2017 American Control Conference
Sheraton Seattle Hotel
May 24–26, 2017, Seattle, USA

Event-based Control as a Cloud Service

Alaa Eldin Abdelaal, Tamir Hegazy and Mohamed Hefeeda

Abstract— **Event-based control has gained significant interest from the research community in recent years because it allows better resource utilization in networked control systems. In this paper, we propose an architecture for offering event-based control as a service from the cloud, which not only improves resource utilization but also reduces the cost and setup time of large-scale industrial automation systems. Providing event-based control from the cloud, however, poses multiple research challenges. We address two of the main challenges, which are the network delays and failures introduced because of moving the controller far away from the plant. We present a delay mitigation technique that maintains the stability and performance of the control system. Our results using commercial clouds show that our delay mitigation technique can handle large communication delays up to several seconds with practically zero effect on the main performance metrics of the system. Moreover, our proposed fault tolerance approach can effectively handle network failures even if the controlled system is thousands of miles away from its cloud controllers.**

I. INTRODUCTION

The classical approach of control systems is based on periodically performing the control actions even if there is no real need for this; for example, when the system has already reached the steady state. To better utilize the system resources, event-based control has been proposed [1], [2]. Such approach can be useful especially when the control system is deployed in resource-constrained environments. In event-based control systems, the control action is only performed when needed; for example when the error is greater than some threshold.

Several researchers highlight the benefits of adopting the event-based control approach especially the savings achieved in both energy and network communication. For example, Ploennigs *et al.* [3] show that this approach can save around 80% of the energy of devices in the context of building automation and control systems.

On the other hand, cloud computing has the potential to improve current automation and control systems. In a recent survey, Kehoe *et al.* [4] highlight the increasing interest in this area by showing the benefits of using cloud computing for both higher and lower level control functionalities. For this latter category, Hegazy and Hefeeda [5] perform a study to calculate the potential savings in both cost and time that can result from offering the control as a service from the cloud. They conclude that for large–scale automation systems, reductions up to 57% of the cost and up to 85% of the start–up time can be achieved. The cost savings come from replacing hardware devices like physical controllers and control cabinets with virtual machines in the cloud and from reducing the number of site visits required by engineers. The time savings come from reducing the time required to assemble and wire hardware devices together because many of them are not needed. This also applies to the shipping time of these devices from engineering locations to the location of the automation system. Moreover, in the same study, the authors highlight several benefits of the agility of the cloud computing model to automation plants. For example, cloud controllers can be deployed in different data centers for different cloud providers which gives more flexibility to the owners of automation systems to choose from a wider range of options. In addition, the complexity of the automation systems deployment is significantly reduced because of the elimination of most of the wiring needed in the system.

In this paper, we propose providing event-based control as a service from the cloud. Our motivation is not only to better utilize the control system resources and save energy, but also to realize the promising advantages of moving the controllers to the cloud in terms of cost and time savings. In order to get the most of these advantages, we have to address multiple challenges including dealing with possible network delays and failures that can affect the stability, performance and reliability of the control system. In this work, we show how the *network delays* and *failures* can be handled in such a way that guarantees the stability and performance of the event-based control system.

To this end, this paper makes the following contributions: first, we propose a delay compensation technique to handle network delays. Our technique employs a delay estimator that feeds the system model with the values of the delay. We show that using our technique does not violate the stability guarantees of the controlled system. Second, we present a fault tolerance technique that makes the system capable of dealing with network failures. This is done by running redundant controllers asynchronously on virtual machines on the cloud that can be geographically far apart from one another. Finally, we implement the proposed techniques in an industry-standard system design software, LabVIEW [6], and we deploy it on different locations of the Amazon cloud. We show that the proposed cloud event-based control service can effectively control distant real systems even under variable Internet delays, and packet losses. Moreover, our results show that up to 88% of the communication messages between the plant and controller can be saved with no noticeable degradation in the control system performance.

This work is partially supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Alaa Eldin Abdelaal and Mohamed Hefeeda are with the School of Computing Science, Simon Fraser University, Burnaby, BC, Canada. {alaa_eldin_abdelaal, mhefeeda}@sfu.ca

Tamir Hegazy is an independent scholar. tamir.hegazy@gmail.com

In the rest of the paper, we discuss the related work in Section II. In Section III, we present the event-based cloud control model. Then, we present our approach to solve the problem of network delays in Section IV, and failures in Section V. We evaluate the proposed approach using LabVIEW and Amazon cloud services in Section VI and we conclude the paper in Section VII.

II. RELATED WORK

One of the early works that considers providing the control functionality as a cloud service is [7]. In this work, Givehchi *et al.* evaluate the approach of moving a Programmable Logic Controller (PLC) to the cloud using a simple discrete process that is, inverting an input signal. A similar study is presented in [8], where the authors propose an architecture for providing control as a service. However, this paper does not discuss any strategy to mitigate the delays or failures.

In [9], Colombo *et al.* present a migration method from a scan-based PLC to event-based Service Oriented Architecture (SOA) system. They show that this is possible except for the hard real-time control tasks. Vick *et al.* [10] identify the motion control sub-tasks of an industrial robot and divide them into soft and hard real time sub-tasks. Soft real time sub-tasks are then moved to the cloud. The rest of the tasks, like position, speed and current control of the robot are done on site not on the cloud. However, the authors do not propose any techniques to deal with network delays and failures. The same limitation applies for the work in [11].

Didic *et al.* [12] test the feasibility of using the cloud in closed loop control systems. They propose a delay mitigation mechanism. However, their work is based on the periodic control approach. Another delay compensation technique is presented in [5]. The authors also proposed a distributed algorithm to handle the failures in their proposed architecture. Their methods show good performance when tested on a soft real-time process from the industrial automation domain, but again this work is based on the periodic control approach.

Based on our review of the event-based control literature, we find that most of the works that addressed the delay problem in the context of event-based control such as [13] and [14], have a main drawback. Those works employ state feedback assuming that full state information is available, which is impractical. On the other hand, the works [15] and [16] make more practical, yet conservative, assumptions, which still limits the applicability of moving controllers to the cloud. In [15], no packet losses or reordering is assumed. In [16], a local controller is required, while in this paper, we move the control functionality altogether to the cloud.

Just like the delay problem, most of the proposed methods to solve the packet loss problem in the context of event-based control are based on state-feedback such as [13] and [17]. As far as we are concerned, only two works [18] and [19] addressed this problem in the output-feedback case. In [18], the authors assume that packet losses only occur in the communication channel from the plant to the controller. The other channel from the controller to the plant is assumed to be a perfect one. A similar assumption is made in [19].

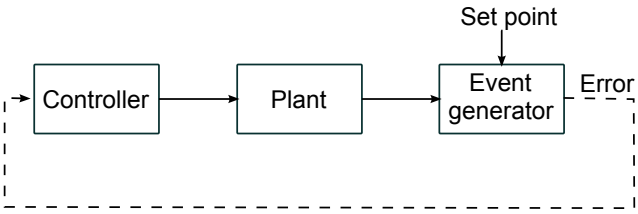


Fig. 1: The structure of the event-based control system.

The authors target a class of denial of service (DoS) attacks. However, the authors assume that the DoS attacks happen only in the communication channel between the sensor and the controller. In practice, this is not a realistic assumption because packet losses can happen in either of the two channels (from the controller to the plant and vice versa).

To the best of our knowledge, the work done in the area of cloud automation with the goal of moving the *control layer* functionality to the cloud is limited. Most of the previous works can be considered as feasibility/pilot studies to explore this area. Moreover, for event-based control, we are not aware of any work that addresses the challenges of moving the controllers to the cloud, especially handling the delays introduced by the Internet and the control and link failures.

III. SYSTEM MODEL

A. Event-based Control Structure

The considered event-based control system structure [20] is shown in Fig. 1. Solid lines in Fig. 1 represent time-based communications and dashed lines represent event-based ones. Moreover, in the presented structure the controller itself is assumed to be a traditional time-based one.

The structure consists of a controller, plant and event generator. The controller is assumed to be a PI controller which is the most common in the industry [21], [22]. The plant is assumed to be a First Order Process with Dead Time (FOPDT) whose general transfer function is shown in (1) where K is the process gain, T is the process time constant and L is the process delay (dead time).

$$P(s) = (\frac{K}{Ts+1})e^{-L}. \quad (1)$$

Higher order processes can be easily approximated to an FOPDT process [23].

The third component of the event-based control structure is the event generator. It is this component that characterizes the system as an event-based one. The event generator receives the process variable from the plant and compares it with the set point to calculate an error value. It decides whether the error should be sent back to the controller based on a sampling algorithm. The used sampling algorithm is called Symmetric Send on Delta (SSOD). In this algorithm, a comparison is done between the current error and the last one. If the absolute value of the difference between the two errors is larger than a tunable parameter Δ, then the current error value is approximated to the nearest multiple of Δ and sent to the controller. Otherwise, nothing is sent from the event generator to the controller.

The main reason for considering the above event-based structure is that in [20] the authors provide the sufficient conditions of the controller parameters that make the system stable without having limit cycles.

B. Proposed Event-based Control Cloud Service

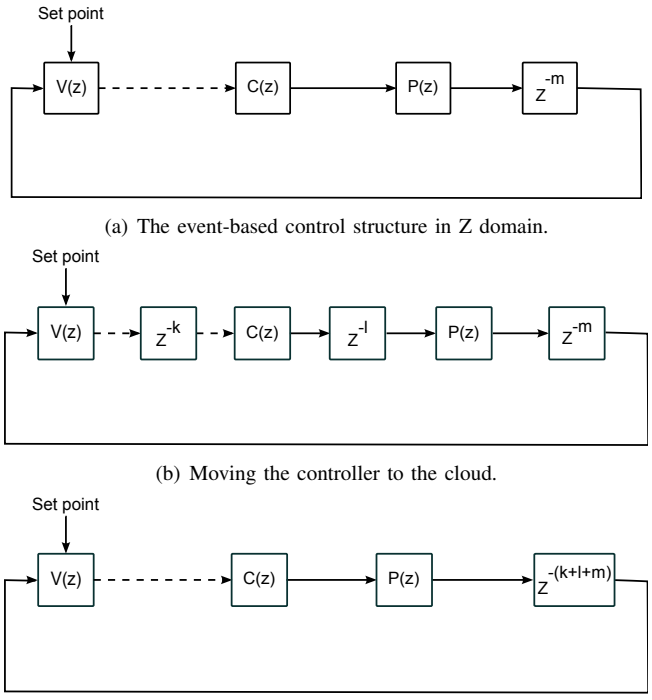
The proposed event-based control cloud service consists of cloud controllers. These controllers are software modules implementing event-based controllers, such as an event-based version of the PID controller in [1] and [24]. Multiple modifications need to be done to handle Internet delays, packet losses, and failures, and to ensure that the control-theoretic performance guarantees are achieved. The controllers are deployed on virtual machines (VMs) and multiple of them can run on the same VM. Our proposed service makes use of the control I/O interface which in many cases is embedded in modern sensors/actuators at the controlled system side. The control I/O interface communicates with the cloud controllers by receiving control actions. These actions are then relayed to actuators of the controlled system. The plant output is then sent to the event generator which in turn decides whether an event has occurred and consequently sends the sampled error signal to the cloud controller. Most recent industrial control I/O devices can communicate over standard Internet protocols. For example, many I/O interfaces support an industry-standard protocol called Modbus, which runs on top of TCP [25]. Similar to HTTP, Modbus is an application-level protocol used to send commands to read/write various registers in the I/O device.

IV. Proposed Delay Compensation

We present our solution to the network-induced delay problem in the case of event-based cloud control. The key idea of our approach is that by moving the controller to the cloud, the network delay problem can be reduced to the problem of controlling an FOPDT process in an event-based manner. This problem is solved in [20] in a way that guarantees the stability of the control system. In our approach, the dead time parameter of the FOPDT includes the estimated network delays. Without reasonably accurate estimation of these delays, the system response is likely to exhibit overshoots and instability [26]. Therefore, an Internet delay estimator is used to feed the system model with these delays.

We start with the general structure of event-based control as shown in Fig. 1. This structure can be rearranged and put in the Z domain as in Fig. 2(a). Here, the plant block in Fig. 1 is divided into two blocks; $P(z)$ representing the first term of the FOPDT model in eq. 1 and Z^{-m} representing the exponential term in the same model which refers to the dead time. $C(z)$ and $V(z)$ represent the transfer functions of the controller and the event generator, respectively.

By moving the controller to the cloud, network delays are introduced between the event generator and the controller on one hand and between the latter and the plant on the other hand. These delays are added to the structure using the blocks Z^{-k} and Z^{-l} as shown in Fig. 2(b), where



(a) The event-based control structure in Z domain.

(b) Moving the controller to the cloud.

(c) Reducing the problem to the one solved in [20].

Fig. 2: The proposed approach to solve the delay problem.

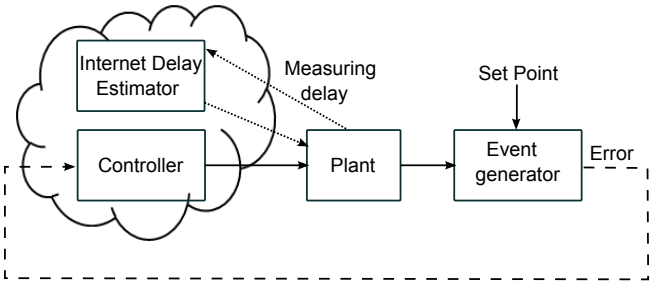


Fig. 3: The proposed model after adding the Internet delay estimator.

k and l represent the values of the delays. We alter the arrangements of the blocks in the feedforward direction and we can see in Fig. 2(c) that the delay problem is reduced to the problem of controlling a first order process with dead time. This is the same problem solved in [20]. The increase in the delay effectively means increasing the dead time of the process. This causes the settling time of the system response to increase while the other performance metrics remain the same as in original system.

Now, we propose adding an Internet delay estimator to measure the Internet/network delay and consequently feed the values of k and l to our model. The event-based cloud control model after adding the Internet delay estimator is shown in Fig. 3.

The Internet delay estimator estimates the roundtrip delay between the controller and the plant using an exponentially moving average for the network delay mean D_i where i represents the current discrete time instance. Similarly, the

estimator employs another exponentially moving variance for the delay variance V_i. D_i and V_i are used to calculate the delay value D_c according to:

$$D_c = \frac{D_i + hV_i^{0.5}}{T_s}, \quad (2)$$

where T_s is the sampling period and h is a positive parameter used to adjust the system response in case of delays greater than D_i. Thus, delay compensation is performed by updating the value of inherent delay block to include communication delays.

V. Proposed Fault Tolerance

In the proposed event-based cloud control service, various types of failures may occur e.g., virtual machine crashes and link failures. These types of failures will result in missing packets between the controlled system and its controllers. The basic idea to handle these failures in our system is by deploying redundant controllers on different virtual machines from different cloud locations. We assume that there are multiple links connecting the plant to the Internet and at least one of them remains up all the time.

The proposed solution for handling failures is based on the Reliable Cloud Controller (RCC) algorithm proposed in [5], which is designed for periodic control. We extend it to support event-based control systems. The RCC algorithm consists of 4 steps. *The first one is the initialization*. The second is the *polling step* where the controllers receive the information from the I/O interface. The third step is the *computing step* where each controller computes its control signal. Furthermore, in this step each controller decides whether it will be the engaged controller. In the last step, *the conditional acting step*, the engaged controller only sends its control signal to the plant. The algorithm gives each controller an *ID* such that the ID of the primary controller is 1 and the secondary controller's ID is 2 and so on. Moreover, each controller has a corresponding *last action age* variable. Each one of these variables is reset to zero each time its corresponding controller sends a control signal to the plant. Otherwise, its value increases by one every sampling period. Furthermore, each controller has an *engagement threshold*. A controller becomes engaged if its engagement threshold is lower than the last action ages of all other controllers whose IDs are less than the ID of this controller.

The algorithm employs the bumpless transfer technique from control theory [27] to achieve smooth handover between controllers. This is important for the system response because redundant controllers may have different output values. So, when a failure happens, the system response may suffer from a bump due to the difference between the output values of the newly engaged controller and the failed one. In the case of using a PID controller, the bumpless transfer is achieved by adjusting the integrator's initial value [28].

We first show that by using the RCC algorithm, we address the packet loss in the two communication channels; i.e., from the controller to the plant and from the sensor to the controller. Consider the polling step of the RCC algorithm while having two controllers, a primary and a secondary (without loss of generality). If there is a timeout, meaning that the message from the sensor to the primary controller did not reach the controller, the primary controller goes to the first step (the initialization step) without performing the control logic. As a result, it will not send anything to the plant. Therefore, The plant will increase the last action age of the primary controller. Once this last action age value reaches the engagement threshold of the secondary controller, this latter will engage. So, this means that the "no communication" state between the sensor and the controller is handled using the polling step of the RCC. The other "no communication" state between the controller and the plant is also handled in a similar fashion. This highlights the difference between our work and previous works in this context. They deal only with one communication direction (from sensors to the controller) but we deal with the other as well (from the controller to the plant).

Applying the RCC as it is to our event-based structure can lead to the following problem. When the primary controller fails, the plant will use the primary controller's last control signal as its input during the failure period of the primary controller. As a result, the plant output will settle to an intermediate value and hence the difference between the current and last errors will be zero. So, the plant will not send anything to the secondary controller whatsoever because the event generator only generates an event if the difference between the errors is greater than Δ. This means that the secondary controller will never be engaged.

To address this problem, we generate *regular events* (known as heartbeats) every *n* sampling periods in addition to the original triggered events. These regular events will only be generated if the time between two consecutive triggered events is more than the period of sending those regular events. In other words, if some time passes without generating triggered events, then a regular event should be generated. Otherwise, this time is reset to zero. Having done this, when the primary controller fails as in the above case, after some time passes, a regular event will be generated and consequently the secondary controller will become engaged.

The worst–case scenario in our proposed approach is when the triggering event condition is always false due to the failure of the engaged controller. In such a case, the plant operates only on the generated regular events until the engagement of the standby controller.

In the following, we analyze the proposed fault tolerance approach in more details considering the worst–case scenario.

First, we note that as long as there is a healthy controller and a working link between this controller and the plant, then the normal operation of the plant is guaranteed. For example, assume that there are more than one healthy controller with working links between them and the plant. Until a new regular event is generated, the last action age of more than one controller may exceed the engagement thresholds of the subsequent ones. So, when a regular event is generated, there may be more than one controller that will be engaged at the

same time. But, as long as the smooth handover is active, the first control signal of all these newly engaged controllers will be the same. Moreover, the controllers are sending their control signals periodically to the plant. This means that every sampling period, the last action age of each of those controllers is reset to zero. So, when the next regular event is generated, only the controller with the smallest ID will continue working and the rest will go to the standby mode.

Next, we study the steady state error and overshoots in the case of controller failures. If the original tuning of our structure without failures leads to no overshoots or steady state errors, then our proposed fault tolerance approach guarantees the same performance under failure as long as there is at least one healthy controller. The reason is that, based on the smooth handover in the double redundancy case (without loss of generality), when the primary controller fails, the first control signal of the secondary controller will be the same as the one that the primary controller would have produced if it did not fail. Moreover, the secondary controller when getting engaged will produce the same sequence of control signals as the ones that the primary would have produced if it did not fail. This is because the control parameters in the primary and secondary controllers are the same. Furthermore, during the failure period of the primary controller, the plant input will be the last received control signal sent by the primary controller. So, during this period and before the engagement of the secondary controller, the plant has the same input and hence produces the same output. So, effectively the same sequence of control signals will reach the plant but after some delay. In other words, the packet loss problem can be reduced to the problem of having the same packets reaching the plant but after some delay.

This delay is finite, and has a known upper bound. This upper bound (in the case of having double redundancy without loss of generality) equals F+(D_2+RTT_2)/T_s-1 sampling periods where F is the number of sampling periods before generating a regular event, D_2 is the engagement threshold of the secondary controller, RTT_2 is the round trip time between the plant and the secondary controller, and T_s is the sampling period of the plant. This delay means an increase of the dead time parameter (L) in the FOPDT model. This means that the only change in the response will be a shift of the response by the amount of this delay. All the other performance characteristics of the original signal are preserved since the rest of the FOPDT parameters are the same. Based on the above, the proposed fault tolerance method will preserve the same performance measures of the original response with no failures. However, there will be an increase in the settling time. This increase is upper bounded by the above mentioned formula when a single failure happens in the system response.

Another note is that no change will occur in the system response upon the recovery of a controller with lower ID than the currently engaged controller. Without loss of generality, consider the case when the secondary controller is engaged and the primary controller recovers. Because of the smooth handover, the first control signal of the primary controller will be the same as the control signal of the secondary controller. While waiting for a new event to be generated, the two controllers will treat the last received output from the plant as their input signal. As a result, the two controllers will continue producing similar control signals because the control parameters in the two controllers are the same. During this waiting time, the two controllers send their control signals every sampling period to the plant and hence resetting their corresponding last action age variables. When the plant generates a new event (either regular or triggered), the secondary controller will go to the standby mode and only the primary controller will be the engaged one.

Finally, we discuss the benefits of our fault tolerance approach in terms of lower number of messages and hence bandwidth savings and better network resource utilization. To asses these savings, the ratio between the number of messages required using our approach and the periodic one will go to 1/n when time goes to infinity, where n is the number of sampling periods that should pass until a regular event is generated. So, for example, if the regular events are generated every 10 sampling periods, then this ratio goes to 10%. This means that when time goes to infinity, the savings in the bandwidth go to 90% compared with the traditional periodic case.

VI. Evaluation using LabVIEW and Amazon Cloud

In this section, we evaluate the proposed event-based cloud control approach using LabVIEW and Amazon cloud and show how a simulated system performs using these approaches in the cases of having network delays and failures.

For the plant, we consider the speed control system of a DC motor [29]. This system is chosen because it can be found in many industrial automation plants. The motor is a bidirectional brushless DC motor. Its rotation frequency can reach up to 27,000 RPM. An analog servo drive is used to control the motor. A tachometer is used to measure the motor's speed. The FOPDT model of this system is obtained by using the area method [30]. The transfer function of this system is approximated to the FOPDT process shown in:

$$P(s) = (\frac{0.526}{0.5402s + 1})e^{-2s}. \quad (3)$$

The DC motor system is emulated using LabVIEW and deployed on a machine in our lab in Vancouver, Canada. We use Amazon EC2 as our cloud provider. The PI primary controller is deployed on a virtual machine (VM) in Singapore (more than 8,000 miles away from the plant). The secondary controller is deployed on another VM in Sao Paulo, Brazil (more than 6,000 miles away from the plant). The communication between the plant and controllers is done using Modbus over TCP protocol. The sampling period of the simulated plant is equal to 300 milliseconds. If 10 sampling periods pass without generating triggered events, then a regular event will be generated.

To evaluate the proposed fault tolerance method, we feed the system with a step input at time t=0 s. At this time the

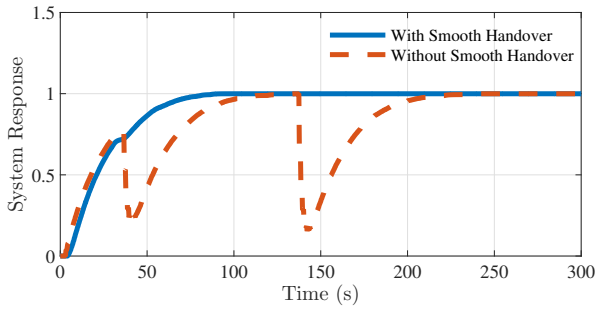


Fig. 4: The System response using the fault tolerance algorithm with the smooth handover enabled and disabled.

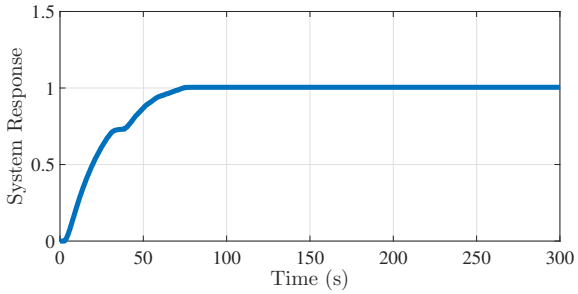


Fig. 5: The System response using the fault tolerance algorithm with smooth handover enabled while having three redundant controllers.

primary controller is engaged while the secondary controller is in standby mode. We then fail the primary controller by disconnecting it during the transient state of the system response at t= 30 s. After that, we restart the primary controller again after the system reaches the steady state at t= 135 s. The normalized system response during this process is as shown in the blue (solid) curve in Fig. 4. We can see that despite the failure of the primary controller, the system response acts as if there were no failure.

To show the importance of the smooth handover method, we repeat the above experiment after disabling this method. The red (dashed) curve in Fig. 4 shows that at the time of handover between the two controllers there is a bump in the response. These bumps are highly undesirable in practice. Furthermore, enabling the smooth handover method reduces the number of generated events. In our results, the number of generated events without the smooth handover method (166 events on average) is about 42% more than the number of generated events when using it (117 events on average). This shows another reason why smooth handover is critical in the context of event-based cloud control.

Another note here is the savings achieved in the event-based case compared to the periodic one. So, during 1000 sampling periods and when disabling the smooth handover functionality, our event-based approach needs only 16.6% on average of the messages in the periodic case even with the additional events required to overcome the bumps in the response. When the smooth handover is enabled, our approach only needs 11.7% on average of the messages in the periodic case, meaning that our approach saves more than 88% of the bandwidth in this case.

To show the extensibility of the fault tolerance approach, we test the case of having three redundant controllers; a primary controller in a VM in Oregon, the United States (about 500 miles away from the plant in Vancouver, Canada), a secondary controller in a VM in Singapore and a tertiary controller in a VM in Sau Paulo, Brazil. We start with having the primary controller as the engaged controller while the others are in the standby mode. We then fail the primary and secondary controllers at time t= 30 s. The tertiary controller takes the lead. We then restart the secondary controller at t= 90 s and then the primary controller at t= 135 s. Fig. 5

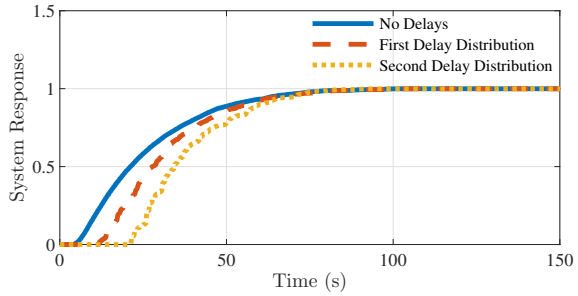


Fig. 6: System response under three delay distributions.

shows that despite those failure and recovery processes, the plant response achieves almost smooth set point tracking.

To stress-test our approach of handling network delays under the same testing setup described above, we use Microsoft's Network Emulator For Windows Toolkit [31] to add additional emulated delays to the existing real network delays. The added delays follow the normal distributions which are characterized by the mean μ and standard deviation σ. We use two delay normal distributions, the first one has μ= 1 s and σ= 0.7 s, and the second one has μ= 2 s and σ= 1.4 s. The added delays include both channels from the controller to the plant and vice versa. Moreover, it is important to note that with large delay values, packets can be received out of order at the destination or even lost. In our solution, since we use Modbus over TCP protocol, lost packets are resent and TCP ensures the re-ordering of out-of-order ones. However, a challenging situation happens because several packets can be transmitted at the same time to the plant. In this case, the plant executes only the last packet and discards the rest.

Fig. 6 shows the system responses in three cases. The blue (solid) curve shows the system response without adding emulated delays while the red (dashed) and orange (dotted) curves show the system response under the first and second delay normal distributions, respectively. The figure shows that the three curves are almost the same when it comes to steady state errors and overshoots. The difference between them is that the settling time increases with the increase of the added delay.

Table I compares the three responses based on overshoots, steady state error, settling time (that is, the time required

TABLE I: Performance measures of the DC Motor speed control system under different delay normal distributions.

Delay Normal Distribution (s)	Overshoots?	Settling time (s)	Steady State Error	Number of Events
No emulated delay	No	63.9 s	No	70
μ= 1 s and σ= 0.7 s	No	65.4 s	No	71
μ= 2 s and σ= 1.4 s	No	70.5 s	No	71

to stay within 5% of the final value) and the number of generated events (during 500 sampling periods) in each case. We can notice here that even under emulated network delays more than 10 times the sampling period of the plant, the system did not overshoot or deviate from its final value.

VII. Conclusion

In this paper, we introduced event-based control as a cloud service. Although previous works in the literature discussed this idea, none of them provided a way to deal with the large induced delay values and network failures that occur due to moving the event-based controllers to the cloud. We showed how the delay problem can be reduced to the problem of controlling a process with dead time. We presented our model of event-based cloud control systems which included an Internet delay estimator. We showed how our model solved the delay problem in such a way that guaranteed the system stability. We evaluated the proposed approach under a wide range of delay values and our results showed the robustness of the proposed approach. We also presented an approach to deal with network failures and packet losses. Our results showed the validity of our approach even if the cloud controllers were thousands of miles away from the controlled plant.

References

[1] K.-E. Årzén, "A simple event-based PID controller," in *Proc. of IFAC World Congress*, Beijing, P.R. China, January 1999, pp. 423–428.

[2] K. J. Åström and B. Bernhardsson, "Comparison of riemann and lebesque sampling for first order stochastic systems," in *Proc. of IEEE Conference on Decision and Control, (CDC'02)*, vol. 2, Las Vegas, NV, December 2002, pp. 2011–2016.

[3] J. Ploennigs, V. Vasyutynskyy, and K. Kabitzsch, "Comparative study of energy-efficient sampling approaches for wireless control networks," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 416–424, August 2010.

[4] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 398–409, April 2015.

[5] T. Hegazy and M. Hefeeda, "Industrial automation as a cloud service," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 10, pp. 2750–2763, October 2015.

[6] "Why Use LabVIEW?" http://www.ni.com/white-paper/8536/en, March 2009.

[7] O. Givehchi, J. Imtiaz, H. Trsek, and J. Jasperneite, "Control-as-a-service from the cloud: A case study for using virtualized plcs," in *Proc. of IEEE Workshop on Factory Communication Systems (WFCS'14)*, Toulouse, France, May 2014, pp. 1–4.

[8] T. Goldschmidt, M. Murugaiah, C. Sonntag, B. Schlich, S. Biallas, and P. Weber, "Cloud-based control: A multi-tenant, horizontally scalable soft-plc," in *Proc. of IEEE International Conference on Cloud Computing (CLOUD'15)*, New York City, NY, June 2015, pp. 909–916.

[9] A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J. L. Martínez Lastra, *Industrial Cloud-Based Cyber-Physical Systems*. Springer, 2014.

[10] A. Vick, V. Vonasek, R. Penicka, and J. Kruger, "Robot control as a service: Towards cloud-based motion planning and control for industrial robots," in *Proc. of IEEE International Workshop on Robot Motion and Control (RoMoCo'15)*, Poznan, Poland, July 2015, pp. 33–39.

[11] A. Vick, C. Horn, M. Rudorfer, and J. Kruger, "Control of robots and machine tools with an extended factory cloud," in *Proc. of IEEE World Conference on Factory Communication Systems (WFCS'15)*, Palma de Mallorca, Spain, May 2015, pp. 1–4.

[12] A. Didic and P. Nikolaidis, "Real-time control in industrial IoT," Master's thesis, Malardalen University, Sweden, 2015.

[13] D. Lehmann and J. Lunze, "Event-based control with communication delays and packet losses," *International Journal of Control*, vol. 85, no. 5, pp. 563–577, 2012.

[14] F. Forni, S. Galeani, D. Nešić, and L. Zaccarian, "Event-triggered transmission for linear control over communication channels," *Automatica*, vol. 50, no. 2, pp. 490–498, 2014.

[15] X.-M. Zhang and Q.-L. Han, "Event-based dynamic output feedback control for networked control systems," in *Proc. of American Control Conference, (ACC'13)*, Washington, DC, June 2013, pp. 3008–3013.

[16] H. Yu and P. J. Antsaklis, "Event-triggered output feedback control for networked control systems using passivity: Achieving l2 stability in the presence of communication delays and signal quantization," *Automatica*, vol. 49, no. 1, pp. 30–38, 2013.

[17] V. Dolk and W. Heemels, "Dynamic event-triggered control under packet losses: The case with acknowledgements," in *Proc. of International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP'15)*, Krakow, Poland, June 2015, pp. 1–7.

[18] A. Cetinkaya, H. Ishii, and T. Hayakawa, "Event-triggered output feedback control resilient against jamming attacks and random packet losses," *Proc of. IFAC-PapersOnLine*, vol. 48, no. 22, pp. 270–275, September 2015.

[19] V. Dolk, P. Tesi, C. De Persis, and W. Heemels, "Output-based event-triggered control systems under denial-of-service attacks," in *Proc. of IEEE Conference on Decision and Control (CDC'15)*, Osaka, Japan, December 2015, pp. 4824–4829.

[20] M. Beschi, S. Dormido, J. Sanchez, and A. Visioli, "Characterization of symmetric send-on-delta PI controllers," *Journal of Process Control*, vol. 22, no. 10, pp. 1930–1945, December 2012.

[21] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. Imperial College Press, 2009, vol. 57.

[22] K. J. Åström and T. Hägglund, "The future of PID control," *Control engineering practice*, vol. 9, no. 11, pp. 1163–1175, November 2001.

[23] ——, *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[24] V. Vasyutynskyy and K. Kabitzsch, "Implementation of pid controller with send-on-delta sampling," in *Proc. of International Conference Control*, Glasgow, Scotland, UK, August 2006.

[25] "Introduction to Modbus TCP/IP (white paper)," http://www.acromag.com/sites/default/files/Acromag_Intro_ModbusTCP_765A.pdf, 2005.

[26] E. Fridman, *Introduction to time-delay systems: Analysis and control*. Springer, 2014.

[27] J. D. Bendtsen, J. Stoustrup, and K. Trangbæk, "Bumpless transfer between advanced controllers with applications to power plant control," in *Proc. of IEEE Conference on Decision and Control (CDC'03)*, vol. 3, Maui, Hawaii, December 2003, pp. 2059–2064.

[28] H. Wade, *Basic and Advanced Regulatory Control: System Design and Application*, 2nd ed. Research Triangle Park, North Carolina, USA: ISA, 2004.

[29] Á. Ruiz, J. E. Jiménez, J. Sánchez, and S. Dormido, "Design of event-based PI-P controllers using interactive tools," *Control Engineering Practice*, vol. 32, pp. 183–202, November 2014.

[30] A. Visioli, *Practical PID control*. Springer Science & Business Media, 2006.

[31] "Network Emulator For Windows Toolkit," https://blog.mrpol.nl/2010/01/14/network-emulator-toolkit/.