Event-based Control as a Cloud Service

by

Alaa Eldin Abdelaal

B.Sc., Mansoura University, Egypt 2012

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Spring 2017

Approval

Name:	**Alaa Eldin Abdelaal**
Degree:	**Master of Science**
Title:	***Event-based Control as a Cloud Service***
Examining Committee:	**Chair:** Dr. Ramesh Krishnamurti Professor

Dr. Mohamed Hefeeda
Senior Supervisor
Professor _____

Dr. Tamir Hegazy
Supervisor
Independent Scholar _____

Dr. Richard Vaughan
Supervisor
Associate Professor _____

Dr. Nick Sumner
Internal Examiner
Assistant Professor _____

Date Defended: 19th, April, 2017

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 thesis, 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 mitigating long and variable network delays and handling controller failures introduced because of moving the controller far away from the plant. We propose novel methods to solve the delay and failure problems and we show that these methods maintain the stability and performance of the control system. We implemented the proposed methods and deployed them on the Amazon cloud. Our results 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, the proposed fault tolerance approach can transparently handle controller failures even if the controlled system is thousands of miles away from its cloud controllers.

Keywords: Cloud computing; industrial automation; delay compensation; fault tolerance; event-based control

Dedication

To my beloved parents.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my senior supervisor, Dr. Mohamed Hefeeda, for mentoring me in the past two years with his patience and immense knowledge. I would also like to express my deepest thanks to Dr. Tamir Hegazy for his time and support. I am very grateful for all I have learned from him and for his continuous help in all stages of this thesis. Also, I would like to express my gratitude to Dr. Richard Vaughan, my supervisor, and Dr. Nick Sumner, my thesis examiner, for being on my committee and reviewing this thesis. I also would like to thank Dr. Ramesh Krishnamurti for taking the time to chair my thesis defense.

I would like to thank all my colleagues at the Network Systems Lab for their support and help. It was my honor to work with these talented people. I would also like to thank my friends for their encouragement and constant support.

I am indebted to my family for their love, encouragement, and endless support. I owe my deepest gratitude to my parents. I owe my loving thanks to my wife, Maram. Without their encouragement and understanding, it would be impossible for me to finish this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

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. By using the network as one of the components in modern control systems, event-based control [5], [7], [6], a relatively new approach of control, emerges with the goal of efficiently utilizing the network resources, and consequently saving energy in networked control systems (NCS). 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. Event-based control can be further classified into two main types: event-triggered control and self-triggered control [30]. In the former, the state or the output of the system is continuously monitored in order to check whether the event-triggering condition is satisfied and consequently performing the control action, while in the latter the system calculates the next update/event time as a function of its state at the current time.

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.* [45] show that this approach can save around 80% of the energy of devices in the context of building automation and control systems. This is comparable to what Pawlowski *et al.* [44] report after applying the event-based approach to the greenhouse climate control problem where the goal is to regulate the temperature to improve the growth of crops in greenhouses. Moreover, in the context of wireless sensor networks in industrial settings, Araujo *et al.* [4] show that event-based control techniques can save up to 85% of the communication messages and about 54% of the of battery lifetime of the sensors.

On the other hand, cloud computing has the potential to improve current automation and control systems. In a recent survey, Kehoe *et al.* [33] highlight the increasing interest in

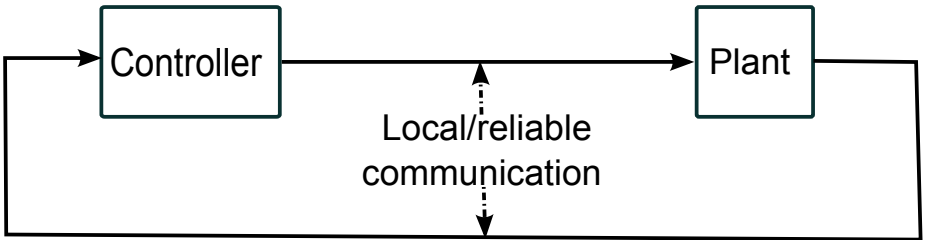


Figure 1.1: Current feedback control structure.

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 [32] 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.

In this thesis, we propose providing event-based control as a service from the cloud. Our motivation is not only to better utilize the 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 some challenges to overcome. These challenges include dealing with possible delays, failures and security related issues 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 the performance of the event-based control system. Addressing security challenges is outside the scope of this thesis.

1.2 Problem Statement and Thesis Contributions

Offering the control functionality as a cloud service has its challenges. These challenges include network delays, failures and packet losses. Although these challenges may seem the same as the ones in the general area of networked control systems, our case is more complex.

In current feedback control systems as shown in Fig. 1.1, a controller is placed within a small physical distance from the controlled system. Further, a controller communicates with the controlled system over a dedicated local area network (LAN). Such setup provides small, mostly deterministic communication delays, which are often negligibly small compared to the sampling period. By providing the control functionality as a cloud service as shown in Fig. 1.2, however, communication between the system being controlled and its controllers may have to go over the wide area network (likely the Internet). This will result in variable delays or even loss of packets carrying control actions and sensing values. In other words, larger variable delays added to the control loop could even break the control loop altogether upon Internet link and/or cloud failures.

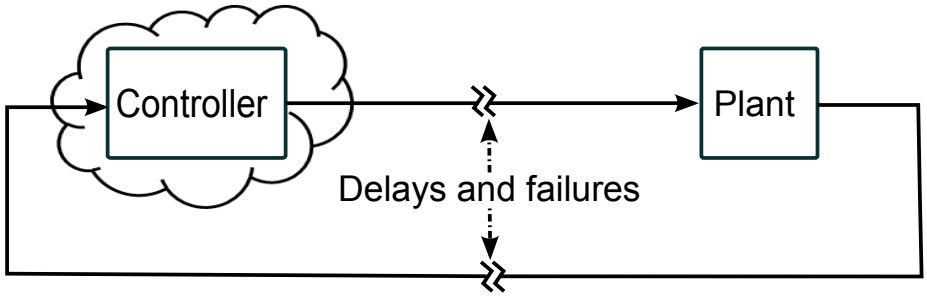


Figure 1.2: Cloud control structure.

Handling failures is also more challenging in our case. Redundant controllers are used to overcome failures in current automation systems. These controllers are co-located and have access to the same memory and variables. This facilitates the handover from one controller to another in case of failures. Redundant controllers in our case are deployed on different virtual machines in different locations in the world. This complicates the handover process between controllers in case of failures.

Based on the above, we can state our research question in this thesis as follows:

How can we compensate for the effects of network delays, packet losses and failures on output-feedback event-triggered cloud control systems?

To answer this question, this thesis makes the following contributions (some of which are published in [1]):

1. We propose a delay compensation technique to handle network delays. Our technique employs a delay estimator which 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.

2. 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.

3. We implement the proposed techniques in an industry-standard system design software, LabVIEW [35], and we deploy it on different locations of the Amazon cloud. We show that the proposed cloud event-based control service can effectively control a distant plant even under variable Internet delays, and packet losses. Moreover, we analyzed the proposed techniques and we show that our approach is no worse than the periodic approach from the network utilization point of view. In the best case scenario using our approach, the system does not need to generate any events during the steady state part of its response. Under our experimental setup, our results show that 95% of the communication messages between the plant and the controller can be saved with no noticeable degradation in the control system performance in a limited

time period. Increasing this period results in even more savings while maintaining the same control performance.

1.3 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents a background on event-based control and networked control systems. It also surveys the related works in the literature. Chapter 3 presents the details of the proposed event-based control cloud service and our proposed techniques to handle network delays and failures. Chapter 4 presents our evaluation setup and results. Chapter 5 concludes the thesis.

Chapter 2

Background and Related Work

In this chapter, we start by providing a brief background on the general area of networked control systems as well as event-based control systems along with some of their applications. We then present the related works in the literature.

2.1 Networked Control Systems

In control theory, a networked-control system (NCS) is a feedback control system in which the control loops are closed by a communication network [28]. A typical networked control system consists of a plant or a controlled system, controller(s) and sensor(s). Two or more of these components exchange signals using a network. Compared with traditional control systems, NCS has several advantages. These include the increased flexibility of the system, the elimination of unnecessary wiring (especially in wireless NCS), and the relatively lower cost. These advantages among others make NCS a better choice in many applications like industrial automation, tele-operation, transportation networks and electrical power grids.

On the other hand, to realize the full potential of NCS, some challenges need to be overcome. These challenges include network-induced delays, packet losses and network security [61], [65]. Network-induced delays include sensor-to-controller delays and controller-to-plant delays. These delays in practice are random and variable and can lead to degradation of the system performance or even instability of the system altogether. In addition, several factors such as network congestion and interference can hinder a sent packet from reaching its destination which leads to packet losses. This problem can be considered as the more general case of the delay problem at which the delay value is infinite. Such a problem makes one or more component of a NCS unaware of the others which can cause some unpredictable behaviors in the system. Moreover, network security is a major concern in NCS especially when using wireless communication. Transmitted data across the system components may be exposed to malicious attacks such as Denial of Service (DoS) attacks.

Most of the research in the area of networked control systems is devoted to solving these problems.

In networked control systems, transmitted signals are sampled in a time-based or event-based fashion. Consequently, networked control systems are classified into time-based systems and event-based ones. Time-based control systems are the ones used in the earlier studies in this area. In these systems, the signal transmission occurs every sampling period regardless of the current state of the system. This traditional approach has the advantage of being relatively simpler in design and analysis. However, this periodic approach does not take into consideration the utilization of the network and computational resources. These two factors are important in control systems where sensors and actuators are battery-powered and/or operating on limited bandwidth. To meet these needs, event-based control systems were proposed. In this case, it is the system need rather than the time passed that is used to determine when the signals should be transmitted. For example, a control signal can be sent from the controller to the plant only if the error at the plant side is greater than some value.

Although there are many researchers working in the area of event-based control, the theoretical foundation of this area is not solid compared with the periodic control approach [30], [39]. So, as a road map of research in this area, it is important to develop a mature theory, validate it practically and provide a way to decide when it is better to use each of these two control approaches.

In the next section, we give a brief background on event-based control systems and some of its applications.

2.2 Event-based Control

An event-based control loop consists of a plant, controller and sensor(s). Another component is also needed, which is the event generator. This additional component is responsible for deciding when a given signal in the system should be transmitted based on a sampling algorithm. In practice, this event generator is implemented on smart sensors and actuators or it is a part of the control logic itself in the controller. In the former case the sampled signal is the feedback signal from the plant to the controller and in the latter the control signal itself is sampled.

2.2.1 Event-based Control Types

Research in the area of event-based control is conducted in two main directions, constituting the two types of the event-based control approach:

1. Event-triggered control:
   In this type, the state or the output of the system is continuously monitored in order

to check whether the event triggering condition is violated or not and consequently performing the control action. According to what is monitored in the system, this type can be further classified into:

(a) State-feedback Event-triggered Control: The state of a system refers to the set of variables that enable us to predict the behavior of the system in the future. In state-feedback control systems, these variables are measured and used to calculate the control actions. The drawback of this approach is that full state measurements are not available in practice [10].

(b) Output-feedback Event-triggered Control: The output of a control system is the variable that is being controlled e.g., the liquid level in a tank. This variable can be easily measured. That is why output-feedback control is a more preferred option in practice [10].

2. Self-triggered Control: In this type, there is no need to continuously monitor the system state/output. Instead, the system calculates the next update/event time as a function of its state at the current time based on the plant dynamics [52].

The application domains of event-based control include wireless networked control systems and the distributed control of multi agent systems. A survey in this area can be found in [38]. Also, [30] provides a good introduction to the field.

2.2.2 Applications of Event-based Control

In this subsection, we introduce some applications of event-based control from the literature.

In [3], the authors present an event-triggered controller and they test their controller along with the traditional periodic controller on a physical system. They compare the performance of the two controllers with and without introducing network delay and packet loss. For the event-triggered controller, they define mathematically the triggering condition to happen when the error signal exceeds some value depending on the state of the system. The controller is a state-feedback and it is implemented using a microcontroller unit embedded on what they call the Event Generation Circuit (EGC). It is a low cost circuit that performs four main functions:

1. Compute the control signal
2. Decode the sensor signals
3. Check if the event condition is satisfied
4. Estimate the states of the plant

The plant itself is a 3D crane in a lab size. The Event Generation Circuit (EGC) is the controller whose signal is transferred using the EGC wireless node. This signal is then received by the actuation wireless node and transferred to the crane actuators/motors using a CPU. The encoders on the crane are the sensors that send the feedback signals to the Event Generation Circuit.

From a control point of view, it is an 8th order multi-input multi-output (MIMO) nonlinear plant. To control it, they divide the plant into two subsystems and design a controller for each one. They also add to their setup 10 inference wireless nodes so that they can increase the traffic over the network and measure the performance of the controllers in such a case. According to their evaluation, the event-triggered controller behaves in a similar way as the traditional periodic controller. In addition, the former has the advantage of achieving better utilization of the network as well as the increasing the battery life of the sensors and actuators.

Event-based control can be applied in robotics as well. In [46], the authors use a remote event-triggered controller for trajectory tracking task of a unicycle mobile robot. The main motivation is to reduce the dependency on the network and hence minimize the effect of packet losses and network delays. The controller communicates with the robot via a wireless network. They use a camera as their sensor. The camera sends its signal to the controller using a dedicated wire. The robot mathematical model that they use is a nonlinear one. They compare the performance (in terms of the tracking error) of the event-triggered controller with the periodic controller and they conclude that the performance is almost the same.

Another application is on robots coordination as in [49]. The authors use a self-triggered control approach to remotely regulate the linear and angular velocity of four robots. They use a wireless network for the communication between the robots and the remote controller. They use a linear time invariant model for the robots. The basic idea here is designing an adaptive controller according to the current network load. This is done by choosing the value of some control parameters according to not only the error in the state of the system but also to the current load of the network. The simulation results they provide show that this approach is slightly lower in performance than the traditional periodic control approach. However, the adaptation property of their proposed controller makes it better in network utilization than other self-triggered controllers that are designed by assuming a maximum value of the network delay.

Process control can also be provided in an event-based approach as in [36] where the authors apply the event-triggered control on a thermofluid plant using an event-based version of the classical PI controller. Again, they use a nonlinear model of the plant for the control system design. The event generator in their case is part of the sensor and not the controller. The main contribution here is that they prove that the event-based approach is robust to bounded disturbances.

So, from the above examples, it is now clear that event-based control can be applied in a wide variety of applications.

2.3 Related Work

Several cloud services have emerged to take advantage of the cloud computing model promises of cost saving, flexibility and agility in the area of robotics and automation. For example, Kehoe *et al.* [33] propose Robotics and Automation as a Service (RAaaS). They envision a scenario when a user can enter his robot model information to a website and then chooses his/her desired algorithm to perform the required task, like grasping or motion planning. The chosen algorithm runs in the cloud and the results are sent to the robot for execution. Kumar *et al.* [34] present "Carcel" which is a cloud-based system for assisting autonomous vehicles. This allows the vehicle to plan its path based on not only its own sensors but also on remote sensors in the road infrastructure. Moreover, Mohanaraja *et al.* [40] propose a Platform as a Service (PaaS) framework, called Rapyuta, tailored for robotics applications. Rapyuta allows robots to move heavy computations to the cloud. It also enables sharing experiences between robots by providing access to the RoboEarth knowledge repository, which is a World-Wide Web for robots [50].

In the area of industrial automation, Industrial Internet [21] is a term introduced by General Electric where Internet-based technologies (including cloud computing) are applied to industrial applications in different aspects. In Germany, a similar initiative is introduced under the term "Industry 4.0" predicting a fourth industrial revolution that will make use of networking and Internet of Things (IoT) to improve the state of current industrial automation [25]. Potential applications that can benefit from this new paradigm include transportation, power production, oil and gas industries and healthcare. A related concept in this context is cloud manufacturing [59], which refers to paradigm in manufacturing that is service-oriented and knowledge-based with the aim to achieve higher efficiency while consuming lower energy. A recent survey on this area can be found in [29].

All of the above works focus on improving high level tasks like planning and optimization with the help of cloud computing. In this thesis, we are concerned with the low level task, that is the control functionality itself and how it can benefit from cloud computing.

One of the early works that considers providing the control functionality as a cloud service is [26]. 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 the input signal. They compare the results of this setting with the ones obtained using a local PLC doing the same task. The comparison shows that the cloud-based PLCs yield almost the same performance for sampling periods greater than 128 ms. Thus, they conclude that this solution is promising for soft real-time applications.

A similar study is presented in [27]. The authors propose an architecture for providing control as a service. They test this architecture on a simple discrete process from the building automation domain. They also discuss some scenarios in which the cloud-based control can be beneficial. According to their proposed architecture, the control programming can be done via a web-based Integrated Development Environment (IDE). These programs are then sent to the soft-PLC component where they are executed in an event-based approach. However, this paper does not discuss any strategy to mitigate the delays or failures that may occur.

In [17], 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. In their proposed architecture, the low-level control functionality originally done using PLC is performed using a service bus that communicates wirelessly with the field devices, i.e., sensors and actuators. They conclude that the system performance is comparable to a legacy on-site PLC.

Vick *et al.* [54] 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 [53].

Didic *et al.* [18] 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. They test their mechanism on a hard real-time process, that is controlling/regulating the brightness of a LED using an adaptive PI controller. The results show that having a network delay greater than the process sampling period leads to a degradation of the proposed system performance.

Another delay compensation technique is presented in [32]. 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 address the delay problem in the context of event-based control like [37], [4], [63], [22] and [24] to name a few, have a main drawback. Those works employ state-feedback approach assuming that the full state information of the system is available. However, in practice this assumption does not hold which hinders the applicability of these approaches. As a result, a practical solution to consider is using output-feedback event-based control. In this context, we only encounter two works [64] and [62] that address the delay problem. Even in those works, some conservative assumptions are used that make the proposed solution inapplicable in the case of moving the controllers to the cloud. In [64], the authors assume that there are no packet losses or packet reordering during the data transmission over the

Table 2.1: Summary of the related work.

Prior Work	Hard/Soft Real-time	Delay Compensation	Fault tolerance	Example Application	Control Approach
[27]	Soft	No	No	Building Automation	Event-based Control
[17]	Soft	No	No	Lubrication System	Event-based Control
[26]	Soft	No	No	Inverting a Digital Input Signal	Periodic Control
[54], [53]	Soft	No	No	Industrial Robots	Periodic Control
[18], [42]	Hard	Yes	No	LED Brightness Regulation	Periodic Control
[32]	Soft	Yes	Yes	Solar Power Plant	Periodic Control
This work	Soft	Yes	Yes	DC Motor Speed Control	Event-based Control

network. In reality, with a high value of the delay, there will be packet losses. In the other work [62], the proposed method requires a local controller at the plant side and in our work we propose moving 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 [37] and [19]. As far as we are concerned, only two works [16] and [20] address this problem in the output-feedback case. In [16], 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 [20]. The authors target a class of denial of service attacks (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 very limited. Most of the previous works can be considered as feasibility/pilot studies to explore this area. These studies are compared together in Table 2.1. Moreover, for event-based control, we are not aware of any works that address the challenges of moving the controllers to the cloud, especially handling the delays introduced by the Internet and the control/link failures.

Chapter 3

Proposed Architecture

In this chapter, we present our approach for realizing output-feedback event-based control as a service from the cloud. We start with a brief introduction of the considered event-based control structure. Then, we discuss the benefits of moving the controllers to the cloud. After that, we present our proposed techniques to handle network delays and failures.

3.1 Event-based Control Structure

The considered event-based control system structure [13] is shown in Fig. 3.1. Solid lines in Fig. 3.1 represent time-based communications and dashed lines represent event-based ones. Time-based communications refer to communications that are performed periodically. Event-based communications refer to communications that are done only when needed based on an event-triggering condition. The structure consists of a controller, a plant and an event generator. The controller is assumed to be a PI controller, which is the most common type of controllers in the industry [43], [8]. The plant is assumed to be be a First Order Process with Dead Time (FOPDT) whose general transfer function is shown in Eq. (3.1), where K is the process gain, T is the process time constant and L is the process delay (dead time).

$$P(s) = (\frac{K}{Ts+1})e^{-Ls}. \tag{3.1}$$

Many higher order processes can be easily approximated to an FOPDT process. Moreover, in practice one does not have to have the analytical model of the higher order system to get the equivalent FOPDT model. We can get this model from the step response curve of the original plant [9].

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

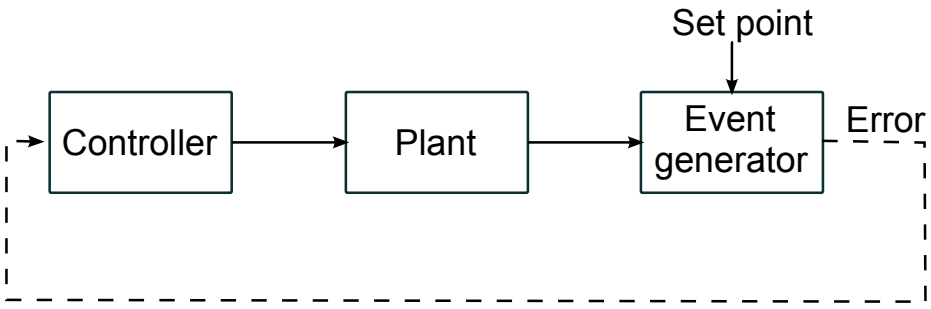


Figure 3.1: The structure of the event-based control system.

(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 called 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. By rounding the error value to the nearest multiple of Δ, there is always a state where the sent error value from the event generator to the controller is equal to zero. This allows the system to eliminate limit cycles in its response [13].

The main reason of considering the above event-based control structure is that in [13] the authors provide the sufficient conditions of the controller parameters that make the system stable without having limit cycles. We note that other structures exist, such as [47]. However, this structure requires a local controller at the plant side, which makes it impractical for our proposed cloud service.

3.2 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 [5] and [51]. Some 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 [2]. Similar to HTTP, Modbus is an application-level protocol used to send commands to read/write various registers in the I/O device.

3.3 Moving Controllers to the Cloud: Motivation

Moving the controllers to the cloud can lead to significant benefits. According to the cost analysis performed in [32], large-scale automation systems like the case in oil and gas industry can achieve significant cost savings by adopting the cloud control paradigm compared with the current one. Those savings come from the following:

- For the upfront cost, hardware devices like physical controllers and control cabinets are not needed. Those are replaced by virtual machines in the cloud.

- Labor costs are reduced. Moving the controllers to the cloud reduces significantly the number of site visits required by engineers. According to [32], the hourly rate of an engineer on site is 3 times more than his/her rate in office.

- Although there are additional costs required to lease the virtual machines, the above study shows that these costs are overcome by the savings achieved in the maintenance costs.

In addition to the cost savings, moving the control functionality to the cloud can significantly reduce the time required for automation plants to start working. This is studied in [31] and its results are summarized as follows:

- Getting rid of many hardware components, e.g., controllers, cables and control cabinets, reduces the time required for hardware configuration. This includes the time to assemble and wire hardware devices together. This also reduces the time associated with the testing phase of any automation plant. By adopting cloud controllers, testing can be performed much faster without the need to deal with any sort of electrical signals.

- The time needed for shipping different hardware devices from engineering locations to the location of the automation system is greatly reduced because we do not need most of these devices using our cloud control services.

- The time required for the software development is expected to reduce. The reason is that by having the control logic itself in virtual machines in the cloud, the time required to duplicate this logic into several hundreds or even thousands of machines is shorter compared with the case of having hardware controllers.

Furthermore, several benefits can be realized from the agility associated with the cloud computing model. First, automation systems that use a cloud control service do not need to stick with one automation provider anymore. Instead, cloud controllers can be deployed in different data centers for different cloud providers which gives more flexibility to the owners

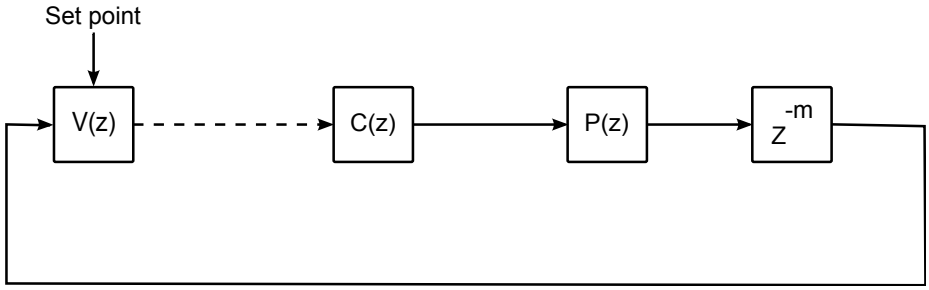
of automation systems to choose from a wider range of options. Second, the complexity of the automation systems deployment is significantly reduced because of the elimination of most of the wiring needed in the system. Third, instead of duplicating different hardware components as a backup for the main ones, cloud controllers can be an easier and cheaper option to do so. This can be particularly beneficial at the time of maintenance of the physical controllers.

Many application domains can benefit from our proposed service. For example, large automation systems, e.g., oil and gas plants, contain tens of thousands of sensors and actuators. Developing an automation system for such large-scale plants is costly and time-consuming. Internet of Things (IoT) applications can also benefit from our model. Possible examples include building automation applications that are based on low-power, low-range, limited-bandwidth technologies (e.g., ZigBee) [27]. The same applies to robotics. Applications that require hundreds of robots, like the case in [60], can use our model to ensure better resource utilization, lower cost and higher agility. Another interesting area of application is online optimization control techniques [58]. A possible example is the model predictive control (MPC) [15] at which the control system starts with a model of the environment and optimize the model parameters online. Some computationally intensive tasks can be used for this purpose like machine learning. Such tasks are tightly coupled with the control functionality and using our proposed system, they can be provided as a service from the cloud. In this case, the resulting system can also benefit from the computational power of the cloud in addition to saving cost and time.

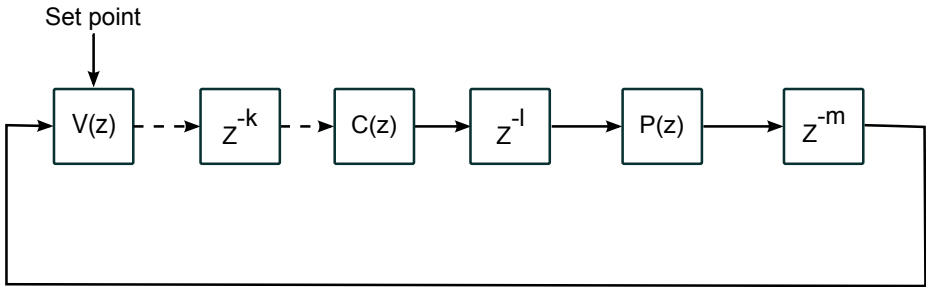
3.4 Proposed Delay Compensation for Event-based Cloud Control

We propose 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 [13] 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 [23]. Therefore, an Internet delay estimator is used to feed the system model with these delays.

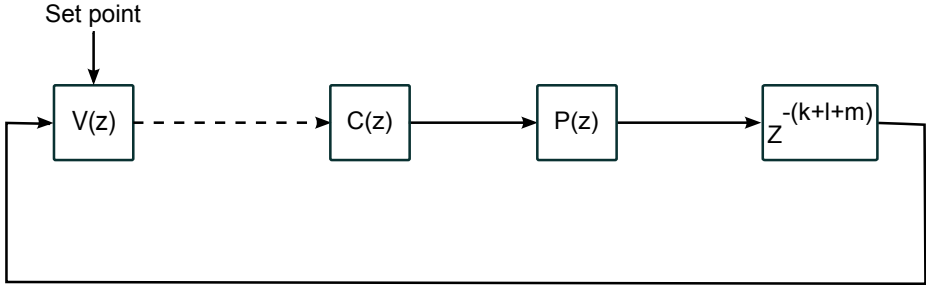
We start with the general structure of event-based control as shown in Fig. 3.1. This structure can be rearranged and put in the Z domain as in Fig. 3.2(a). Here, the plant block in Fig. 3.1 is divided into two blocks; $P(z)$ representing the first term of the FOPDT model in eq. 3.1 and Z^{-m} representing 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.



(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 [13].

Figure 3.2: The proposed approach to solve the delay problem.

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. 3.2(b), where k and l represent the values of the delays. We can alter the arrangements of the blocks in the feed-forward direction and we can see in Fig. 3.2(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 [13]. 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 conserving the other performance metrics of the original system response.

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. 3.3.

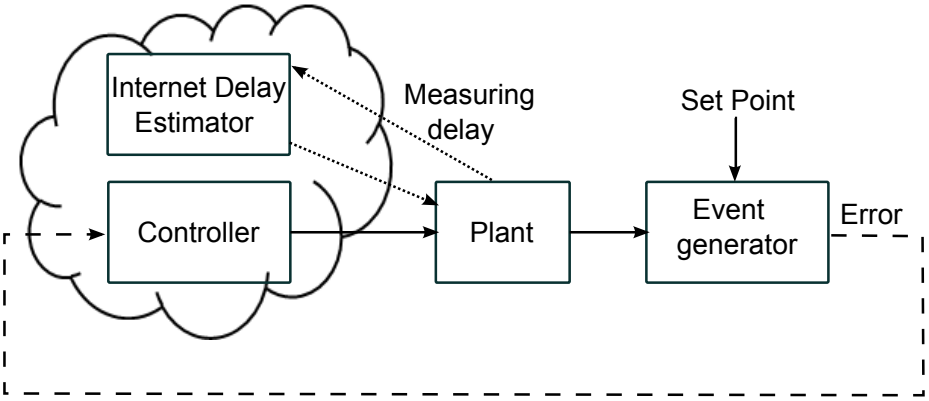


Figure 3.3: The proposed model after adding the Internet delay estimator.

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 Eq. (3.2)

$$D_c = \frac{D_i + hV_i^{0.5}}{T_s}, \tag{3.2}$$

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. Similar delay estimator is used in [32].

3.5 Proposed Fault Tolerance for Event-based Cloud Control

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.

3.5.1 The Reliable Cloud Controller (RCC) Algorithm

The proposed solution for handling failures is based on the Reliable Cloud Controller (RCC) algorithm proposed in [32], which is designed for periodic control. We extend it to support event-based control systems. The RCC algorithm runs every sampling period and it consists

of 4 steps. The first one is the *initialization step* where each controller is assigned an ID. The second is the *polling step* where the controllers receive the plant information from the I/O interface. This information includes the process variable (the plant output) and the value of the last control action executed by the plant. In this step, if the controller does not receive anything from the plant, then the algorithm does not continue to the next steps and it goes back again to the first one. The third step is the *computing step* where each controller computes its control signal based on the received information. 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.

3.5.2 Smooth Controller Handover

The handover from a failed controller to a newly engaged one can cause a bump in the system response. This bump occurs due to the difference between the output values of the newly engaged controller and the failed one. Since we use a PI controller in our event-based control structure, this bump can occur because the integrator parts of the controllers may have different starting times for their integration intervals. The RCC algorithm employs the bumpless transfer technique from control theory [12] to achieve smooth handover between controllers. The key idea of this technique is to adjust the integrator's initial value [56] for the engaged controller before sending its first control signal to the plant. This adjustment is done by subtracting the contribution of the P component of the PI controller from the last control signal of the failed controller. The result of this subtraction is the integrator's initial value of the newly engaged controller.

3.5.3 RCC Theoretical Guarantees

The authors in [32] provide *three theoretical guarantees* on the system response using their RCC algorithm. The first one is that if the current controller with the smallest ID fails then the healthy controller with the second smallest ID will take the lead automatically. The second guarantee is that if a controller, with smaller ID than the current engaged controller, recovers, then it becomes engaged and the current controller returns to the standby mode. The third guarantee is that the handover process between the controllers is done in a smooth

way without affecting the system response. We will show shortly how these three guarantees are extended to the case of having event-based controllers.

3.5.4 Addressing the Packet Loss Problem Using RCC

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) and it does not perform the control logic. As a result, it will not send anything to the plant. The plant in this case will increase the last action age of the primary controller because it did not receive any control signal. 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).

3.5.5 RCC Algorithm for Event-based Control

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. That is why a modification is needed so that RCC algorithm can work well in the case of event-based control.

The basic idea of the proposed modification is to add another possible condition to generate a new kind of events, which we call "handover events", in addition to the generated events if the difference between the current error signal and the last one is greater than Δ. This new condition is true when a handover should occur between redundant controllers.

A handover occurs either from a controller with lower ID (failed primary controller) to a one with a higher ID (a secondary controller) or the other way around. In the first case, a handover event is generated if the engagement threshold of the controller with higher ID

is less than the last action ages of all the controllers with lower IDs. In the second case, a handover event is generated once a controller with lower ID recovers.

To explain how this is done, consider the case of having a primary controller, a secondary controller and the plant. The calculation of the last action age values for the two controllers is done at the plant side. In our method, the plant knows the engagement thresholds of the two controllers. If the plant does not receive a control signal from a controller, it increases its last action age value by one every sampling period. Now, when the primary controller fails, the plant increases its last action age value. When this last action age value exceeds the engagement threshold of the secondary controller (which the plant already knows), the event generation condition is true and the plant sends a message to the secondary controller. This message contains the plant output and the last action age value of the primary controller. Once this message reaches its destination, the secondary controller becomes engaged. A similar scenario occurs when the primary controller recovers. In such a case, the plant generates another handover event and communicates with both the secondary and primary controllers. This forces the former to return to the standby mode and the latter takes the lead again.

3.6 Analysis and Discussion

In the following, we analyze the proposed fault tolerance approach in more details.

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, consider the case when all other controllers whose IDs are less than the healthy controller's ID are non-reachable. So, there will be a moment when the engagement threshold of this healthy controller is less than the last action ages of all these controllers. This will force the plant to generate a handover event and hence the healthy controller will take the lead. Because of the smooth handover, this controller will continue working starting from the last control action received by the plant. This eventually guarantees the normal operation of the plant.

Second, 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 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 under one failure is: $(D_j+RTT_j)/T_s$ sampling periods where D_j is the engagement threshold of the newly engaged controller C_j, RTT_j is the round trip time between the plant and the controller C_j, 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 due to 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, our 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. In this case, again the plant is forced to generate a new handover event. This means that the plant will send a message containing the last action age value of the recovered controller (which is now zero) to the current controller. Thus, the recovered controller will become engaged and the current controller will go to the standby mode.

Finally, the benefit of our fault tolerance approach is that it requires lower number of messages and hence achieves bandwidth savings and better network resource utilization. Using our proposed method, the expected number of the generated events is the summation of two components. The first one is the expected number of events required to reach the steady state. The second one in our method is the number of handover events. In other words, the number of the handovers that happened during the lifetime of the system. This means that, in the best case scenario, if no handovers occur, then no additional events are generated. In such a case, the system needs only the triggered events required during the transient part of its response. Since the event triggering condition is checked every sampling period, our proposed approach is no worse than the periodic approach. This is true because in the worst case scenario, a new event is generated every sampling period and this is what happens in the traditional periodic case.

Chapter 4

Evaluation

In this chapter, we evaluate the proposed event-based cloud control structure.

4.1 Experimental Setup

Example Plant. We consider the speed control system of a DC motor [48]. 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 [55]. This system is chosen because it can be found in many industrial automation plants. The transfer function of this system is approximated to the FOPDT process shown in Eq. (4.1).

$$P(s) = (\frac{0.526}{0.5402s + 1})e^{-2s}. \tag{4.1}$$

The DC motor system is emulated using LabVIEW and deployed on a machine in our lab in Vancouver, Canada. LabVIEW, or Laboratory Virtual Instrument Engineering Workbench, is an integrated development environment produced by National Instruments Corporation. It is used for control and emulation purposes in both academia and industry [35]. In these applications, the user interface is called the front panel and the code itself is written in another entity called the block diagram. The execution order of the code in LabVIEW is determined by the flow of the data between different nodes in the block diagram.

Cloud Controllers. 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 locations of the plant and cloud controllers is shown in Fig. 4.1.

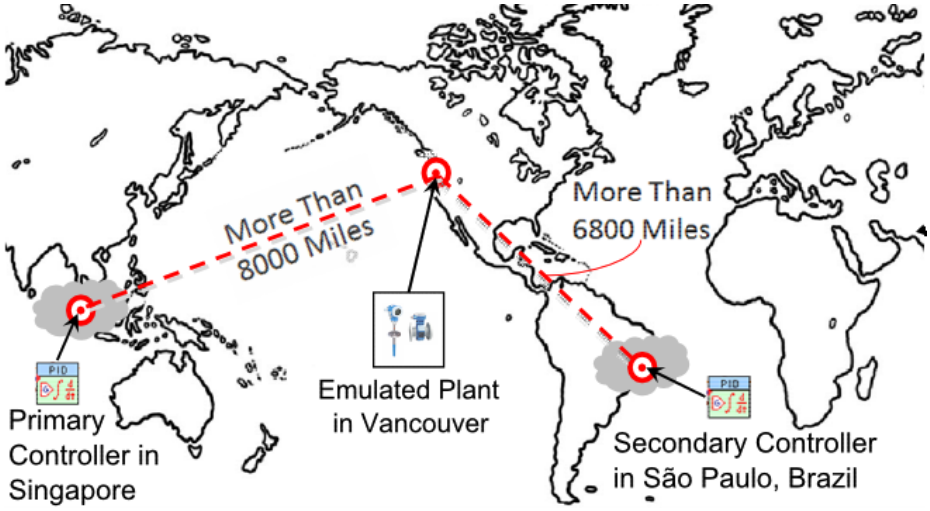


Figure 4.1: Locations of the cloud controllers and plant.

The communication between the plant and the controllers is done using Modbus over TCP protocol. Modbus is an application level protocol that follows the master-slave architecture. It is used as a communication protocol between industrial automation devices like programmable logic controls (PLCs). Using Modbus, the master device sends requests to the slave and waits for the latter to respond.

In our experiments, we inject emulated network delays between the plant and the controllers. In addition, we fail the controllers and bring them back online at random times. Moreover, we consider the case of having more than 2 redundant controllers. Furthermore, we test the system performance in the case of injecting disturbance. The sampling period of the simulated plant is equal to 300 milliseconds.

Performance Metrics. We choose the following as our performance metrics [11]:

- Maximum overshoot percentage: the maximum percentage by which the system response exceeds its target value during the transient response.
- Settling time: the time taken for the system response to stay within 5% of the final value.
- Steady state error: the difference between the reference input signal (the set point value) and the final value of the step response signal.
- Number of generated events: the number of feedback signals that are sent from the plant to the controller. Since the size of the feedback messages in the proposed approach is the same as the one in the periodic approach, the savings in the number of these messages reflect the savings in the communication bandwidth.

4.2 Fault Tolerance Evaluation

Using the above setup, we evaluate our proposed fault tolerance method. We feed the system with a step input at time t=0. At this time the 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 system response during this process is as shown in the blue solid curve in Fig. 4.2. We can see that despite the failure of the primary controller, the system response is close to the case of having no failures that is shown in the red dotted curve in Fig. 4.2. The small difference between the two responses can be seen at t= 30 s when the primary controller failed. At this time, the system response in the case of failures remains the same for a small period of time until the secondary controller becomes engaged and then the response continues rising again towards the steady state. Despite that, the system response in the case of failures does not suffer from any steady state errors just like the no failures case.

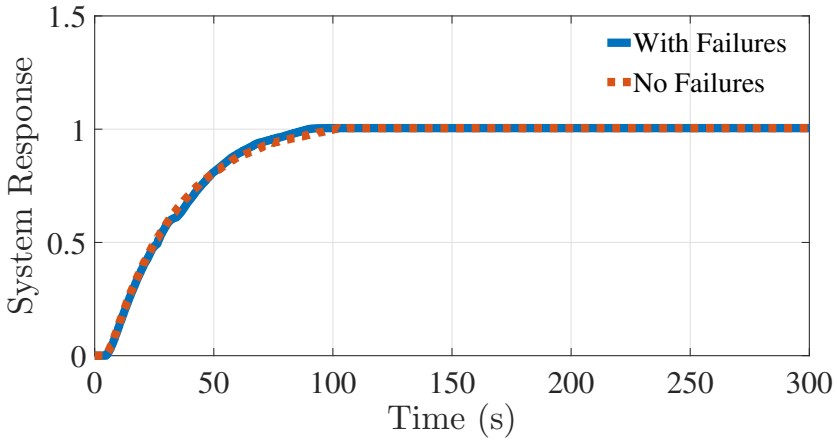


Figure 4.2: The system response with and without failures. The red dotted curve is the system response without having any failures. The solid blue curve is the system response in the case of failures using the fault tolerance algorithm with bumpless transfer enabled.

4.3 Effect of the Smooth Handover

To show the importance of the smooth handover method, we repeat the above experiment after disabling this method. The response curve in Fig. 4.3 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 (117 events on average) is about 59% more than the number

of generated events when using it (48 events on average). This shows another reason why smooth handover is critical in the context of event-based cloud control.

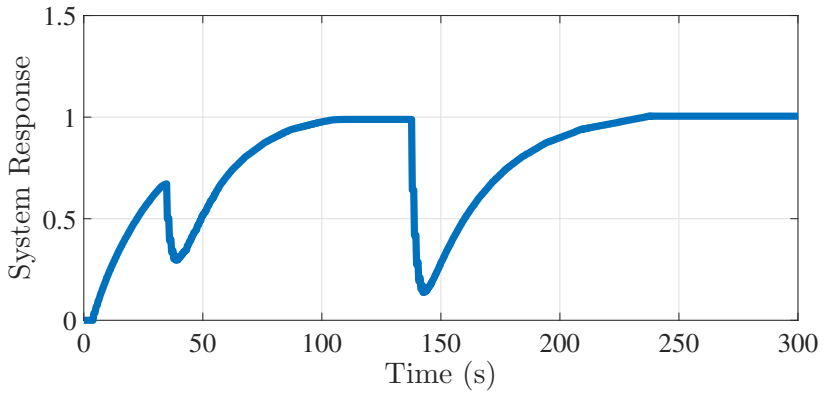


Figure 4.3: The system response using the fault tolerance algorithm with bumpless transfer disabled.

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 11.7% on average of the messages in the periodic case even with the additional events that are generated to overcome the bumps in the response. When the smooth handover is enabled, our approach only needs 4.8% on average of the messages in the periodic case, meaning that our approach saves more than 95% of the bandwidth in this case. It is important to note here that the size of the sent message in the proposed approach is the same as the one in the periodic approach. This means that the savings in the bandwidth are the same as the savings in the number of sent messages.

4.4 Performance Under Variable Internet Delays

In the previous sections, the observed Internet delay between the controllers and the plant is around 200 ms. The reported results above show that such delays do not challenge the proposed system. Therefore, 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 [41] to add additional emulated delays on top of the existing real network delays. The added delays follow the normal distributions that are characterized by their mean μ and standard deviation σ. We use two delay distributions, the first one has μ= 1 s and σ= 0.7 s, while 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

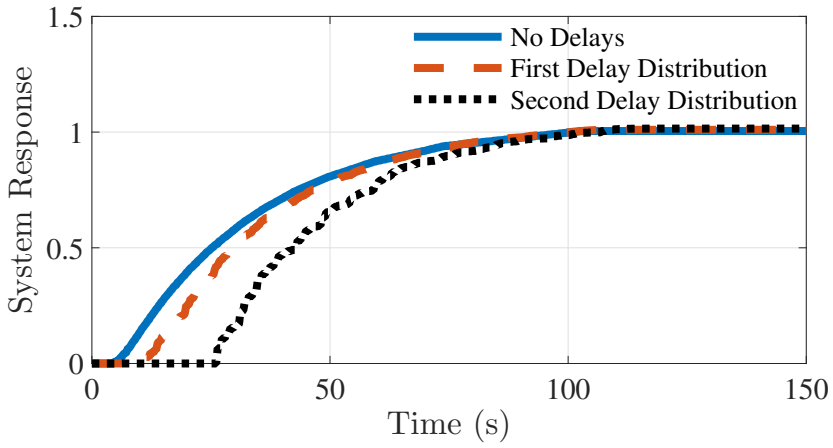


Figure 4.4: System response under three different delay distributions. The blue solid curve shows the system response without adding emulated delays while the red dashed and black dotted curves show the system response under two delay normal distributions with μ= 1 s and σ= 0.7 s (red dashed curve) and μ= 2 s and σ= 1.4 s (black dotted curve), respectively.

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. 4.4 shows the system response in three cases. The blue solid curve shows the system response without adding emulated delays while the red dashed and black dotted curves show the system response under the first and second normal delay distributions, respectively. The figure shows that the three curves are almost the same when it comes to steady state errors and overshoots. However, under the second delay distribution there is a small steady state error (1% of the final value) but in practice, this error is negligible in many applications. We can also notice that the settling time increases with the increase of the added delay. In addition, we emphasize here that these extreme delay cases are rare in practical situations. The main reason for using them is to stress-test the proposed delay compensation technique to show that it can handle them while maintaining good control performance.

Table 4.1 compares the three responses based on overshoots, steady state error, settling time and the number of generated events (during 500 sampling periods) in each case. Notice that in all the cases in the table, our system needs only about 9% of the messages compared with the periodic case.

Table 4.1: Performance measures of the DC Motor speed control system under different delay distributions.

Delay Distribution (s)	Overshoots	Settling time (s)	Steady State Error	Number of Events
No emulated delay	0.0%	78.9 s	0.0%	47
μ= 1 s and σ= 0.7 s	0.0%	78.3 s	0.0%	45
μ= 2 s and σ= 1.4 s	0.0%	85.8 s	1%	44

4.5 Performance Under Controller Failure and Variable Internet Delays

In the previous sections, we test our proposed fault tolerance and our delay mitigation techniques separately. We now show the performance of our proposed fault tolerance in the case of having large network delays and failures simultaneously. We use the same testing setup as mentioned above where the primary and secondary controllers are on VMs in Singapore and Brazil, respectively. The added network delay between the controllers and the plant follows the first normal distribution mentioned above with μ= 1 s and σ= 0.7 s. Under this emulated delay, we fail the primary controller at t=30 s and restart it at t=135 s.

Fig. 4.5 shows a comparison between the performance of the fault tolerance approach with and without emulated network delay. The red dashed curve is the system response under network delay while the blue solid curve is the system response under no emulated delay. We notice that the two curves are almost the same except for the shift that occurred to the system response in the delayed case because of the added network delay. The zooms on the system response in Fig. 4.6 highlight the system responses at the failure and recovery times of the primary controller. Fig. 4.6(a) shows that when the primary controller fails, the system response remains the same for a small time and then rises again gradually when the secondary controller becomes engaged. After reaching the steady state, the system response does not change at all when we restart the primary controller as we can see in Fig. 4.6(b).

Moreover, during 1000 sampling periods, the number of generated events in the delayed response equals 53 events on average which is comparable to the undelayed case (48 events).

4.6 Handling Multiple Redundant Controllers

To show the extensibility of the fault tolerance approach, we test the case of having three redundant controllers; a primary in a VM in Singapore, a secondary in a VM in Brazil and a tertiary controller in a VM in Oregon, the United States (about 500 miles away from the plant in Vancouver, Canada). We start with having the primary controller as the engaged

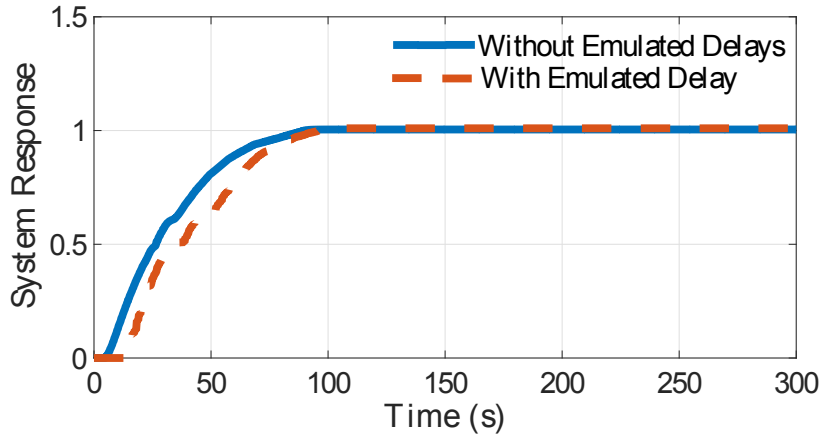
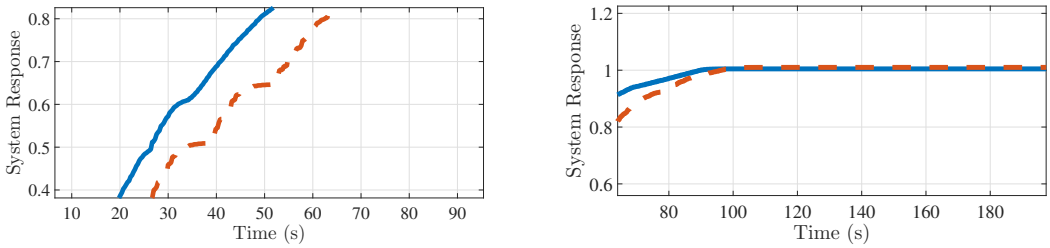


Figure 4.5: The system response when testing the fault tolerance algorithm under failures and variable delays.



(a) The system response when we fail the primary controller at time t= 30 s.

(b) The system response when we restart the primary controller at time t= 135 s.

Figure 4.6: The system response at the failure and recovery times of the primary controller.

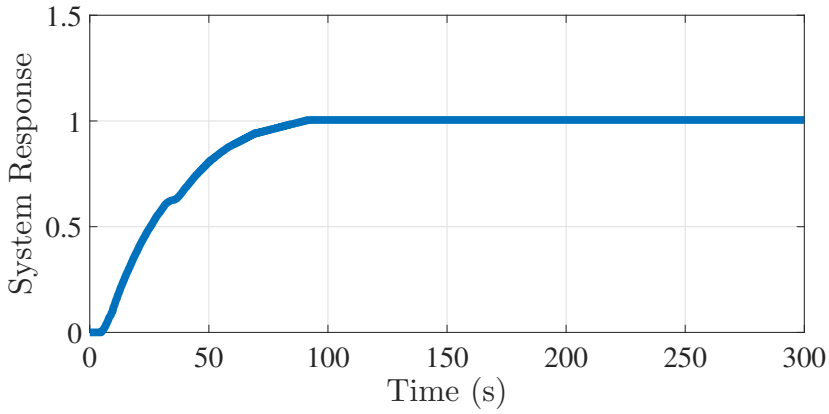
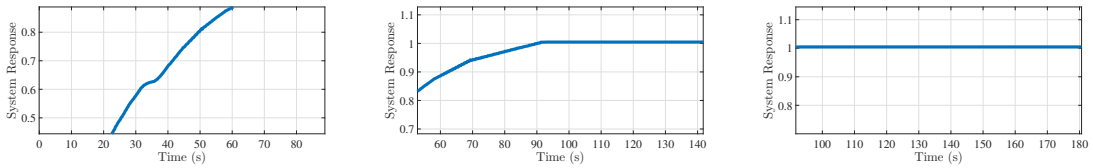


Figure 4.7: The system response using the fault tolerance algorithm with smooth handover enabled while having three redundant controllers.

controller while the rest 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



(a) The system response when we fail the primary and secondary controllers at time t= 30 s.

(b) The system response when we restart the secondary controller at time t= 90 s.

(c) The system response when we restart the primary controller at time t= 135 s.

Figure 4.8: The system response at the failure and recovery times of the primary and secondary controllers.

secondary controller at t= 90 s and then the primary controller at t= 135 s. Fig. 4.7 shows that despite these failure and recovery processes, the plant response achieves almost smooth set point tracking. We say "almost" because at the time between the failure of one controller and the engagement of another one during the transient response, the system response remains fixed. This can be seen at t= 30 s in Fig. 4.8(a) which shows a zoom on the system response at this time. The system response then continues to rise again towards the set point value once the newly engaged controller takes the lead. At the steady state, restarting the secondary and primary controllers does not affect the system response as shown in Figs. 4.8(b) and 4.8(c), respectively.

In addition to set point tracking, during 1000 sampling periods, the number of generated events is equal to 50 events, which is a saving of 95% in the number of required messages in the periodic case.

4.7 Performance Under Disturbance

We consider the performance of our system under disturbances. We use the same test setup above with two controllers, a primary in Singapore and a secondary in Brazil. The test starts with the primary controller in the lead and the secondary controller in the standby mode. We then add a continuous unit load disturbance step at t= 120 s. After that, we fail the primary controller at t= 225 s and restart it again at t= 327 s.

Fig. 4.9 shows that the primary controller can overcome the disturbance and the system response gradually reaches the set point again. However, we notice that the system suffers from the disturbance every time a handover occurs. This can be seen from the bumps in the response in Fig. 4.9. The reason for this problem is that at the handover, the plant sends its last action to the newly engaged controller. This last action is defined as the last executed action by the plant. This is effectively the sum of the last control signal of the last engaged controller and the added disturbance. When a handover occurs, the total control action sent to the plant is equal to this last action value in addition to the disturbance. That is why the system response suffers from the disturbance when a handover occurs.

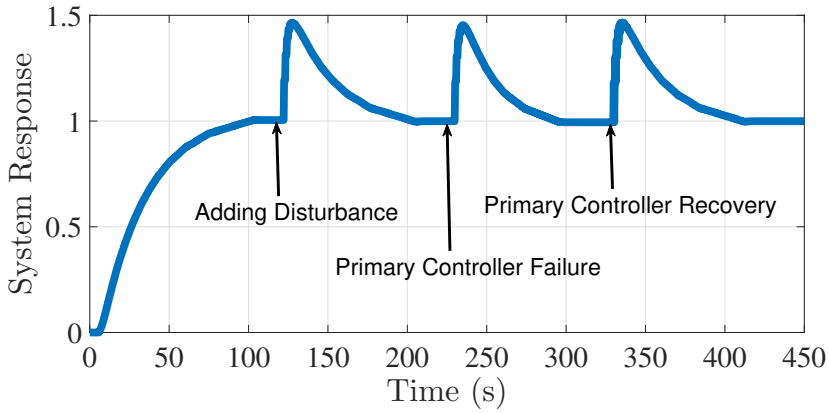


Figure 4.9: The system response using the fault tolerance algorithm under disturbance using the old definition of the last action value.

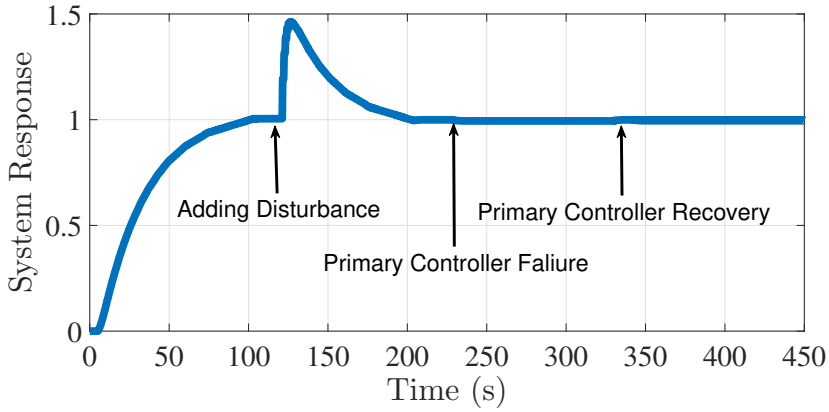


Figure 4.10: The system response using the fault tolerance algorithm under disturbance using our new definition of the last action value when using cloud controllers.

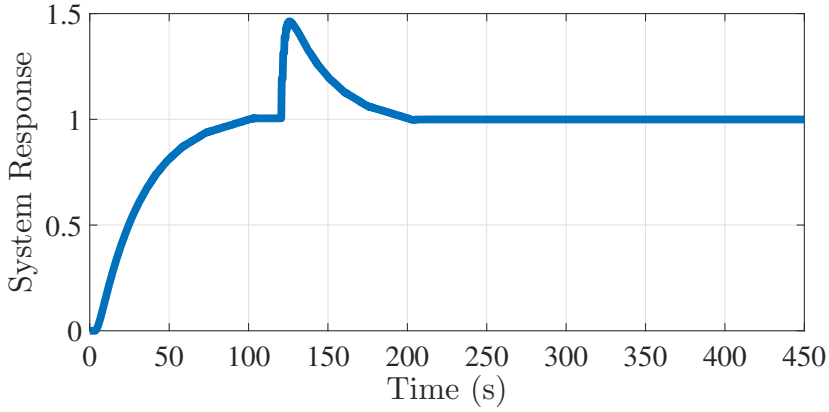


Figure 4.11: The system response using the fault tolerance algorithm under disturbance using our new definition of the last action value when using local controllers.

Table 4.2: Performance measures of the system using the fault tolerance with multiple values of Δ.

Δ value	Overshoots	Settling time (s)	Steady State Error	Number of Events (during 1000 sampling periods)
0.004	0.0%	106.2 s	0.0%	305
0.035	0.0%	98.1 s	0.0%	82
0.07	0.0%	90.6 s	0.0%	48
0.14	1.5%	81.9 s	1.5%	27
0.4	2%	60.9 s	2%	12

To fix this problem, we redefine the last action to be the last control signal of the last engaged controller. Fig. 4.10 shows the system response after this modification. As we can see, the disturbance causes only one bump in the system response when it is introduced at t=120 s and no artifacts occur in the response at the times of the handovers between the two controllers.

We repeat the disturbance experiment when the controllers are local and the system response is shown in Fig. 4.11. We note that the magnitude of the system response when adding the disturbance and the time to get back to the set point value afterwords in the case of using cloud controllers is comparable to the case of using local controllers. In all cases, the magnitude of the system response when adding the disturbance is governed by the system time constant (which is equal to 0.5402 in our plant). In addition, the time to get back to the set point value after adding disturbance can be faster by increasing the K_i parameter of the PI controller. However, this can come at the cost of having oscillations and overshoots in the system response.

It is important to note here that our system needs only 77 events during 1500 sampling periods in this test, saving about 95% of the messages compared with the periodic approach despite having to deal with the added disturbance.

4.8 Effect of Δ Value

In this section, we perform a sensitivity analysis of the proposed fault tolerance and delay compensation techniques using multiple values of Δ which is the parameter involved in the event generation process. We perform the same fault tolerance and delay compensation experiments mentioned above. In the fault tolerance experiments, we fail the primary controller at t= 30 s and restart it again at t= 135 s. In the delay compensation experiments, we add an emulated network delay that follows the normal distribution whose mean μ= 1 s and standard deviation σ= 0.7 s on top of the real network delay. We repeat the fault tolerance and delay compensation experiments with five different values of Δ and compare between the system responses in all these cases based on our performance metrics mentioned

Table 4.3: Performance measures of the system using the delay compensator with multiple values of Δ.

Δ value	Overshoots	Settling time (s)	Steady State Error	Number of Events (during 500 sampling periods)
0.004	0.0%	90.3 s	0.0%	250
0.035	0.0%	84 s	0.0%	78
0.07	0.0%	78.3 s	0.0%	45
0.14	2%	75.9 s	2%	25
0.4	7%	undetermined	6%	10

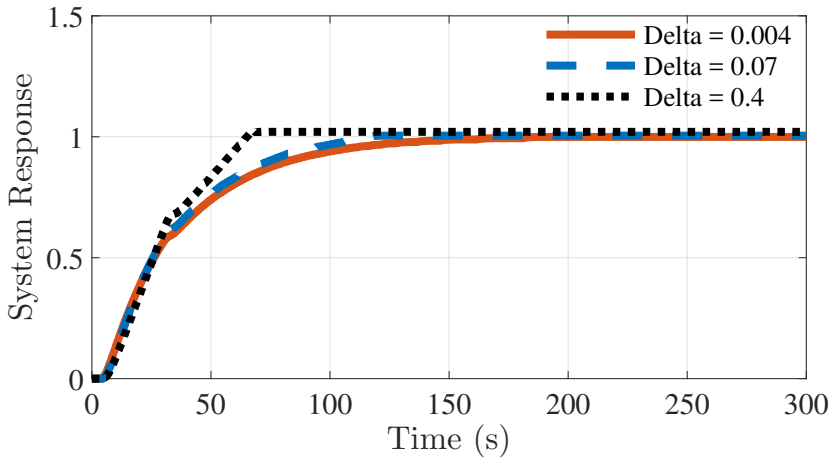


Figure 4.12: The system response using the fault tolerance algorithm with multiple values of Δ.

before. Table 4.2 shows this comparison when we test our fault tolerance. As seen from the table, with smaller values of Δ, the system response does not suffer from any overshoots or steady state errors. These two artifacts occur with larger values of Δ. In terms of settling time and the number of generated events, we can realize from the results that these two metrics decrease as the Δ value gets larger. Table 4.3 shows similar trends when we test our delay compensator with the same values of Δ. We notice from this table that the overshoots are larger when Δ=0.14 and 0.4. With the latter value of Δ, the steady state error is 6% which leads to undetermined settling time (which is defined as the time taken for the system response to stay within 5% of the final value). Fig. 4.12 and Fig. 4.13 show the system response curves using three Δ values in the case of using the proposed fault tolerance and delay compensation techniques, respectively.

The reason behind the decrease of the settling time with larger values of Δ is due to the behavior of the integrator part of the PI controller. Having larger values of Δ leads to lower number of generated events with longer times between consecutive ones. During this long time, the PI controller acts based on the last received output value from the plant. With

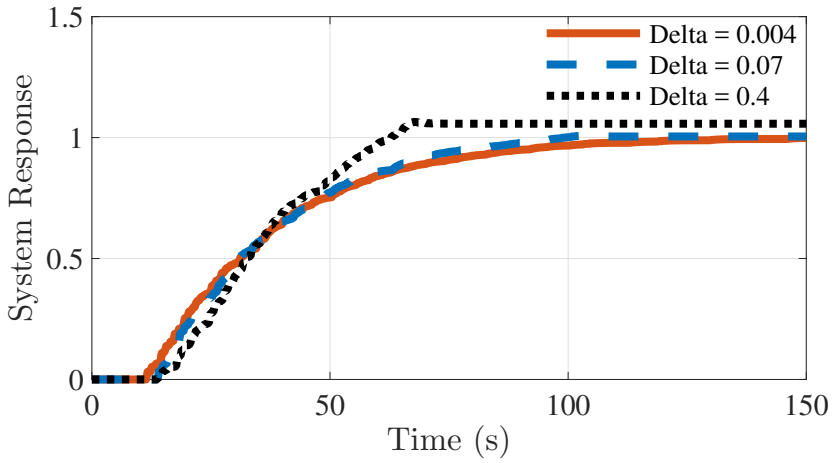


Figure 4.13: The system response using the delay compensator with multiple values of Δ.

this value being the same during longer time periods, the integrator of the PI controller accumulates the same error value every sampling period. This accumulation is translated into an increase in the control signal in order to eliminate this error. As a result, the plant acts aggressively in response to these large control signals which causes a faster response and consequently smaller settling times. In contrast, with smaller values of Δ, the plant sends its output more often to the controller. This means that the error value decreases more frequently. As a result, the integrator accumulates less error which eventually leads to more relaxed control signals. The system response to such signals is smoother, less aggressive and takes longer time to settle.

Beschi *et al.* [14] study the influence of the value of Δ on the event-based control system performance when using the SSOD algorithm. They conclude that the number of generated events is inversely proportional to the value of Δ and that the difference between the event-based control performance and the performance of the traditional periodic control system is upper bounded by the value of Δ. In other words, the choice of Δ is based on the desired number of generated events and the desired precision of the event-based control system, which is aligned with our results above. It is worth noting that the stability of the system is independent of the value of Δ [13] which gives the freedom to the designer to choose Δ according to the desired performance.

4.9 Summary of the Results

The experiments in this chapter showed the robustness of the proposed event-based cloud control system and the associated solutions to the problems of network delays and failures.

Having controllers deployed in VMs thousands of miles away from the plant did not challenge our fault tolerance technique. Handover between controllers happens in a smooth

way thanks to using the bumpless transfer technique from control theory. We showed that this applies to the case of having more than 2 redundant controllers. We saw that smooth handover contributes to saving more events/feedback signals between the plant and the controller in case of failure.

Furthermore, network delays increases the settling time of the system, but other performance metrics stay almost the same. Even with a delay value that is more than 10 times the sampling period of the plant, the only effect is observed in the steady state error and this effect is practically negligible in many applications. We also showed that, under such challenging network delays, our fault tolerance technique is still able to preserve the system performance.

In addition, our system is able to adapt to added disturbances. Our results showed that the system can gradually reject/tolerate such disturbances. Even in the existence of these undesirable disturbances, the handover between controllers upon failures did not suffer from any artifacts. Moreover, we showed how the value of Δ affected the system response using our proposed techniques.

Finally, it was noted that our system is able to perform well according to our metrics using as few as 5% only of the feedback signals that are required in the case of using the periodic control approach during a limited time period. The required feedback signals compared with the periodic approach can be even less than 5% as time goes to infinity especially if handovers do not occur frequently between redundant cloud controllers. This means achieving better network resource utilization without affecting the major control performance metrics of our system.

Chapter 5

Conclusions and Future work

5.1 Conclusions

In this thesis, 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 as well as network failures and packet losses that may 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 solves the delay problem in such a way that guaranteed the system stability. We evaluated the proposed approach under a wide range of delay values. Our simulation results showed the robustness of the proposed approach.

In addition, we presented an approach to deal with network failures and packet losses. We deployed our controllers on virtual machines using Amazon cloud in Singapore and Brazil while having our plant in Vancouver. We tested the system robustness in the case of failures and external disturbances. Our simulation results showed the robustness of our approach even if the cloud controllers were thousands of miles away from the controlled plant.

5.2 Future Work

Our work can be seen as a first step towards a full event-based cloud control system where communications between all the system components occur only when needed. In our proposed structure, only the feedback signals between the plant and the controller are transmitted in an event-based fashion. A possible extension to our work is then to consider the communication in the feedforward direction from the controller to the plant. This may be realized by combining the Symmetric Send on Delta PI (SSOD-PI), that we use in our work, and PI-SSOD structures [13] together in a unified event-based control structure. In such a

structure, stability guarantees should be devised and several modifications may be required to address the network delays and failures problems.

We adopted the event-triggered scheme in sampling the signals in our control structure. A possible direction for future works is to consider the self-triggered scheme. Finally, another direction is extending our work to the include more complex control loops like cascade control [57].

Bibliography

[1] Alaa Eldin Abdelaal, Tamir Hegazy, and Mohamed Hefeeda. Event-based Control as a Cloud Serivce. In *Proc. of American Control Conference (ACC'17)*, Seattle, WA, May 2017.

[2] Introduction to Modbus TCP/IP (white paper). `https://goo.gl/wzq3xx`, 2005.

[3] Faisal Altaf, José Araújo, Aitor Hernandez, Henrik Sandberg, and Karl Henrik Johansson. Wireless event-triggered controller for a 3d tower crane lab process. In *Proc. of IEEE Mediterranean Conference on Control & Automation (MED'11)*, pages 994–1001, Corfu, Greece, June 2011.

[4] Jean Araujo, Manuel Mazo, Adolfo Anta, Paulo Tabuada, and Karl H Johansson. System architectures, protocols and algorithms for aperiodic wireless control systems. *IEEE Transactions on Industrial Informatics*, 10(1):175–184, February 2014.

[5] Karl-Erik Årzén. A simple event-based PID controller. In *Proc. of IFAC World Congress*, pages 423–428, Beijing, P.R. China, January 1999.

[6] J. Karl Aström. *Analysis and Design of Nonlinear Control Systems: In Honor of Alberto Isidori*, chapter Event Based Control, pages 127–147. Springer Berlin Heidelberg, 2008.

[7] Karl Johan Åström and Bo Bernhardsson. Comparison of riemann and lebesque sampling for first order stochastic systems. In *Proc. of IEEE Conference on Decision and Control, (CDC'02)*, volume 2, pages 2011–2016, Las Vegas, NV, December 2002.

[8] Karl Johan Åström and Tore Hägglund. The future of PID control. *Control engineering practice*, 9(11):1163–1175, November 2001.

[9] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[10] Karl Johan Aström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[11] M Bandyopadhyay. *Control Engineering Theory and Practice*. Prentice-Hall, New Delhi, India, 2006.

[12] Jan Dimon Bendtsen, Jakob Stoustrup, and Klaus Trangbæk. Bumpless transfer between advanced controllers with applications to power plant control. In *Proc. of IEEE Conference on Decision and Control (CDC'03)*, volume 3, pages 2059–2064, Maui, Hawaii, December 2003.

[13] Manuel Beschi, Sebastián Dormido, José Sanchez, and Antonio Visioli. Characterization of symmetric send-on-delta PI controllers. *Journal of Process Control*, 22(10):1930–1945, December 2012.

[14] Manuel Beschi, Sebastián Dormido, José Sánchez, and Antonio Visioli. Tuning of symmetric send-on-delta proportional–integral controllers. *IET Control Theory & Applications*, 8(4):248–259, February 2014.

[15] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[16] Ahmet Cetinkaya, Hideaki Ishii, and Tomohisa Hayakawa. Event-triggered output feedback control resilient against jamming attacks and random packet losses. *Proc of. IFAC World Congress*, 48(22):270–275, September 2015.

[17] Armando Walter Colombo, Thomas Bangemann, Stamatis Karnouskos, Jerker Delsing, Petr Stluka, Robert Harrison, François Jammes, and José Luis Martínez Lastra. *Industrial Cloud-Based Cyber-Physical Systems*. Springer, 2014.

[18] Alma Didic and Pavlos Nikolaidis. Real-time control in industrial IoT. Master's thesis, Malardalen University, Sweden, 2015.

[19] VS Dolk and WPMH 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)*, pages 1–7, Krakow, Poland, June 2015.

[20] VS Dolk, P Tesi, C De Persis, and WPMH Heemels. Output-based event-triggered control systems under denial-of-service attacks. In *Proc. of IEEE Conference on Decision and Control (CDC'15)*, pages 4824–4829, Osaka, Japan, December 2015.

[21] Peter C Evans and Marco Annunziata. Industrial internet: Pushing the boundaries of minds and machines, November 2012. Techincal Report, General Electric.

[22] Fulvio Forni, Sergio Galeani, Dragan Nešić, and Luca Zaccarian. Event-triggered transmission for linear control over communication channels. *Automatica*, 50(2):490–498, 2014.

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

[24] Eloy Garcia and Panos J Antsaklis. Model-based event-triggered control with time-varying network delays. In *Proc. of IEEE Conference on Decision and Control and European Control Conference, (CDC-ECC'11)*, pages 1650–1655, Orlando, Florida, December 2011.

[25] Alasdair Gilchrist. *Industry 4.0*. Apress, 2016.

[26] Omid Givehchi, Jahanzaib Imtiaz, Henning Trsek, and Jurgen 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)*, pages 1–4, Toulouse, France, May 2014.

[27] Thomas Goldschmidt, Mahesh Murugaiah, Christian Sonntag, Bastian Schlich, Sebastian Biallas, and Peter Weber. Cloud-based control: A multi-tenant, horizontally scalable soft-plc. In *Proc. of IEEE International Conference on Cloud Computing (CLOUD'15)*, pages 909–916, New York City, NY, June 2015.

[28] Rachana A Gupta and Mo-Yuen Chow. Overview of networked control systems. In *Networked Control Systems*, pages 1–23. 2008.

[29] Wu He and Lida Xu. A state-of-the-art survey of cloud manufacturing. *International Journal of Computer Integrated Manufacturing*, 28(3):239–250, March 2015.

[30] WPMH Heemels, Karl H Johansson, and Paulo Tabuada. An introduction to event-triggered and self-triggered control. In *Proc. of IEEE Conference on Decision and Control (CDC'12)*, pages 3270–3285, Maui, HI, December 2012.

[31] Tamir Hegazy and Mohamed Hefeeda. The case for industrial automation as a cloud service. `nsl.cs.sfu.ca/techrep/SFU-CMPT_TR_2013-25-1.pdf`, June 2013. Techincal Report, Simon Fraser University, Canada.

[32] Tamir Hegazy and Mohamed Hefeeda. Industrial automation as a cloud service. *IEEE Transactions on Parallel and Distributed Systems*, 26(10):2750–2763, October 2015.

[33] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, April 2015.

[34] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A cloud-assisted design for autonomous driving. In *Proc. of SIGCOMM MCC Workshop on Mobile Cloud Computing*, pages 41–6, 2012.

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

[36] D Lehmann and J Lunze. Extension and experimental evaluation of an event-based state-feedback approach. *Control Engineering Practice*, 19(2):101–112, February 2011.

[37] D Lehmann and J Lunze. Event-based control with communication delays and packet losses. *International Journal of Control*, 85(5):563–577, 2012.

[38] Magdi S Mahmoud and Muhammad Sabih. Networked event-triggered control: an introduction and research trends. *International Journal of General Systems*, 43(8):810–827, April 2014.

[39] Marek Miskowicz. *Event-based control and signal processing*. CRC Press, 2015.

[40] Gajamohan Mohanarajah, Dominique Hunziker, Raffaello D'Andrea, and Markus Waibel. Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, April 2015.

[41] Network Emulator For Windows Toolkit. `https://goo.gl/ZIVHJq`.

[42] Pavlos Nikolaidis, Alma Didic, Saad Mubeen, Hongyu Pei-Breivold, Kristian Sandström, and Moris Behnam. Applying mitigation mechanisms for cloud-based controllers in industrial IoT applications. In *Proc. of Internet-of-Things Symposium*, Amsterdam, The Netherlands, October 2015.

[43] Aidan O'Dwyer. *Handbook of PI and PID controller tuning rules*, volume 57. Imperial College Press, 2009.

[44] Andrzej Pawlowski, José Luis Guzmán, Francisco Rodríguez, Manuel Berenguel, J Sánchez, and Sebastián Dormido. The influence of event-based sampling techniques on data transmission and control performance. In *Proc. of IEEE Conference on Emerging Technologies & Factory Automation, (ETFA'09)*, pages 1–8, Mallorca, Spain, September 2009.

[45] Joern Ploennigs, Volodymyr Vasyutynskyy, and Klaus Kabitzsch. Comparative study of energy-efficient sampling approaches for wireless control networks. *IEEE Transactions on Industrial Informatics*, 6(3):416–424, August 2010.

[46] Romain Postoyan, Marcos Cesar Bragagnolo, Ernest Galbrun, Jamal Daafouz, Dragan Nešić, and Eugênio B Castelan. Event-triggered tracking control of unicycle mobile robots. *Automatica*, 52:302–308, February 2015.

[47] Ángel Ruiz, Jorge E Jiménez, José Sánchez, and Sebastián Dormido. A practical tuning methodology for event-based PI control. *Journal of Process Control*, 24(1):278–295, January 2014.

[48] Ángel Ruiz, Jorge Eugenio Jiménez, José Sánchez, and Sebastián Dormido. Design of event-based PI-P controllers using interactive tools. *Control Engineering Practice*, 32:183–202, November 2014.

[49] Carlos Santos, Felipe Espinosa, Enrique Santiso, and Manuel Mazo. Aperiodic linear networked control considering variable channel delays: Application to robots coordination. *Sensors*, 15(6):12454–12473, May 2015.

[50] Moritz Tenorth, Alexander Clifford Perzylo, Reinhard Lafrenz, and Michael Beetz. Representation and exchange of knowledge about actions, objects, and environments in the roboearth framework. *IEEE Transactions on Automation Science and Engineering*, 10(3):643–651, July 2013.

[51] Volodymyr Vasyutynskyy and Klaus Kabitzsch. Implementation of pid controller with send-on-delta sampling. In *Proc. of International Conference Control*, Glasgow, Scotland, UK, August 2006.

[52] M. Velasco, J. Fuertes, and P. Marti. The self triggered task model for real-time control systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS'03)*, volume 384, Cancun, Mexico, December 2003.

[53] Axel Vick, Christian Horn, Martin Rudorfer, and Jorg Kruger. Control of robots and machine tools with an extended factory cloud. In *Proc. of IEEE World Conference on Factory Communication Systems (WFCS'15)*, pages 1–4, Palma de Mallorca, Spain, May 2015.

[54] Axel Vick, Vojtech Vonasek, Robert Penicka, and Jorg 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)*, pages 33–39, Poznan, Poland, July 2015.

[55] Antonio Visioli. *Practical PID control*. Springer Science & Business Media, 2006.

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

[57] Harold L Wade. *Basic and advanced regulatory control: system design and application*. ISA, 2nd edition, 2004.

[58] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. In *Proc. of IFAC World Congress*, pages 6974–6979, Seoul, Korea, July 2008.

[59] Dazhong Wu, Matthew J Greer, David W Rosen, and Dirk Schaefer. Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems*, 32(4):564–579, October 2013.

[60] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9, Spring 2008.

[61] Yuan-Qing Xia, Yu-Long Gao, Li-Ping Yan, and Meng-Yin Fu. Recent progress in networked control systems: A survey. *International Journal of Automation and Computing*, 12(4):343–367, August 2015.

[62] Han Yu and Panos 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*, 49(1):30–38, 2013.

[63] Dong Yue, Engang Tian, and Qing-Long Han. A delay system method for designing event-triggered controllers of networked control systems. *IEEE Transactions on Automatic Control*, 58(2):475–481, 2013.

[64] Xian-Ming Zhang and Qing-Long Han. Event-based dynamic output feedback control for networked control systems. In *Proc. of American Control Conference, (ACC'13)*, pages 3008–3013, Washington, DC, June 2013.

[65] Xianming Zhang, Qing-Long Han, and Xinghuo Yu. Survey on recent advances in networked control systems. *IEEE Transactions on Industrial Informatics*, 12(5):1740 – 1752, October 2016.