Applying Mitigation Mechanisms for Cloud-based Controllers in Industrial IoT Applications

Pavlos Nikolaidis*, Alma Didic*, Saad Mubeen†, Hongyu Pei-Breivold‡, Kristian Sandström‡, Moris Behnam†
*† Mälardalen Real-Time Research Centre (MRTC), Mälardalen University, Sweden
‡ ABB Corporate Research, Sweden
*{adc13001, pns13002}@student.mdh.se
†{saad.mubeen, moris.behnam}@mdh.se
‡{hongyu.pei-breivold,kristian.sandstrom}@se.abb.com

***Abstract*—Cloud computing and Internet of Things (IoT) are two notable concepts that have evolved significantly over the past few years. In the automation industry, clouds are often used for monitoring vast amounts of data generated on the shop floor. Whereas, IoT is used to simplify the end devices and their connections to the rest of the system. In this paper we investigate the interplay of these two concepts and their use in the control applications in the automation industry. We develop a prototype in the industrial setup to explore the use of IoT devices that communicate with a cloud-based controller. Using the prototype, we perform a number of experiments to investigate the consequences of having a cloud server between the end device and the controller. Within this context we consider arbitrary jitter and delays, i.e., they can be smaller, equal or greater than the sampling periods. Moreover, we apply mitigation mechanisms to deal with the delays and jitter that are caused by the local and wide area networks (LAN and WAN).**

I. Introduction

In recent years, cloud computing has drawn the interest of industry. Many research works have been conducted on how cloud infrastructures can be included in industry automation systems eg., [1], [2]. Particularly, clouds have been used for monitoring industrial process, creating short and long term reports for different characteristics of the shop floor. Cloud servers enable permanent storage of long historical data and thus it can be utilized to prevent failures or monitor productivity. Although cloud computing seems adequate to monitor industrial process, it cannot be used to control industrial machines, mainly, due to unpredictable WAN latencies. Therefore, utilizing local resourceful servers, local clouds, can alleviate unpredictable WAN latencies. While local clouds seem to offer an attractive solution, it is expensive to acquire and maintain such infrastructure.

Recent advantages in IoT can be leveraged to solve the aforementioned cost problem. Accordingly, costly industrial controllers can be replaced by cheaper components that can sense the process and send data to the local cloud. Since the devices in the context of IoT are not very resourceful, they cannot handle intensive tasks. Hence, a local powerful cloud infrastructure is needed. A main advantage of devices in the IoT concept is that they can enter or leave the network dynamically without influencing the rest of the system. As a result, observability of the system can be increased or decreased on demand. In addition, the centralized controller running on the cloud can perform optimizations or changes in the process that can instantly affect the entire system. Due to the different services that are expected by local clouds, Cisco proposed a 3-tier local cloud computing architecture called *fog computing* [3]. Fog computing has been introduced to overcome the drawbacks of cloud computing.

A. Problem Statement & Paper Contribution

IoT, fog computing and cloud computing can be combined to form a new operational scheme that can benefit the industrial applications in terms of scalability, flexibility and cost effectiveness. In it, the IoT devices would be communicating with the controller located on the local cloud server, providing real-time control as a service. One of the main challenges that is faced when a networked controller is included in the closed-loop control system is how latencies affect the performance of the system. Another challenge is how to mitigate or even compensate these effects on the end device.

In order to address these challenges, we develop a control system application prototype by exploiting the principles of IoT, fog computing and cloud computing. Using the prototype, we perform a number of experiments to investigate the impact of local and wide area networked controller on the closed loop control. In order to do the performance evaluation, we consider arbitrary jitter and delays, i.e., they can be smaller, equal or greater than the sampling periods. Additionally, we apply two mitigation mechanisms for the end device. These mechanisms do not use any internal information from the controller, in fact, they rely only on the received data.

B. Paper Layout

The rest of the paper is organized as follows. Section 2 describes the related work. Section 3 describes the prototype architecture. Section 4 presents the experimental evaluation. Section 5 discusses the results. Section 6 concludes the paper.

II. Background and Related Work

A. Background

Cloud computing is an operational scheme that provides resources such as computational power and storage to users. Cisco, introduced a new operational scheme called fog computing in 2012. Fog architecture is organized in 3 tiers. The first tier is responsible for Machine-to-Machine (M2M) communication and supports up to sub-second responses. The first tier can also filter or pre-process the vast amount of data before it gets sent to higher tiers. That way the higher tiers only get the selected information they need and can easily communicate with multiple first tier fog nodes. The second and third layers are mainly for Human-to-machine (H2M)

interactions and they can provide up to sub-minute responses. The second and third tier can also filter data before sending them to cloud for further processing. Fog computing is very important in the IoT era.

Internet of things (IoT) [4] has gained wide popularity recently. Research studies predict that IoT will bring around 26 billion devices to the connected world [5]. In IoT, devices are connected through a network. They share data, information and even resources to accomplish their goal or increase total system intelligence. As a result, IoT has applications in a wide range of areas, such as health monitoring, home automation, environmental and agricultural applications, etc [6]. A report in The Economist states that cows will be monitored to ensure the quality of meat and each cow will produce 200MB of data each year. Although this is not much data, USDA estimates that there are around 1.2 billion cows around the world resulting in 224 PB of data each year for cow monitoring only[1]. IoT introduces a new data flow and fog computing can be used to analyze and extract useful data. Since it is not possible to store all this data in clouds for processing, fog servers will be used for pre-processing to filter data.

The aforementioned data flow is also interesting for industrial applications. The industries probably produce more data than cattle and can take advantage of the full potential of fog architecture. While the first tiers of fog can be used for closed loops between industrial machines and fog server, the other two layers can use data to monitor the machine and prevent future hardware failures. While the latter has been investigated, we focus more on the closed control loops over a network.

A typical closed control loop, as shown in Figure 1, consists of a controller that controls a physical process through sensors and actuators. The controller is usually located close to the actual process. In our case we investigate the use of a controller that is placed on a server, local or remote, as shown in Figure 2. In this case there are latencies from the network that are included in the control loop. Depending on the system and on the duration of the latencies this can cause various problems for the system, and make it unstable.

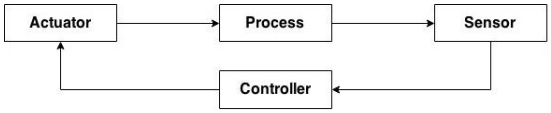

Fig. 1. Closed control loop

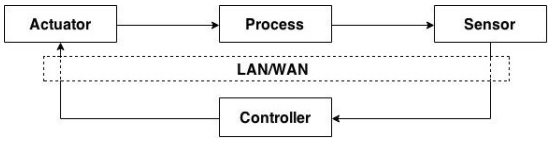

Fig. 2. Closed loop with networked controller

B. Related work

Utilizing cloud computing in industrial automation isn't a completely new idea. However, most of the existing works focus on higher levels than the field-level, where clouds are used for data monitoring and assisting in management, such as [1], [7], [8], [9]. However, very little work is done so far that involves cloud computing within control loops in the industrial automation [8], [10]. The problem of Networked Control Systems (NCS) has been adressed in 2001 in [11]. Plenty of work exists regarding this problem such as [12], [13]. In NCS the controller is tuned in order to compensate network delays or data dropout. In our case the local tuned controller is moved to the cloud and delay mitigation mechanisms are utilized on the actuator-sensor side. These mechanisms are aimed to mitigate the effects introduced by the network. Hegazy et al. [14] consider an industrial scenario where the controller is placed on the cloud. They consider having redundant controllers for fault tolerance and to enable the use of different cloud providers, and they provide an algorithm for a smooth transition between the different controllers. They also propose a delay compensator based on a smith predictor to mitigate the delays of a remote server. However, the smith predictor is a model based predictor, and in our case the goal is to have a simple mitigating mechanism on the device side that the IoT device can work with. Additionally, the cases they consider the delays are considerably smaller than the sampling period. We not only consider this case but also investigate the cases where the delays are considerably longer than the sampling period. In [10], Givehchi et al. investigate the use of a virtualized Programmable Logic Controller (PLC). They show that a virtualized PLC can provide similar behaviour to a hardware PLC, but it's performance decreases considerably when the sampling frequency increases.

III. Proposed System Architecture

A. Prototype architechure

A prototype is built in order to create a representation of the real case. The setup, shown in Figure 3, consists of an end device connected to a switch through which it can reach a local server (fog node) or a remote one (cloud node). Since the setup is realised in a local environment, a delay emulator is used for simulating the distance between the device and the cloud node.

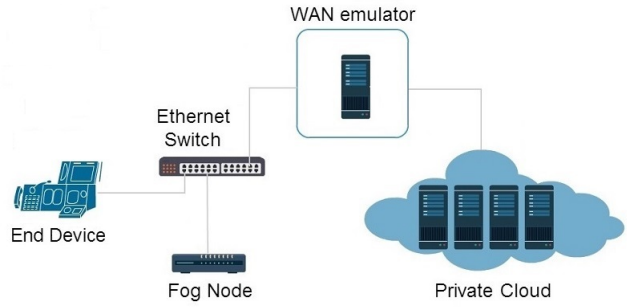

Fig. 3. Architecture overview

The fog and cloud nodes are set up as an Ubuntu server each. For emulating the connection over WAN with the cloud, a software solution called WANEM[2] is used. WANEM is based on the Linux kernel and it utilizes NetEm[3], among other, functionalities in a simple graphical user interface provided on a live Knoppix[4] disk. WANEM enables specifying typical

[1]http://www.ers.usda.gov/topics/animal-products/cattle-beef/statistics-information.aspx

[2]http://wanem.sourceforge.net/

[3]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[4]http://knoppix.net/

network problems, such as delays, jitter, packet loss, packet corruption, connection loss, etc. It also enables specifying the bandwidth, setting a correlation percentage for each characteristic and a few distributions for the delays (normal, pareto, paretonormal) for simulating various realistic internet conditions. All communication routed through WANEM is affected by the set parameters. The end device is an Arduino Uno[5] running a small program that involves sensing and actuating. It sends the sensed value to the server (fog or cloud) where a control value is calculated. The control value is then sent back to the Arduino to actuate. The control loop consists of a photo resistor and an LED. The controller can change the brightness of the LED until it reaches the desired value (set to 700 in the experiments). For the device-server communication, two different approaches are used. The first one, called polling, is listed in Algorithm 1. It involves the Arduino waiting for, or *polling*, the server to establish the connection. In this case, after sampling the sensor value and sending it to the server, the device does nothing until the response from the server is received and could potentially wait forever for it. This approach could be used in a scenario where the sensor and actuator are placed on the same device.

Algorithm 1 Polling

```
1:  procedure POLLING CONTROLLER
2:  begin:
3:      sensorValue ← readSensor();
4:      sendMsgToServer(sensorValue);
5:  loop:
6:      if resultAvailable() then
7:          result ← readResult();
8:          applyValue(result);
9:          goto begin;
10:     goto loop;
```

The second approach, called non-polling, is listed in Algorithm 2. In this case the device sends the sensed value and checks if a reply is available. If there isn't any message, it does not wait for the server to reply, hence it is *not polling* the server. Instead, it repeats its cycle, sending another value and checking for a received message again. On the server side this means that multiple messages with the same value will be received. This might lead the controller to think that the calculated values are taking no effect and thus respond more aggressively, which is not a desired behaviour. This approach suits a case where the sensor and actuator are on separate devices.

B. Delay mitigation

In order to mitigate delays caused by the networked controller, having some simple mechanism on the device side is desirable. Typically, for control loops with time delays (also referred to as dead time) in which the sensing needs to be done a certain time after the actuating, a model based controller is used. The controller then contains a model of the process it is controlling and is able to time the adequate response based on the models behaviour. However, these controllers are not intended for variable delays and do not fit into the idea of the

[5]http://www.arduino.cc/en/Main/ArduinoBoardUno

Algorithm 2 Non-polling

```
1:  procedure NON-POLLING CONTROLLER
2:  begin:
3:      sensorValue ← readSensor();
4:      sendMsgToServer(sensorValue);
5:      if resultAvailable() then
6:          result ← readResult();
7:          applyValue(result);
8:      goto begin;
```

desired simple mechanism. In our case the control is achieved with a PI controller. Since the delays affect the integral term of the controller, an adaptive PI controller [15] would be sufficient to deal with the delays. The adaptive controller can be tuned to adapt to the delays and is simple enough for our purposes. Statistical prediction methods are also used for predicting the delays [16] or, in dynamic offloading, to asses the current state of network parameters (such as bandwidth or loads on the server end) to avoid measuring them too often which increases overhead [17]. This is another method that was considered in our experiments, where the predictions could be used to predict a missed value from the controller. Due to the nature of the algorithms we use (polling and non-polling) and the considered mitigation methods, the adaptive PI controller suits the non-polling approach while the prediction methods are used on the polling approach. In order to improve the polling method, a timeout is added and a prediction method is used in case no message is received from the server before the timeout. This timeout is set to the average round trip time (RTT) for the device-server communication, which is 18 milliseconds. Two prediction methods are used: exponential moving average and double exponential smoothing model. Both of these prediction methods make use of all the data collected, without having to store or manage a large number of variables or the need to collect more than two values to start off. The exponential moving average [18], also called exponential smoothing, calculates a weighted average of the previous data values x_t, as presented in (1), where A is a value between 0 and 1. The predicted value for the next cycle, F_{t+1}, is the value calculated in the current cycle ((2)).

$$s_t = Ax_t + (1 - A)s_{t-1} \tag{1}$$

$$F_{t+1} = s_t \tag{2}$$

The double exponential smoothing average model is calculated by the "Holt model" forecast [19]. It calculates two terms in each cycle, as seen in (3) and (4), and has two smoothing factors A and B, both of which take values between 0 and 1. The second term b_t represents the change in the slope, or the trend. This method calculates F_{t+m}, the predicted value of x_{t+m} at time $t + m$, $m > 0$, by using (5)

$$s_t = Ax_t + (1 - A)(s_{t-1} + b_{t-1}) \tag{3}$$

$$b_t = B(s_t - s_{t-1}) + (1 - B)b_{t-1} \tag{4}$$

$$F_{t+m} = s_t + mb_t \tag{5}$$

In case of the non-polling approach, due to delays, the controller receives the same value multiple times before a change is registered. When the controller receives multiple

messages with the same value, it affects the integral factor of the PI controller. In this case an adaptive PI controller is suitable to mitigate the affects of the delays on the controller. A smoothing factor, a, is calculated based on the sampling period and the round trip time, as shown in (6) to lessen the influence of these changes caused by delays.

$$a = \frac{sampling\ period}{sampling\ period + RTT} \tag{6}$$

This smoothing factor is then applied to the received values before they are used by the actuator.

IV. Experimental Evaluation

We have conducted a large number of experiments in [20]. In this section we discuss the results obtained from both the polling and non-polling approach, where delays and jitter are added to the control loop. The sampling period is set to 14 ms. First the prototype setup is tested with delays and jitter that are comparable to the sampling period, for both polling and non-polling approach. Then the system is tested with delays and jitter that are considerably larger than the sampling period, again for both approaches. The results of these tests are compared to the results where the system implements the delay mitigation mechanisms proposed in Section III-B. The results show that the system has no difficulties with small delays and jitter, in any of the mentioned cases. Therefore, the following results are of the tests where the longer delays and jitter are used.

A. Case 1: Response without mitigation mechanisms

First we investigate the effects of delays and jitter that exceed the sampling period. Delays and jitter are increased until the system starts oscillating. In Figure 4a, the polling approach is used and the system becomes unstable when the delay exceeds 50 ms. The oscillation for 50 ms delay exceeds the limit of the steady state error therefore the system is considered unstable. In Figure 4c, the non-polling approach is used. It can be noticed that the system tolerates smaller delays compared to the polling approach. The system becomes unstable for 23 ms of delay. Moreover, the systems' response downgrades significantly from 15 ms delay. The degradation of system's performance, for this approach, starts when the sensor sends consecutive messages before the actuator receives a message. That happens when delays exceed the sampling period.

In Figures 4b and 4d, the effect of jitter in the control loop is depicted for polling and non-polling approach respectively. As it can be noticed, the overshoot increases as the jitter increases, but the system manages to reach the steady state after some time. Jitter does not affect the system in the same way that constant delays do. Applying jitter means that there will be a variable delay between 0 and the specified value of the jitter. Because of this, the controller can receive some values before the sampling period exceeds but also some that exceed the sampling period for a small amount. Since jitter is a variable delay, it may affect the control loop in a different way on every run.

B. Case 2: Response with adaptive PI controller

In Figure 5a, the adaptive PI controller is used, and the smoothing factor according to formula (6) is set to 0.31. Compared to the behaviour of the controller before the smoothing factor is applied, overshoot has been decreased from 142 to 56 and settling time from 1415 ms to 601 ms.

In Figure 5b, the system becomes unstable before the smoothing factor is applied; whereas, when the smoothing factor is applied it manages to settle after 686 ms with an overshoot of 59. The smoothing factor is set to 0.26. In Figures 5c and 5d, jitter is set to 25 ms and 75 ms respectively. The smoothing factor is computed and RTT is set to the maximum possible value. Smoothing factors are set to 0.26 and 0.1 respectively. In the first case, overshoot has decreased from 134 to 0 and the settling time from 384 to 199. When the jitter is set to 75 ms the system becomes unstable. Utilizing the smoothing factor the system manages to stabilize after 1264 ms with an overshoot of 62.

C. Case 3: Response with prediction methods

Figure 6a shows the system is under a constant delay of 75 ms. In this case, the two prediction methods are compared with respect to the response of the polling method when executed with the same delay. While the exponential moving average has a smoother transition, the double exponential smoothing has a faster settling time. Both methods, however perform better than than the original system.

Similar results are gained when a 100 ms delay is applied, as shown in Figure 6b. We can notice that the responses of both methods have disturbances in the beginning. This is because a smaller number of messages is available in the message buffer in the beginning compared to later, which means the system has to rely on the prediction methods for input. However, the prediction methods depend on the values from the server to improve their accuracy. This is less of a problem later on once the system becomes more stable, since changes in the slope are smaller and therefore it is less likely to predict a value that stands out significantly.

As for the jitter, the system performs much better as compared to the constant delay. Figures 6c and 6d show the prediction methods performing with 25 ms and 75 ms of jitter, respectively. We can see that in both cases, there is barely any overshoot, and compared to the tests with constant delays, the curves are much smoother in the beginning. This is because the length of the delay varies and the system performs better when the system uses less predicted values in the beginning. In the case of jitter there is barely any noticeable difference between the two prediction methods, although the exponential moving average settles slightly faster.

V. Discussion

The controller is clearly affected by the longer delays and jitter. However, the system manages to stabilize when the latencies are smaller than the sampling period for both polling and non-polling approaches. Introducing a local server in the control loop adds a small delay of 2-3 ms and could be considered. However, the setup should be tested with increased load on the server to see how it would affect the response time. The adaptive PI controller is an intuitive solution. It manages to stabilize the response of the system, and offers a great improvement to the non-polling approach in all cases. However, it has no proof for the robustness of the algorithm. It should be examined on a more complex system or have a mathematical proof in order to prove its robustness. The prediction methods perform better under jitter than under constant delays. This is because both methods depend on

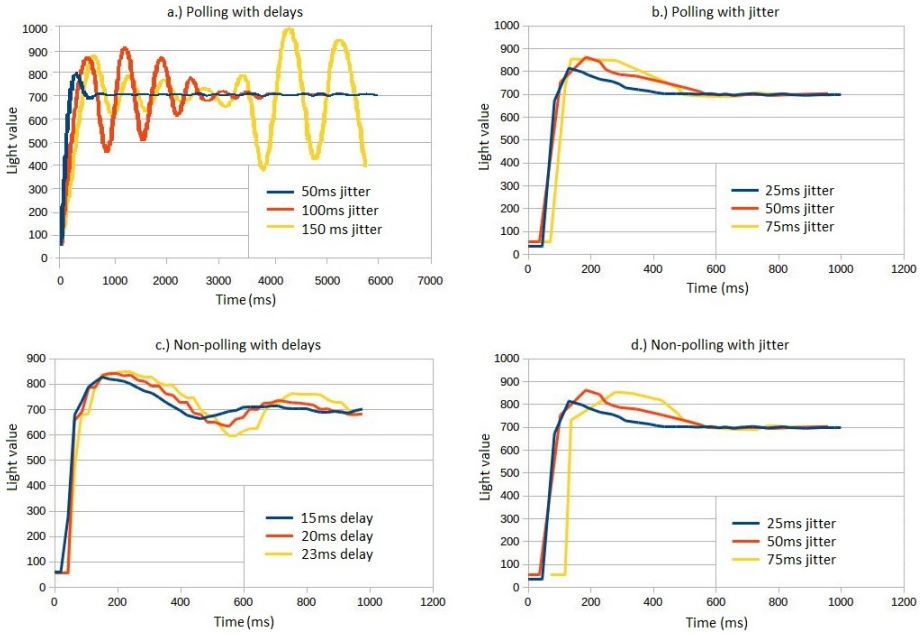

Fig. 4. Local response of the controller under delay and jitter, both polling (a and b) and non-polling (c and d) approaches

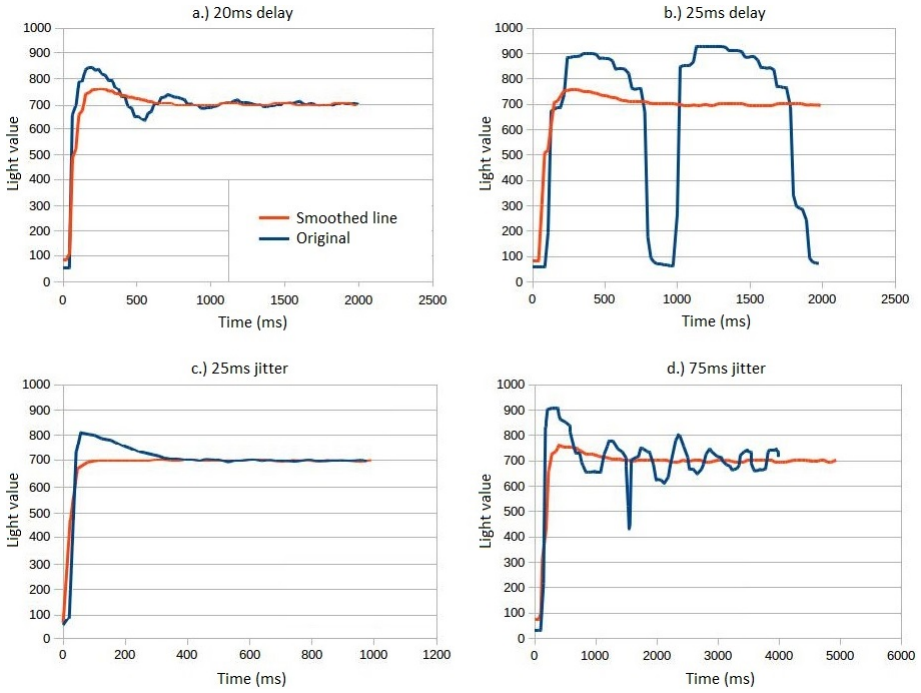

Fig. 5. Adaptive PI at different delays (a and b) and jitter (c and d)

updating their calculations, meaning they depend on getting correct values from the server. In the case of constant delays, most predictions happen in the start while the control values change and before the system reaches its stable point. As the delays increase, the system relies on more consecutive predicted values, worsening the response of the system. In the case of jitter, the total amount of predictions made is roughly the same as for the delays, but the predictions are more distributed over time rather than concentrated in the start, and thus the response is much better. Even though the delay mitigation mechanisms manage to improve the responses under delays and jitter, none of the results are comparable to the local control with no latencies. The system examined here is a relatively simple case. However, the results are reasonable considering the ratio of the sampling period of the process and the delays applied. Most of the research regarding hosting controllers on a cloud, as in [14], consider a more comfortable difference between the sampling period and the network delay.

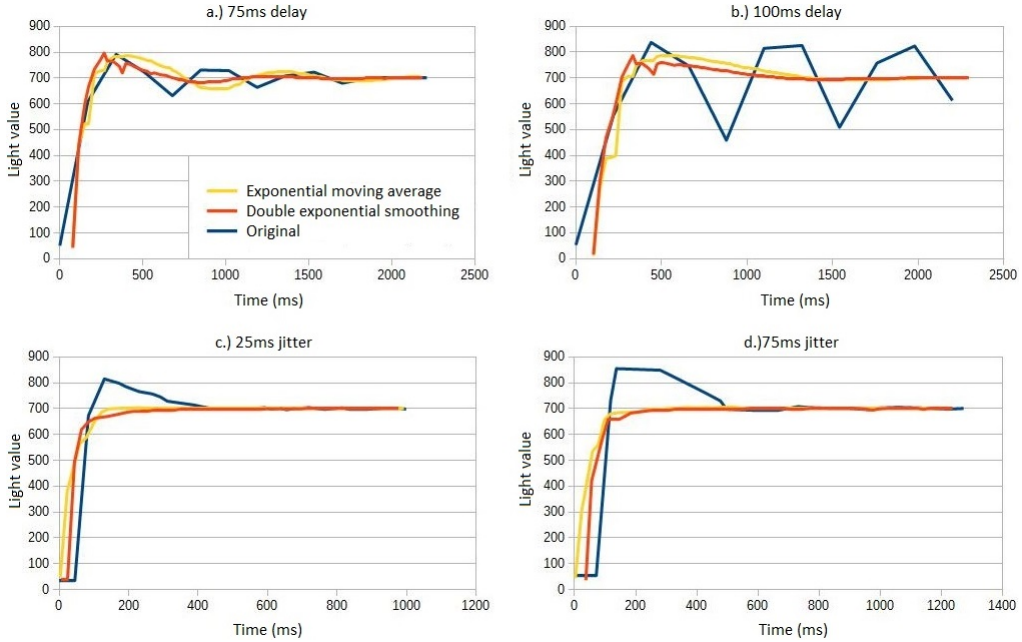

Fig. 6. Prediction methods under delays (a and b) and jitter (c and d)

VI. Conclusion and Future Work

In this paper we have investigated the effects of offloading the controller to a remote server. We built a prototype to create a realistic scenario. Moving the controller to a remote server degrades system's performance. Even small delays affect the control loop when they exceed the sampling period. The forecasting methods that we have investigated, work better with variable delays because the predicted values that are used are more scattered, as opposed to constant values where the predictions are concentrated at the beginning. However, these methods are are challenged in the case when consecutive predicted values are needed, especially in the start up of the control system. Whereas, the adaptive PI controller uses an intuitive idea for mitigating delays inside the network. A formal proof for the robustness of such approaches is needed.

Since not much research has been conducted in this area, there are many possibilities left to try out in the future. The polling and non-polling approaches can be expanded into a more sophisticated system. Moreover, loads can be introduced on both ends to simulate a more realistic approach. These methods can also be tested with a more complicated system, described by a high-order differential equation.

Acknowledgement

The work in this paper is supported by the Swedish Foundation for Strategic Research and the Swedish Knowledge Foundation (KKStiftelsen) within the projects PRESS and RV-REDS respectively. We thank our industrial partner ABB Corporate Research, Sweden.

References

[1] R. Langmann and L. Meyer, "*Automation services from the cloud*," in *11th IEEE International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2014, pp. 256–261.

[2] O. Givehchi and J. Jasperneite, "*Industrial automation services as part of the Cloud: First experiences*," *Proceedings of the Jahreskolloquium Kommunikation in der Automation–KommA, Magdeburg*, 2013.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "*Fog computing and its role in the internet of things*," pp. 13–16, 2012.

[4] K. Ashton, "*That 'Internet of Things' Thing, in the real world things matter more than ideas*," http://www.rfidjournal.com/articles/view?4986, June 2009, [Online; Accessed 06-February-2015].

[5] Gartner, "Gartner says the internet of things installed base will grow to 26 billion units by 2020," http://www.gartner.com/newsroom/id/2636073, accessed 20-March-2015.

[6] A. Serbanati, C. M. Medaglia, and U. B. Ceipidor, *Building blocks of the internet of things: State of the art and beyond*. INTECH Open Access Publisher, 2011.

[7] H. Sequeira, P. J. Carreira, T. Goldschmidt, and P. Vorst, "*Energy Cloud: real-time cloud-native Energy Management System to monitor and analyze energy consumption in multiple industrial sites*," in *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2014.

[8] O. Givehchi, H. Trsek, and J. Jasperneite, "*Cloud computing for industrial automation systems - A comprehensive overview*," in *18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, pp. 1–4.

[9] J. Delsing, F. Rosenqvist, O. Carlsson, A. W. Colombo, and T. Bangemann, "*Migration of industrial process control systems into service oriented architecture*," in *38th Annual IEEE Conference on Industrial Electronics Society (IECON)*. IEEE, 2012, pp. 5786–5792.

[10] O. Givehchi, J. Imtiaz, H. Trsek, and J. Jasperneite, "*Control-as-a-service from the cloud: A case study for using virtualized PLCs*," in *10th IEEE Workshop on Factory Communication Systems*, 2014.

[11] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *Control Systems, IEEE*, vol. 21, no. 1, pp. 84–99, 2001.

[12] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceesings of the IEEE*, vol. 95, no. 1, 2007.

[13] T. C. Yang, "Networked control system: a brief survey," *IEE Proceedings-Control Theory and Applications*, vol. 153, no. 4, 2006.

[14] T. Hegazy and M. Hefeeda, "Industrial automation as a cloud service," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2015.

[15] K. J. Åström and T. Hägglund, *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[16] X.-L. Zhang and P. Liu, "A new delay jitter smoothing algorithm based on pareto distribution in cyber-physical systems," *Wireless Networks*, pp. 1–11, 2015.

[17] C.-S. Shih, S.-M. Wang, J. Chen, and Y.-H. Wang, "*Workload migration framework for streaming applications on smartphones*," in *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2014, pp. 1–8.

[18] R. G. Brown, *Smoothing, forecasting and prediction of discrete time series*. Courier Corporation, 2004.

[19] J. Guerard, *Introduction to financial forecasting in investment analysis*. Springer Science & Business Media, 2013.

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