructure, in addition to traditional software.

Designing controllers that are augmented by the cloud requires an awareness of the cloud's performance properties and how this affects the process under control. With this knowledge one can asses the applicability and potential gains of using cloud technology, and the necessary measures to sustain reliability. The purpose of this work is to study the technological step into using cloud-native technology with a view on high-frequency feedback control systems. What constitutes *high-frequency* in this context relates to the identified performance and is application dependent, and as such, has yet to be defined. To that end, we progressively benchmark the layers of Information and Communications Technology (ICT) that constitute a cloud, of two infrastructure providers

This work has been partially funded by the Wallenberg AI, Autonomous Systems and Software Program (WASP), the ELLIIT strategic research area on IT and mobile communications, Sweden's Innovation Agency (VINNOVA) under the 5G-PERFECTA Celtic Next project, the Swedish Foundation for Strategic Research under the SEC4FACTORY project.

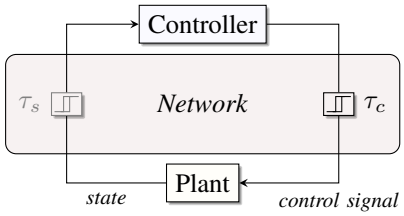



Fig. 1: Networked control.

and then propose and evaluate a feedback control system aligned in frequency and execution requirements with what was observed. Specifically, our contributions towards using cloud-native technology for feedback control systems are:

1) A study of the progression of latency in clouds from theoretical lower bounds to cloud-native application layers.
2) A control-contextual benchmark comparison of lower bound response times from a contemporary private cloud to a public cloud.
3) A proposed cloud-augmented controller, demonstrated through a basic example.
4) An identification of the detrimental effects of offloading feedback control to the cloud. Effects that do not apply to ordinary bit-pipe applications.

Additionally, we present a specific control problem to illuminate performance gains but also detrimental and non-intuitive effects. This comes in two forms, a direct approach which is useful to examine the cloud deployments but is both unreliable and creates an excessive load. In contrast, we show results from an alternative method, with less impressive peak performance gains, but with reliable execution and reduced load on the cloud. Interestingly, response rates are also improved in this mode, making almost all requests useful and allowing the same gains from two seemingly very different deployments. We observe that even when the cloud is unresponsive 87% of the time, we have a measurable benefit of 12% efficiency gains, and can reach an ideal gain of 40% on most platforms. Critical systems may face a reliability issue, and we exemplify altering the dynamics of the controller, reducing the request load while maintaining 10% efficiency gain.

## II. Related work

The interest in using ICT as a means to implement networked control using COTS predates the cloud era. An example is found in [17], where control is implemented over two private Local Area Networks (LANs), to balance a steel ball in a magnetic levitation system. The engineering merit of COTS, which at the time consisted of the Internet, general purpose networks, a web server, and the Common Gateway Interface (CGI), is explicitly acknowledged in this work. More recently, modern systems such as drones have been considered [10], and the potential for implementation using high levels of abstraction. However, such works often focus on single providers and the tools, rather than the control challenge.

Research on deploying traditional feedback control loops to a cloud in general, is often focused on making cloud computing platforms and intermediate networks behave as real-time systems. Targeted efforts have, for example, been made in the areas of deterministic dynamic networks [5], resource allocation in data centers [18], and feedback control implemented in the network [12]. Nevertheless, successful attempts at deploying feedback control loops that span a cloud Virtual Machine (VM) and back, were first made relatively recently [4]. The work in [4] identified that time-varying network delays is a key challenge and it was shown that the availability gap introduced by the cloud could be spanned reliably across multiple clouds, given comprehensive middleware. With more consideration to control theory, the work in [7] showed that a cloud-deployed feedback controller can be a successful agent in a fail-over system, that is both resilient to feedback control and network failures. On a tangent line of research, the authors of [16] demonstrate that the delay incurred by the cloud and the intermediate network can be accommodated for when designing the feedback control loop. Also, not to be underestimated, security is a principal concern when operating in the cloud. On this topic, the authors of [2] showed that the performance overhead of security was manageable in a cloud deployed feedback controller. The above works give credence to the research direction. However, as stated in the beginning of this section, the current state-of-the-art typically propose to change the notion of the cloud in order to fit the needs of feedback control systems, and are not pursuing an adaptation of feedback control systems to the reality of the cloud. The research direction this paper contributes to complements research with the view that there is important progress to be made by considering unadulterated clouds, and as such assumes no real-time support in neither networks nor clouds systems.

## III. Challenges and targeted system

This section provides an anatomy of feedback control systems and presents the targeted system architecture.

### A. Anatomy of feedback control

In a feedback control system, the subject under control is referred to as the *plant*. A plant can be an isolated system such as the joint of a robotic arm, or a large and complex combination of hundreds of valves and switches in a production plant. A *controller* manipulates the plant in order to perform a task such as moving a robotic arm from position A to B or adjusting the rate at which two liquids are combined. A controller is subject to performance *objectives* such as counteracting external forces and friction, or keeping an optimal production rate based on demand and logistics. To capture such behaviors and dynamics, a controller often relies on a *model* that represents the physical system and possibly its environment. Model simplifications, parameter estimations, measurement errors, and unknown disturbances from the environment, incur additional complexity in controlling the plant. The basic control problem consists in deriving a controller which can manipulate the plant to meet the objectives and performance goals. In extension, it also involves finding a representative model, tuning model parameters, formalizing

objectives and performance goals, and handling disturbances in the environment.

### B. Computer controlled systems

From a computer engineering perspective, feedback control systems are discrete-time synchronous applications that must run continuously and have strict timing requirements. If the system's performance begins to deteriorate, it may be challenging, or impossible, to recover, without drastic measures and grave consequences. Consequently, feedback controllers are conventionally implemented on industrial grade hardware - proprietary plant-side Programmable Logic Controllers (PLCs) that have close to deterministic execution times and a long mean time between failures large mean time to failure. A great deal of care is taken to ensure that controllers execute timely, periodically, and that signaling, i.e. sensory input and control output, is deterministic.

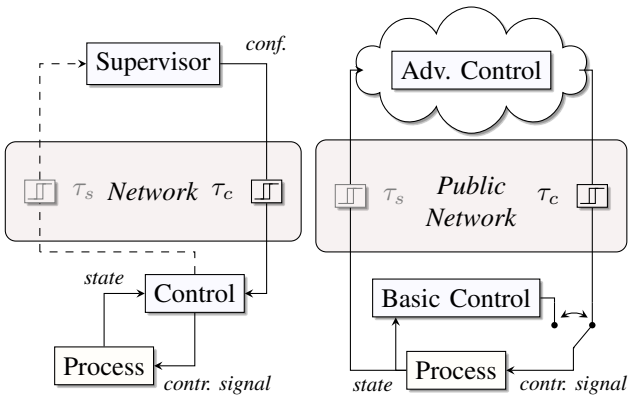
### C. Networked control

The notion of *networked control* implies that the plant and the controller are separated by a network. In addition to the non-trivial interplay between the controller and the plant, networked control is further complicated by network delay and packet loss. Networked critical control is depicted in Figure 1. Here, networked control is critical because the control must function over an unreliable network or the plant objective will fail. For such a scenario, network control theory studies the requirements that must apply to the communication channel in order for the overall system to meet the objectives and not malfunction. For the Maglev system in [17] it is found that there must never be more than three lost packets in succession if the system is to remain operational.

Another type of networked control is shown in Figure 2a. Here, the critical control is not performed over the network. An independent local control loop is acting on the plant to fulfill the current objectives, as defined by a *supervisor controller*. The supervisor controller, which can act at a frequency much lower than what is required to keep the plant operational, sets the parameters of the local controller, altering its task and/or changing the performance objectives. With supervisory control, network latency and packet loss may affect the overall performance but should not inadvertently cause the plant to malfunction. An example of supervisory control is also present in [17], in terms of an implementation that supports manual configuration of the Maglev controller through a web page.

### D. Control-over-the-Cloud

The examples in Section III-C serve to distinguish the type of control we are concerned with. The target in the paper is the critical control in Figure 1. However, the proprietary real-time system at the controller side of the network is replaced by a cloud infrastructure, as shown in Figure 2b. Further, the network is a public network. Here we use the term public network to refer to a network that is shared, and which may or may not be accessible by unknown external parties, but importantly, cannot provide formal guarantees. The loss of real-time execution and the possibly unlimited network degradation are both fatal to the reliability of the control system. To ensure that timely control input is continuously available at the plant, a basic local feedback control must be present. We refer to this basic local control as the *ancillary controller*. The ancillary controller is switched in to recover when networked control fails. A more advanced, high performance, controller is executed in the cloud. While the ancillary control is be able to fulfill the control objective, this, networked, controller enables improved performance. As with supervisory control, a functional network allows the system to reach its full potential. In contrast to supervisory control, the networked controller is 1) manipulating the plant directly and 2) not necessary for the ancillary controller to receive and execute new objectives. Latency imposed by shared networks and cloud infrastructures has impeding effects in terms of how often the ancillary control must compensate for the networked controller's inability to act.




(a) Supervisory control (b) Control over the cloud

Fig. 2: Supervisory v.s. critical control over the cloud

## IV. Experiment setup

In this section, we establish a set of experiments to determine how appropriate the cloud is in realizing the system presented in Section III-D and Figure 2b, based on the COTS design criteria established in Section I. Additionally, we present a simple reference feedback control system on which some of the experiments are based.

### A. Objective

The primary goal in this paper is to study the implementation of an example feedback controller using the architecture in Figure 2b, on a cloud-native platform executing in an ordinary, shared cloud. It is assumed that cloud latency will have largely detrimental effects on the performance of the controller. It is known that the chosen plant can be controlled with a sample rate as low as $15\,\mathrm{Hz}$, but to improve performance, a higher frequency is preferable. It is therefore of interest to investigate the achievable response rate from the cloud. It is also of interest to understand where latency is introduced and, for the most effective use of COTS, how much performance that is lost by choosing technologies at a high level of abstraction. We believe that assessments of cloud latency often are overly

pessimistic. This can be because judgments are based on complex applications, obsolete technology, or incorporates worst case scenario assumptions. We therefore choose to reduce complexity as much as possible and aim to find the achievable lower bounds on response times. To make it easier to refer and reason about the objectives we declare them in the form of three assumptions.

*Assumption 1:* Cloud-native platforms are composed of many systems-of-systems, which results in, for each successive system or OSI layer, a delay with greater variance. This effect will from now on be referred to as *platform noise*.

*Assumption 2:* Control applications of the type in Figure 2b can benefit from the cloud using cloud-native services.

*Assumption 3:* The targeted cloud control system incorporates multiple dynamic systems, and cannot be trivially predicted. In extension, combining results from studies of individual components is at high risk of leading to incorrect assessments of the suitability of the cloud.

### B. Experiments

A set of experiments were designed to test Assumptions 1-3. They are as follows:

**Baseline Round-Trip Time (RTT)** Investigates the minimal time it takes to reach the cloud, and go back again, at each successive layer in an infrastucture's stack. These experiments are fundamental to testing Assumptions 1 and 2. In the RTT experiments, we quantify the latency cost of accessing the cloud and find the latency profiles for a set of communication technologies and cloud infrastructures. In extension, we will be able to determine the upper bound frequency for control.

**Computational complexity vs. response time** Cloud resources are subject to varying control-problem complexity, per request, and the RTT as well as the execution time is measured. These experiments will explicitly test Assumption 2 and determine if Assumption 1 is load dependent. Depending on the state of the plant, the controller requires different amounts of computations in the cloud. To solve the control problem, the controller predicts the plant state over time. The length of this prediction affects the computational load. To create a varied load, we alter the state of the plant and the configuration of the advanced controller's prediction length. To recreate identical conditions for each deployment, three static control-problem workload intensities where determined through experimentation; light, medium, and high. To create the different intensities, the state is changed to produce different degrees of computationally complex problems.

**Feedback control** Using the findings from the above experiments, we proceed to test Assumption 3 and evaluate the experimental feedback control system presented in Section V. It is constructed using a basic, ancillary, controller and an advanced, networked, controller as in Figure 2b. We refer to the combination as simply *the controller*. A *set-point* tells the controller what state the plant should be in. The controller's ability to quickly drive the system to a new state determines its efficiency. As a complication, there are *constraints* to the plant state and the input signal. The networked controller knows about these constraints and takes them into consideration. The ancillary controller does not handle constraints and therefore the efficiency of the local control must be restricted, to reduce the risk and severity of constraint violations. Our performance evaluation criteria is the ability of controller to stay within the constraints and to quickly reach the set-point.

Note that these experiments run in batches, with each batch run as quickly a possible on a single thread. Although the cloud can be used for massively parallel computations, it is important to run single, serialized requests to find the shortest response times. There are 10 requests in each batch. One request is sent initially and disregarded to remove cold starts from the data.

### C. Performance metrics

The cloud performance is asserted in terms of a requests RTT, and execution time, measured in milliseconds. Medians and percentiles are used to show and compare latency distributions. For the feedback control experiment, we define three metrics that allow us to assess the control performance over time. They are all relative to the basic controller. Low values are desirable for all metrics.

**Relative Accumulated Violations (RAV)** A measure of the total violation of constraints. $p_k$ is the state at sample $k$. $c_0, c_1$ are the upper and lower constraints, respectively.

$$\frac{1}{T}\sum_{k=0}^{T}\sum_{j=0}^{1} max(0, |p_k| - |c_j|) \tag{1}$$

**Relative Accumulated Error (RAE)** A measure of the controllers ability to reach the requested set-point. $p_k$ is the state at sample $k$ and $r_k$ is the set-point at sample $k$.

$$\frac{1}{T}\sum_{k=0}^{T} |p_k - r_k| \tag{2}$$

**Relative Maximum Constraint Violation (RMCV)** Measures the maximum distance from the constraint of the system state over time. $p$ is an array of all state samples. $c_0, c_1$ are the upper and lower constraints, respectively. Subtraction and taking the absolute value is done element-wise.

$$max(|\vec{p}| - |c_0| \cup |\vec{p}| - |c_1|) \tag{3}$$

### D. Cloud infrastructure

The experiments are conducted on two cloud infrastructures; 1) a public cloud with an expected higher RTT and a very large and highly tuned infrastructure 2) an edge cloud with an expected minimal RTT, comprised of a small scale Data Center (DC). We choose to include both for two reasons. One is to observe the trade-off between RTT and compute capacity, if any. The second is to make it possible to distinguish the impact of the cloud from other system components, such as the client implementation and intermediate networks. Although this is a

subset of available cloud providers and their service abstractions and offerings, we deem these two to be representative. The two cloud infrastructures are as follows:

**Edge Data Center (EDC)** is an OpenStack-based DC in Lund, Sweden hosted as a research platform by Ericsson ($1.3km$ ($\approx 0.8mi$) from the plant). It is representative of a proximal private cloud, alternatively, an Edge DC.

**AWS** eu-north-1 (a reasonably proximal AWS region) in Stockholm, Sweden ($480km$ ($\approx 300mi$) from the plant) which is a representative of a COTS public cloud DC.

The plant is connected to the cloud infrastructures with a high-bandwidth and low-latency national back-bone network. The RTT is measured for a set of communication and cloud platform technologies; Internet Control Message Protocol (ICMP) echo, User Datagram Protocol (UDP), Transmission Control Protocol (TCP), gRPC (a cross-platform RPC framework), Representational state transfer (REST), and Function as a Service (FaaS). They are representative of a journey though the Open Systems Interconnection (OSI) stack and the progressive value added by a cloud. At the application layer, Hypertext Transfer Protocol (HTTP) 1.1 and 2 are used in the form of REST and gRPC. For completeness, at the bottom on the stack, the RTT for light in fiber is included, and the theoretical RTT of a single IP packet.

A minimal *echo response* is implemented for each communication technology and hosted on VMs (EC2 t3.micro and OpenStack c1m05) and containers (Hosted on equally dimensioned K8S clusters), when technically permissible. VM and container deployments will from now on be collectively referred to as Infrastructure as a Service (IaaS). Henceforth, we refer to the combination of communication technology, service abstraction, and run-time paradigm as a *deployment*.

Finally, the controller is implemented in Python using CVXGEN[1] [9] and adapted to the above set of deployments, so that its response time can be compared to the RTT measurements. The networked controller is implementation using FaaS, i.e. it is stateless. Its function is requested using REST and native AWS Boto3 calls. When deployed directly on a VM, Flask is used to handle incoming requests. The execution of the controllers is inherently single-threaded and is thus reflective of a lower performance bound. We acknowledge that parallelism in execution and admission can have a different performance outcome but also comes at a greater cost to the operator of a cloud-controlled plant. Further, we have chosen to not include an assessment of intermediate nodes in the routing paths because of many non-observable nodes, especially when entering the cloud infrastructure, and the excessive overhead such measurements adds to the experiments.

For reproducibility and for conducting the experiments at scale, a custom, automated deployment platform is provided. With that platform, we are able to reproduce the experiments and consistently observe the behavior of the system over long periods of time. Note that it is not our intention with this paper to contribute with a universal benchmark of the cloud. Such a study would be too large for this context, and the performance of public and private cloud infrastructures change over time, quickly making such a study obsolete.

[1]A code generator for convex optimizations, found at www.cvxgen.com

## V. Feedback control details

In this section, we provide details on the ancillary controller at the plant and the networked controller in the cloud, used in the feedback control experiments. The below details can be read out of interest or as a reference for reproduction, but are not necessary to understand the bulk of results and the discussion that follows. With the reference system presented here, we proceed to realize the targeted system as presented in Section III-D.

### A. Plant

Our example process is the *ball and beam*, detailed in [15]. The ball and beam process is non-trivial, have many controller implementations, is observable and intuitive to evaluate, and can be compactly detailed. The objective is to position and balance a ball on a rotating beam. The controller sets the angular velocity of the beam. The controller gets two measurements from the plant, 1) the position of the ball on the beam 2) the angle of beam. Finally, the plant inputs a control signal. To be able to control the progression of time and eliminate excessive noise, we simulate the process using the following simple model

$$\begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -sin(\theta) \cdot g \cdot m \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ v \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u \quad (4)$$

Here, $p$, $v$ denotes the ball's position and speed, respectively, while $\theta$ denotes the beam's angle. $m = \frac{5}{7}$ is the mass constant for the ball that we are controlling on the rotating beam, and $g$ is the acceleration constant. The beam's speed is controlled by the input $u$ and a change of beam speed is assumed to be instantaneous. The process model is linearized around $\theta = 0$ and sampled at $20\,\mathrm{Hz}$. This rate is chosen based on the results in Section VI. The minimum feasible control frequency for the plant is $15\,\mathrm{Hz}$. When simulated, the non-linear model in Equation (4) is used with an Euler approximation of $100$ iterations per sample. This creates a small discrepancy between the simulation and the discretized model of the controller. The discrepancy is small but ensures that the control problem cannot be perfectly solved by predicting the discrete response.

### B. Ancillary Controller

Two controllers are derived to create the setup in Figure 2b. The ancillary controller is an unconstrained, Linear–Quadratic regulator (LQR) [6]. The complexity of computing a control decision for the Linear Quadratic (LQ) controller is low. It generates the control signal as $u = Kx_e$ where $x_e$ is the state error and

$$K = \begin{bmatrix} -31.0027 & -13.9077 & 20.4937 \end{bmatrix} \quad (5)$$

The gain matrix is the solution to the minimization of $J(x, u)$ with respect to $u$, where $J(x, u)$ is defined as

$$J(x,u) = \sum_{i=-\infty}^{\infty} x_i^T Q x_i + u_i^T R u_i \tag{6}$$

The cost matrices for the controller are

$$Q = diag([400, 5, 0]), R = 0.25, \tag{7}$$

where $diag(v)$ represents a matrix with the elements of $v$ in the diagonal and all other elements set to zero.

### C. Networked Controller

The networked controller is a Model Predictive Controller (MPC) [11]. The MPC is an optimal controller with the optimization problem formalized as:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & J(x,u) \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k) \\ & h(x) \leqslant 0, g(u) \leqslant 0 \end{aligned} \tag{8}$$

This controller takes the plant state and control signal constraints into account and therefore does not have an analytical solution. The controller must perform a numerical optimization online and $J(x, u)$ is defined over a *horizon*, $N$,

$$J(x,u) = \sum_{i=0}^{N} x_i^T Q x_i + u_i^T R u_i \tag{9}$$

Every request sent to the cloud minimizes the cost function $J(x, u)$. Note that $x_i$ in (9) is a vector that represents the states in (4), $u$ is a vector of control signals and $u_i$ is a scalar. In Equation (8), $f(x_k, u_k)$ is the linear model derived from (4), and $h(x)$ and $g(u)$ represent state and input constraints, linear in the implementation. The state and input constraints are

$$|p| \leqslant 0.53, \ |v| \leqslant 1.5, \ |\theta| \leqslant 0.7, |u| \leqslant 4.4 \tag{10}$$

The objective of the controller is again defined in Equation (7).

The MPC solves the control problem on-line by finding a series of control inputs that represent the optimal control of the process, based on a prediction of the plant state. In minimizing $J(x, u)$ with respect to $u$, the solver produces a sequence of control actions. Only the first control actions, $u(0)$, is output to the process and a new optimization is started for the next sample. The parameter $N$ determines the number of samples the controller predicts and calculates. The complexity of the MPC makes its computational requirements much larger than for the LQR, which is trivially calculated. In addition, the numerical optimization is iterative, and the computations vary over time depending on how many iterations are needed to solve the control problem for the current state of the system. This creates a variable load in the cloud.

### D. Implementation

To provide a time frame in which the controller can send requests and receive responses from the cloud, the networked controller applies a fixed, one sample delay to the control signal. It accounts for the delay using its built in prediction, i.e. at time $t$ the networked controller uses a predicted state $\hat{x}$ to calculate a control signal $u_n$ for time $t + h$, where $h$ is the sample rate. In the basic, straight forward implementation, the controller always requests support from the network. At every iteration of the controller, a request is sent to the network and if the RTT, $\tau$, is less than $h$, the response, $u_n$, is applied in the next iteration. When the request latency is too large, $\tau > h$, it uses the ancillary control. In the second implementation, the controller doesn't always send requests to the cloud and it transitions differently to the local control mode. This version is a fixed horizon variant of the controller in [13].

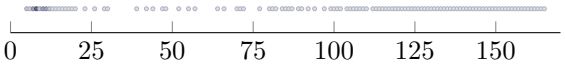



Fig. 3: Registered number of optimizer iterations. The darker cluster at the lower end is where the bulk of requests are.

A benefit of the MPC is that its computational requirements can be scaled. The performance and load of the controller depends on the design parameter $N$ in Equation (9). Adjusting this parameter scales the problem complexity linearly. The solver that computes Equation (8) also requires an unknown and variable number of iterations to find the optimal control sequence for a given $N$, depending on the state of the plant. In our experiments, we observe a range of horizons and define three load scenarios based on the number of iterations. Running the feedback controller for some time reveals a range of iterations that the solver uses with $N = 30$, see Figure 3. Consequently, we define our load scenarios from Section IV-B as; 1) *light* with 5 iterations 2) *medium* with 38-82 iterations 3) *heavy* with 93-152 iterations. There is a range of iterations for the medium and heavy cases. Although the system state is fixed in the optimization, when N changes the number of iterations can also change. We note that the number of iterations primarily change from N=5 to N=10. The change is otherwise insignificant, and does not affect the overall results.

## VI. Results

We now proceed to evaluate Assumptions 1-3 through results from the experiments detailed in Section IV.

### A. Upper frequency bounds and noise floor

With the goal of using cloud-native platforms, testing Assumption 1 entails answering whether or not a significant amount of delay and variance is added for each layer in the cloud. Additionally, evaluating Assumption 2 requires an assessment of the frequency at which the cloud is able to reply timely. Figure 4 shows the RTT progression for our two cloud infrastructures, from the theoretical latency of light in fiber to highly abstract FaaS deployments.

Although a large portion of the response time is consumed in flight in the AWS REST deployments, all deployments are compatible with Assumption 2. Lambda leaves little room for frequencies above the requirement of Assumption 2, while the data suggests multiples of improvement across all deployments in the proximal DC. The platform noise increases when moving up the stack as assessed by Assumption 1. In the lower layers, it looks similar for the two infrastructures, and is large

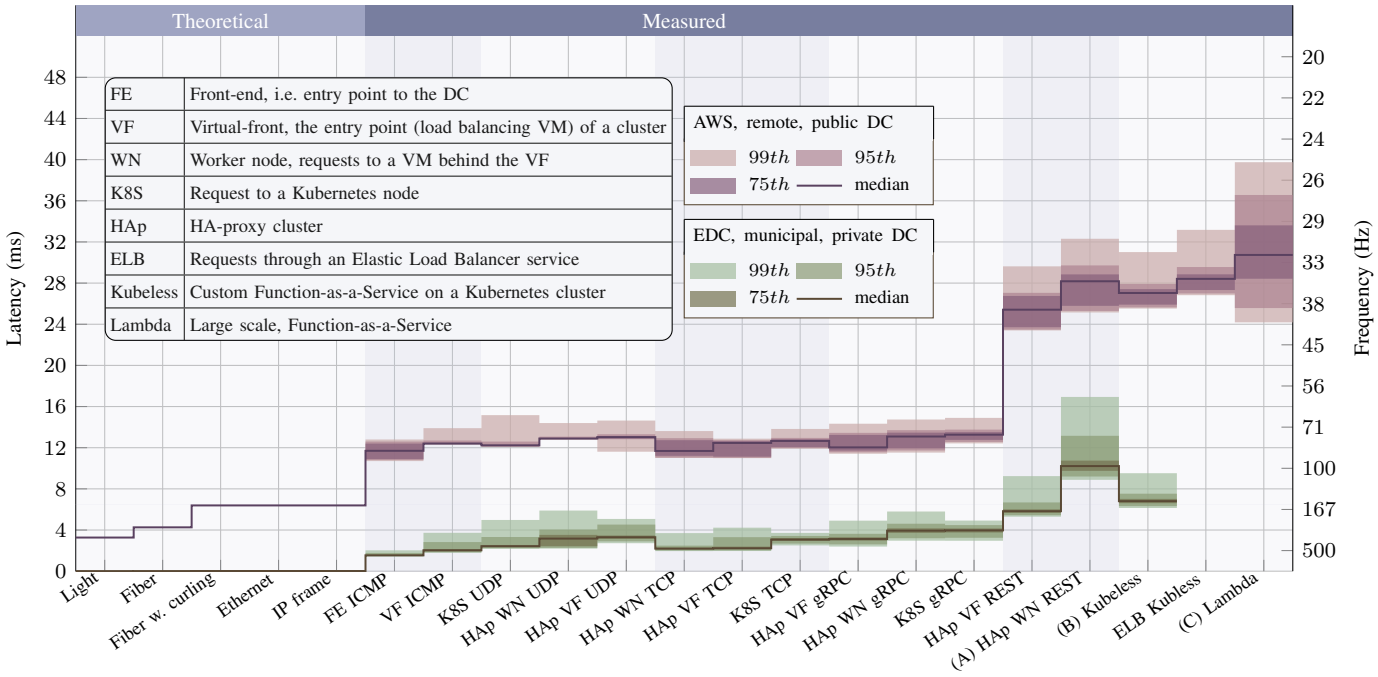



Fig. 4: Echo request progression from theoretical round-trip delays to cloud service response times.

in relative terms for the proximal DC. In absolute terms, in relation to Assumption 2, the noise looks sufficiently small.

The discrepancy from the theoretical latency in fiber to ICMP (first entry into the stack) is an indication of the overhead in the intermediate networks. The increase is 90-fold and 2-fold, for EDC and AWS eu-north-1, respectively, with the theoretical values for EDC being in the order of microseconds. Looking at ICMP, at the bottom end of the IP-stack, at an almost 370 times greater distance (from EDC to AWS eu-north-1), the latency is a factor 8 higher. The RTT over UDP is a factor 4 greater in AWS eu-north-1 compared to EDC. Note also that the RTT's variance in the case of EDC is significantly larger.

When communicating with the compute resources inside a DC, the most significant degradation, in either DC, is when stepping into the application layer, such as adding an HTTP 1.1 header. REST has twice the RTT when compared to TCP, the underlying transport protocol employed by HTTP. Further, gRPC takes advantage of HTTP/2 header compression, and therefore achieves a lower response time, almost half.

REST Application Program Interfaces (APIs) implemented with HTTP is a ubiquitous technology among cloud applications and is the common approach for accessing FaaS platforms. As a design criteria, FaaS is our targeted service model and Figure 4 shows that we can continue to investigate FaaS as an alternative compatible with Assumption 2. It is clear from Figure 4, that the layers of software platforms and opaque infrastructure management policies increase the latency in comparison to ICMP, 5 and 2.5 fold, in EDC and AWS eu-north-1, respectively.

Based on the observed RTT, the achievable response frequency from the cloud varies between $100\,\mathrm{Hz}$ to $500\,\mathrm{Hz}$ and $33\,\mathrm{Hz}$ to $80\,\mathrm{Hz}$, in EDC and AWS eu-north-1, respectively. The theoretical upper bound is the RTT in fiber, which allows for $100\,\mathrm{kHz}$ and $156\,\mathrm{Hz}$, in EDC and AWS eu-north-1, respectively. However, UDP is the first point of access to the compute resources in the cloud, and frequencies of $500\,\mathrm{Hz}$ and $80\,\mathrm{Hz}$ are achieved for the respective cloud infrastructures.

In conclusion, there is no reason to disqualify any deployments just yet but it is worth keeping in mind that these numbers are lower bound since the request payloads are minimal and no valuable computation is done in the cloud. Next, we subject the systems to load in order to evaluate Assumption 1 and the expected RTTs for the experimental control system.

### *B. Noise floor and the response to load*

Having established the upper bound response frequency and minimum variance in Section VI-A, we are ready to evaluate the response frequency and platform noise when executing the controller. Since the REST and FaaS deployments are our targets, we proceed with that subset of deployments, namely, (A) REST over High Availability-proxy (HAp) (B) Kubeless (C) AWS Lambda on eu-north-1. We are interested in determining the response rate that can be achieved in these deployments and how a load affects the platform noise.

The plots in Figure 5 show response times per controller horizon for each load scenario and deployment. A linear increase in computation time is expected with an increasing horizon ($N$, on the x-axis). This is clearly visible in Figure 5, and apart from Lambda, there are only small discrepancies from this linear response.

There are notable differences in response times between the deployments. Although the execution times visually start off at a similar level, the gradient on EDC, going from light to heavy load, is significantly steeper than on AWS, specifically,

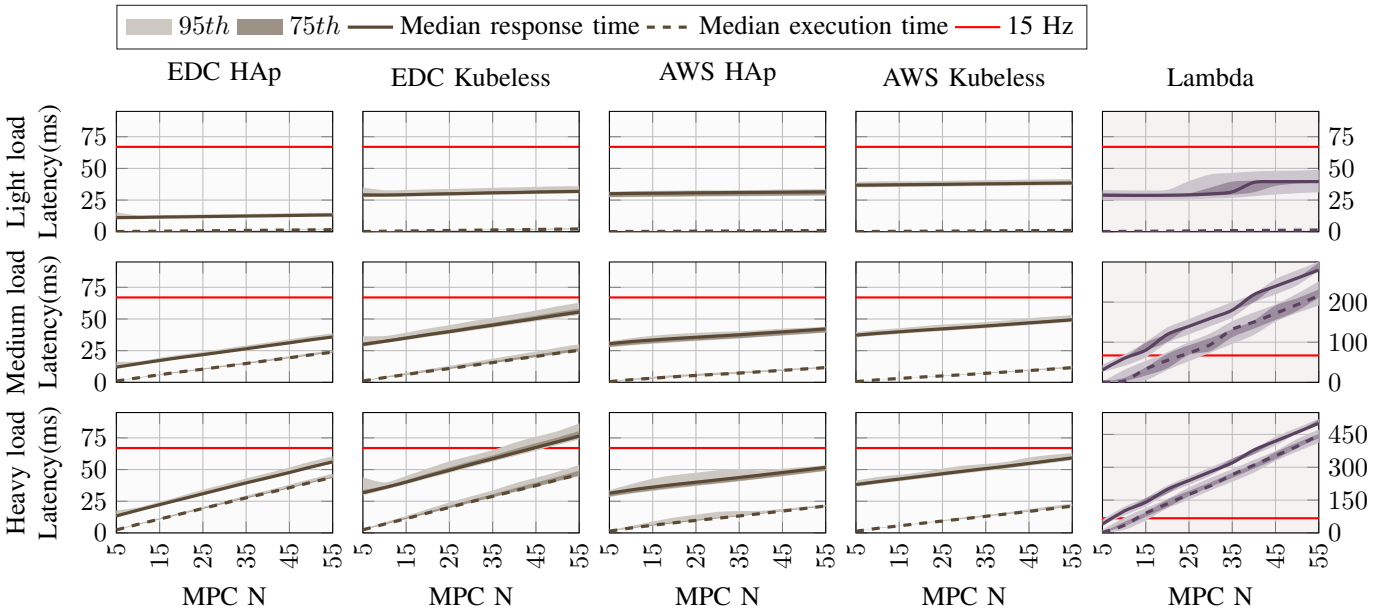



Fig. 5: Latency vs. horizon, showing median and percentiles.

on average 2.3 greater, consistently across the deployments. EDC variance also increases with the load. This is particularly evident in the EDC Kubeless deployment. However, on AWS the trend is not as pronounced. Instead, there is an initial increase followed by a decline in variance for AWS HAp, which peeks between $N = 15$ and $N = 35$ for both execution and response time in the heavy experiment. It is also clearly visible that the Kubeless deployment on EDC incurs a large response time penalty and that execution times are affected negatively, especially in terms of variance. This is reminiscent of the aforementioned unavoidable platform noise.

There is a large offset in response time between EDC HAp and EDC Kubeless. This is in contrast to what is observed in Figure 4, where Kubeless has a faster response. The same effect is seen for AWS although less pronounced. Looking at the light load and small $N$, the HAp deployment is inline with what is expected, given Figure 4. The Kubeless deployments for some reason seems to incur a penalty. This may be related to a configuration issue but can also be attributed to a form of platform noise, especially since the Kubeless deployment is the more complex of the two.

The visually most striking result is observed for Lambda. In the light load scenario, the execution time is comparable across all deployments but the Lambda response time has a clear mode shift between $N = 30$ and $N = 40$. Note also that the variance in response time is significantly larger than for the other deployments. The medium complexity case presents the same attributes. Here, multiple modes are clearly present, at around $N = 15$ and $N = 35$. Also, the execution time is 20 times that of HAp and Kubeless, and Lambda has gained significant variance, across the range of controller horizons. These phenomena point to a complex cloud-native platform, built to be shared by many at a large scale, handle massive parallel requests, and not carry a single real-time thread.

Except for Lambda and the response time difference between HAp and FaaS, the results in Figure 5 mostly points to a performance difference between the allocated VMs in the two different clouds. To get further insights into the first four deployments they were subjected to a static load for a few hours. Figure 6 shows the median of the execution time, using a sliding window, for a load scenario with $N = 30$ and more than 300 iterations in a single optimization. The median is accompanied by area plots that show the minimum and maximum execution time within the window. What is clearly visible here is that there is a very notable difference in execution time variation between the HAp and Kubeless deployments. There also seems to be more platform noise in the EDC HA deployment than in the AWS HAp deployment. This could be a direct consequence of the much longer (double) average execution time. It is also notable that although the median is almost a straight line for AWS HAp, the brown area plot in the background shows that there are occasions of extended execution time. We attribute some variation and sporadic delays to the non-real-time property of the system, and equate it to any ordinary general purpose computer. However, the bump in execution time in AWS HAp at close to 6 hours is different. Here, the median changes over a longer period of time which indicates that temporarily (although for several minutes) the available computation time decreases.

Another interesting feature of Figure 6 is the difference between the HA Proxy and the Kubeless deployments. Since these deployments are all on different VMs, it is possible that differences in hardware and over-provisioning (i.e. sharing with other users) give rise to such discrepancies. However, note that the effect is consistent and equally present on both infrastructures. This enforces Assumption 1. Even though the experiments execute one service at a time, the additional installed routing, software, monitoring etc that goes into a

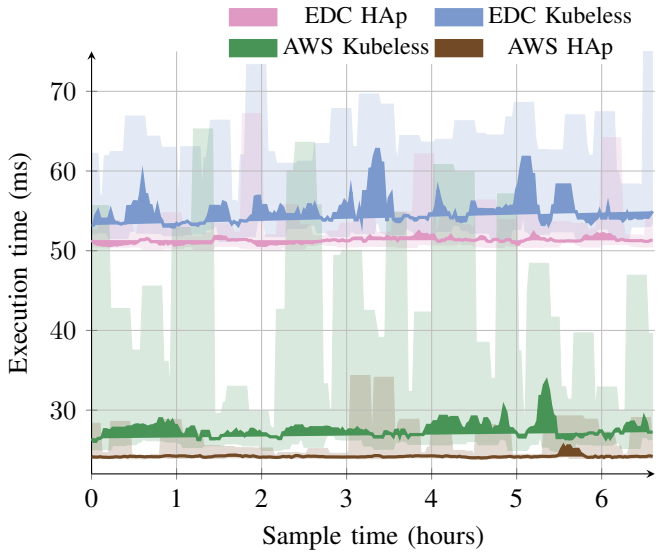



Fig. 6: Execution time of a static load evaluated with two requests every 3 minutes, separated by a 0.5 seconds delay. Solid lines show a sliding window median over 10 samples. Shaded areas show the max and min values in the window.

Kubeless deployment incurs a distinguishable overhead.

To sum up, there is a performance gap between Lambda and the other deployments. In a large scale public service such as Lambda, we expect variations over time, due to for example congestion, but we have also observed service dynamics in the short term, based on our request rate and load. Due to the rapidly diminishing usefulness as the load increases, Assumption 2 does not seem to hold for Lambda. If control frequencies are selected close to the median, Assumption 1 should be relevant for the end result, in which case our cloud native FaaS is at a disadvantage compared to HAp, as seen from the variance observed in Figure 6.

In terms of the example control application, when requiring only a few iterations (light load), the controller can be actuated at a rate of between 25 Hz to 90 Hz depending on the deployment, across all $N$. With the medium and high load scenarios the response rates are between 4 Hz to 90 Hz and 2 Hz to 66 Hz, respectively. This assessment does not take into account the probability of consecutive long delays or the effect that the variance causes. In the final experiments, we keep all deployments, also Lambda, run at 20 Hz and use $N = 50$ to add some degree of loss, not only attributed to network delay.

### C. Examining the closed loop control

We proceed to evaluate the example in Section V, constructed as in Section III-D, and conducted as detailed in Section IV. Kubeless and AWS Lambda represent our cloud-native FaaS, the primary target of interest. In the set we also include the IaaS deployment for REST since this a compatible and comparable service. The local ancillary LQ controller acts as our performance baseline, i.e., the system presented in Section V, with no connectivity to the cloud. We choose to maintain the operating frequency at 20 Hz, i.e. a sample rate and response deadline of 50 ms, from [15]. Further, the MPC horizon is $N = 50$. Based on the findings so far, this should allow successful operation on all IaaS services, while Lambda should be able to provide support for the lower loads.

The experiment results should tell us at least three things; 1) whether the reference system works as expected in terms of meeting expectations from the benchmarks. 2) the response of the system when only the lower loads are able to respond timely, which is what we expect to see from Lambda. 3) how well the switching mechanism works.

Table I provides an overview of the control outcome for the five deployments, in relation to using only the ancillary controller. The second column shows the fraction of time the networked controller was used. Networked control is requested continuously in the first five cases and this column thereby shows the ratio of successful requests. In the last two cases, requests seldom fail, and the number can be directly related to the fraction of time that the controller requests cloud support.

To put results from Section VI-A in context, using the load profile from Figure 3 and the measured response time in Figure 5, we can calculate the expected response rate for each deployment, $E(\tau)$, presented in Table I's 3rd column. From Table I, most noteworthy is the AWS Lambda deployment. Here, only 16% of the requests get responses from the networked controller. This is in line with the results from Section VI-B, where the response times for AWS Lambda were often well above 50 ms, $E(\tau) = 60.82$. Further, the EDC and AWS HA deployments utilized the cloud 95% and 96% of the time. These numbers are 82% and 85% for EDC and AWS FaaS. Three metrics follow, detailed in Section IV-B.

The RAE in Table I, which shows the long term efficiency of the controller, is improved by all cloud-based networked controllers, as expected. In fact, the EDC Kubeless deployment scores a 40 % improvement, same as the ideal network, at 82 % response rate. Even Lambda, with its low response rate, scores a 12 % improvement.

The networked controller knows the system's constraints and consequently attempts to strictly enforce them. There are two complicating disturbances that may cause this to fail: a small model error and network loss. The RAV metric measures the extent to which the controller fails to enforce the constraints by summing up the amount of constraint violation over the course of the experiment. A RAV value above zero is undesirable and could be risky. Being worse than the baseline would be strictly unacceptable. AWS Lambda makes extensive use of the local controller and therefore gets a relatively high RAV. The following four deployments score almost perfectly due to the high response rates from the networked controller.

Several deployments perform on par with ideal values (RAE of 0.6 and zero RAV). Observed with the same metric, Lambda also performs satisfactory compared to the baseline. However, it is important that the controller achieves a low *maximum constraint violation*. This is measured by the RMCV metric, which tells us whether temporary degradation can cause system failure. As seen in the table, this is where the basic cloud controller architecture fails. A simulated RMCV

| *System* | *Cloud* | $E(\tau)$ | *RAE* | *RAV* | *RMCV* |
|---|---|---|---|---|---|
| *Baseline* | 0.00 | 0.00 | 1.00 | 1.00 | 1.00 |
| *Lambda* | 0.16 | 60.82 | 0.88 | 0.68 | 2.47 |
| *EDC HAp* | 0.95 | 15.66 | 0.60 | 0.01 | 2.28 |
| *EDC Kubeless* | 0.82 | 34.67 | 0.59 | 0.01 | 1.90 |
| *AWS HAp* | 0.96 | 31.95 | 0.60 | 0.01 | 2.28 |
| *AWS Kubeless* | 0.85 | 39.78 | 0.59 | 0.01 | 2.28 |
| *AWS HAp** | 0.40 | 15.40 | 0.90 | 0.09 | 0.31 |
| *Lambda** | 0.40 | 14.40 | 0.90 | 0.09 | 0.50 |
| *Ideal Network* | 1.00 | 0.00 | 0.60 | 0.00 | 0.00 |

TABLE I: Performance measures for feedback control system. All measures are in relation to the local LQ regulator. A * marks the second controller implementation as described in Section V-D.

above 2 indicates an unrecoverable error in the real plant, and out of the first six cases in Table I, only the baseline looks safe. In part, Assumption 3 states that switching cannot be assumed safe just because the two individual systems are reliable, as is the case with the baseline and the ideal networked controller. Even the limited loss of 4% for AWS HAp is enough to cause repeated failure during an experiment of 3.5 hours.

The situation is remedied in the last two cases in the table. Here, we have partially applied the technique in [13] and let the experiment run for 7 hours. Ideal performance isn't reached with this controller, in terms of RAE, but in return it keeps RAV and RMCV low. The controller also sends fewer requests to the cloud and, due to different dynamics, does not require the high load scenarios under ordinary circumstances. For this reason, it remains reliable and achieves efficiency gains, even when deployed on Lambda. We see also that Lambda is better utilized in terms of successful requests but achieves a slightly lower RAE. AWS HAp makes less use of the networked controller but also scores a significantly higher RAE than its previous counterpart. One thing that sets these two deployments apart is that if a situation is created which requires a heavy computation, which can happen due to an external disturbance, we now know from the previous results that AWS Lambda is likely to fail while AWS HAp has a good chance of retaining good performance.

## VII. Conclusions

In this paper, we advocate a return to core performance assessments of cloud latency and *platform noise*, to determine the effects and limitations of cloud-based critical control of constrained dynamic systems, stemming from the loss of real-time support. We provide a measure of latency and noise through the software stack of two clouds, from fiber up to the cloud native service. With knowledge of the limitations of the request frequency at various levels, we deploy a reference feedback control system, implemented using FaaS. Results show that the basic latency assessment holds in practice, with distinctly measurable efficiency improvements, but it also identifies a challenge to provide techniques that eliminate or reduces the impact of an increased RMCV. This metric relates to system *constraints*, which are not always present in the control solution but of general relevance in practice. Through an improved design we reduce the request load of the cloud controller and, while retaining measurable efficiency gains, provide a reliable closed-loop system. The results speak for the idea of using the cloud for temporary performance boosts in critical control loops, also at relatively high frequency and without real-time guarantees.

We examined the usefulness of a set of cloud platforms and presented a strategy towards enabling the use of cloud for demanding real-time systems. Our view is that this builds confidence in a COTS-like use of the cloud, for control over the cloud, as a way forward in Industry 4.0.

## References


[1] Tarek Abdelzaher, Yifan Hao, Kasthuri Jayarajah, Archan Misra, Shuochao Yao, Per Skarin, Dulanga Weerakoon, and Karl-Erik Årzén. Five challenges in cloud-enabled intelligence and control. *ACM Transactions on Internet Technology (TOIT)*, Oct. 2019.

[2] Andreea B Alexandru, Manfred Morari, and George J Pappas. Cloud-based MPC with encrypted data. In *IEEE Conference on Decision and Control (CDC)*, 2018.

[3] Hongyu Pei Breivold and Kristian Sandström. Internet of things for industrial automation–challenges and technical solutions. In *Int. Conf. on Data Science and Data Intensive Systems*. IEEE, 2015.

[4] Hasan Esen, Masakazu Adachi, Daniele Bernardini, Alberto Bemporad, Dominik Rost, and Jens Knodel. Control as a service (CaaS) cloud-based software architecture for automotive control applications. In *ACM Int. Workshop on the Swarm at the Edge of the Cloud*, 2015.

[5] Rachana A Gupta and Mo-Yuen Chow. *Networked Control Systems: Theory and Applications*. Springer-Verlag London, 2008.

[6] Rudolf E Kalman. Contributions to the theory of optimal control. *Boletin de la Sociedad Matematica Mexicana*, 1960.

[7] Yu Kaneko and Toshio Ito. A reliable cloud-based feedback control system. In *IEEE Int. Conference on Cloud Computing (CLOUD)*, 2016.

[8] Philipp Leitner and Jürgen Cito. Patterns in the chaos—a study of performance variation and predictability in public IaaS clouds. *ACM Transactions on Internet Technology (TOIT)*, Apr. 2016.

[9] Jacob Mattingley and Stephen Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, Mar. 2012.

[10] István Pelle, János Czentye, János Dóka, and Balázs Sonkoly. Towards latency sensitive cloud native applications: A performance study on AWS. In *IEEE Int. Conference on Cloud Computing (CLOUD)*, 2019.

[11] James B Rawlings and David Q Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Pub., 2009.

[12] Jan Rüth, René Glebke, Klaus Wehrle, Vedad Causevic, and Sandra Hirche. Towards in-network industrial feedback control. In *ACM Morning Workshop on In-Network Computing*, 2018.

[13] Per Skarin, Johan Eker, and Karl-Erik Årzén. Cloud-based model predictive control with variable horizon. In *Elsevier Int. Federation of Automatic Control (IFAC)*, 2020.

[14] Per Skarin, Johan Eker, and Karl-Erik Årzen. A cloud-enabled rate-switching MPC architecture. In *IEEE Conference on Decision and Control (CDC)*, 2020.

[15] Per Skarin, William Tärneberg, Karl-Erik Årzen, and Maria Kihl. Towards mission-critical control at the edge and over 5G. In *IEEE Int. Conference on Edge Computing (EDGE)*, 2018.

[16] Axel Vick, Jan Guhl, and Jorg Krüger. Model predictive control as a service—concept and architecture for use in cloud-based robot control. In *IEEE Int. Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016.

[17] Won-jong Kim, Kun Ji, and A. Srivastava. Network-based control with real-time prediction of delayed/lost sensor data. *IEEE Transactions on Control Systems Technology*, Jan. 2006.

[18] Yong woon Ahn and Albert Mo Kim Cheng. Mirra: Rule-based resource management for heterogeneous real-time applications running in cloud computing infrastructures. In *Presented at the Int. Workshop on Feedback Computing*, 2015.