Towards In-Network Industrial Feedback Control

Jan Rüth, René Glebke, Klaus Wehrle
{lastname}@comsys.rwth-aachen.de
Chair of Communication and Distributed Systems
RWTH Aachen University

Vedad Causevic, Sandra Hirche
{lastname}@tum.de
Chair of Information-oriented Control
Technical University of Munich

ABSTRACT

Controlling physical machinery and processes is at the core of production automation. However, challenged by inflexibility, automation and control is evaluating to outsource this control to resourceful cloud environments. While this enables to derive better control through a plethora of measurements, it challenges the control quality through delay introduced through networks.

In this paper, we show how to unify control and communication by offloading delay sensitive control tasks from the cloud to local network elements — a previously unexplored area for in-network processing — enabling both, ultra high quality-of-control while enabling scalable orchestration through cloud environments. Our implementation demonstrates how we combine state of the art control with communication. We achieve this by expressing the control and the datapath in P4 which we synthesize to BPF programs that we execute in XDP environments on Netronome SmartNICs. Further, we highlight the demands of control towards communication to build more involved and complex in-network controllers.

1 INTRODUCTION

The automation and control of industrial machinery is governed by carefully designed control processes and algorithms [17]. Widely known non-industrial control processes range from the self-stabilizing Segways or drones to vehicular cruise controls or house heating systems. In industry, these processes are at the core of robotic automation but they carry through all aspects of modern production. However, driven by increased demands in flexibility and manageability of production sites, control theory is evaluating the softwarization of their field, i.e., the outsourcing of highly localized control processes to general purpose cloud environments [20, 21, 23, 24]. In that regard, control e.g., hopes to utilize the elastic computing and storage of cloud environments to synthesize advanced controllers, e.g., accounting for wear and tear of machinery through constant collection of sensory data and the application of machine learning. However, even though Network Control Systems (NCS) describe the general combination of networking and controlling, they are inherently challenged by *delay* and *jitter* [2, 25]. These

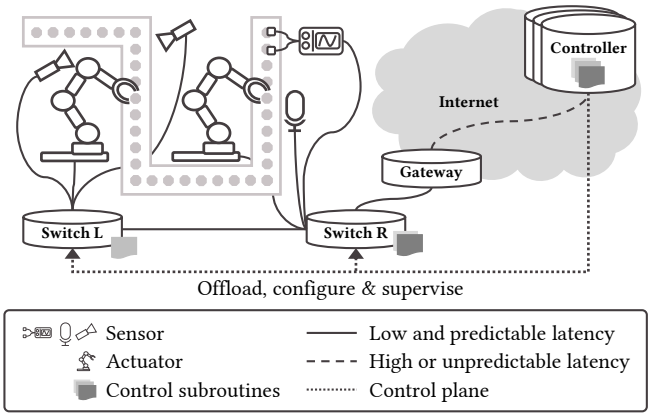


Figure 1: To utilize the resources of cloud computing, process control can offload delay sensitive control tasks to network elements overcoming unpredictable and high latencies of Internet communication.

uncertainties are currently tackled by incorporating worst-case roundtrips times into the control design [10], leading to reduced control performance, e.g., motors can only be controlled with lower speeds slowing down production.

To this end, this paper demonstrates the applicability of in-network processing for control — a previously unexplored area — by offloading small but critical control tasks into network elements managed and organized through remote (cloud) environments. Figure 1 visualizes the orchestration of a factory network from a distant cloud environment enabling to utilize the ultra-low latencies of local communication. Thus, we enable to combine the advantages of localized in-network processing with the resourceful cloud. Specifically, we synthesize control algorithms with a P4 [4] dataplane that we compile to BPF programs and offload to XDP environments as well as onto Netronome Agilio SmartNICs. By building upon the flexibility of P4, we enable to express control demands into the communication path, e.g., we can create lightweight encapsulating protocols that can meet the demand of control for guaranteeing correct, duplicate-free, and fresh data. Further, we can easily control the data flow from sensors through in-network control elements to actuators by directing the respective packets. Specifically, our work makes the following contributions:

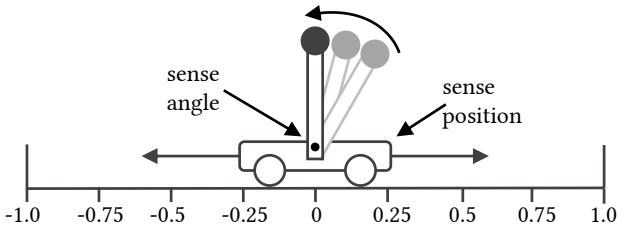


Figure 2: A pendulum on a stiff rod is balanced on top of a cart by moving the cart horizontally. Angle and position can be sampled periodically through sensors.

- We demonstrate the applicability of in-network control to a new and yet unexplored problem space — Network Control Systems.
- Our evaluation uncovers the deep coupling between delay and quality-of-control showing that in-network processing can be a solution to combine resourceful computing with low-latency processing.
- We show that state of the art control can be offloaded to network elements with today's technologies.
- Our analyses further highlight the challenges and demands of control to be applied in a comprehensive manner by in-network processing.

Structure. Section 2 introduces fundamental concepts of control that are relevant for this paper. Following, Section 3 describes our approach in unifying low latency control and cloud computing through in-network processing. We highlight the feasibility of our approach in Section 4 by showing the advantages of in-network processing for control problems. Section 5 discusses future challenges of in-network processing for control tasks. We shortly review related work in Section 6 and conclude the paper in Section 7.

2 NETWORK FEEDBACK CONTROL

Before we begin to explain our approach, we briefly introduce the challenges and the typical approach to controlling physical machinery through networks.

A network control system (NCS) consists of at least three entities, i.e., the system to be controlled which is typically called plant, the network, and the controller. The plant is periodically sampled using sensors which in turn transmit the samples data over the network to the controller. The controller then takes these inputs and calculates a control output with regard to a certain goal that is preprogrammed in the controller and sends it back to the plant such that actuators can modify the physical state of the system. As the control output is subject to inaccuracies, these steps are repeated over and over again, therefore, these systems are typically referred to as *feedback* control systems.

The textbook example of such a system is an inverted pendulum as shown in Figure 2 which we will also referrer to in our evaluation (Section 4.1). Here a pendulum on a stiff rod is installed on a cart that can only move in horizontal direction. The goal of this system is to balance the pendulum in an upright position such that it does not fall over or rotate below the center of the cart. Here one would sample the location of the cart, as well as the angle of the pendulum. The goal of control theory is to design optimal controllers that fulfill this goal. Typically, this is a two-stage process. First, the physical system is modeled as closely as possible using mathematical formulas. This system model is then discretized and finally, a controller is synthesized such that it modifies the system to the desired state, e.g., the pendulum being upright and the cart being at a certain position. There are multiple classes of these controllers, e.g., proportional-integral-derivative (PID) controllers, linear-quadratic regulator (LQR) controllers, or even neural networks. The outcome of this process is typically a set of matrices or vectors that enable to calculate a control output given the sampled input data, i.e., in contrast to the control design which can take long time, the second phase is typically only a matrix multiplication. A challenge of these controllers is delay and jitter. Control theory typically regards the network as a black box with a certain delay and jitter. To account for these, the control design is modified such that the controller can deal with the delay; this is typically done by accounting for the maximum of the expected delay and jitter. Yet, this leads to decreased control performance, e.g., it takes longer to stabilize the pendulum.

3 IN-NETWORK FEEDBACK CONTROL

Ever growing demands in flexibility challenge classical control approaches. To this end, control theory evaluates the feasibility of moving control into cloud environments, a challenge that introduces delay and jitter in the communication path. Our approach introduces a middle-ground between local and cloud-facilitated control by pushing some control logic back from the cloud into the local network. This way, we are able to reduce latencies but still utilize the resourcefulness of the cloud.

We utilize the already existing split in control, i.e., we leave the control synthesis in the cloud, this way it can utilize elastic computation power as well as optimize controllers from a plethora of collected measurements (as e.g., in [6]) of the system that is to be controlled. Yet, we outsource the actual control to the network, i.e., we calculate the actual control output directly on-path, e.g., in the switches of the local network saving on latencies, enabling more aggressive controllers that increase the quality of control.

Thereby, we pave the way for more flexible production sites. On the one hand, we can outsource the classic control

problems into resourceful environments, and on the other hand, our approach enables new means of reliability for cloud-facilitated network control. For example, when the uplink to the Internet fails, causing the cloud not to be reachable anymore, the local network can still control machinery. Further, safety-critical emergency routines can be offloaded into the network. Imagine a human interacting with one or more machines. In order to avoid physical injuries, an emergency stop could then be offloaded into the network such that machines can be halted with high reliability when humans are in too close proximity.

We continue by explaining how we implement such a system that can run on real hardware and the problems that come when doing so.

3.1 In-Network Processing for Control

To evaluate the feasibility of in-network control, we offload the control matrix multiplication to network elements. To this end, we target to utilize the extended Berkeley Packet Filter (eBPF). eBPF (BPF for the remainder) is the evolution of the classical Berkeley Packet Filter (cBPF), that is for example at the core of tcpdump's packet filter. BPF is a pseudo-virtual machine instruction set, specifically designed to meet the requirements for efficient packet processing. It is usually just-in-time compiled to the executing machine's instruction set. BPF programs can be injected into various parts of the Linux kernel or even onto smart NICs to process packets. Its applicability has shown tremendous success for variable firewalls [5] or custom forwarding planes [1, 12, 18]. The most promising entry point for BPF programs is the express datapath (XDP) which enables to hook directly into the network card's driver even before packet control structures are allocated thus saving on general purpose processing. Utilizing XDP enables to create custom protocols that are purpose driven for a certain control problem and it further paves the way for transparent hardware offloading.

Yet, instead of reinventing the wheel, we make use of P4 [4] to describe the datapath. P4 offers great flexibility and has already shown its potential for general purpose switching applications (e.g., [8, 11, 15, 16]). To this end, we utilize the XDP-P4 backend [22] to compile a P4 datapath description to C code that in turn can be compiled using clang and LLVM to BPF bytecode. Thus, we can profit from the streamlined P4 dataplane description but are highly flexible in the way we can deploy using BPF. We continue by describing our datapath and especially, how we embed an LQR-controller, a prominent class of optimal controllers, in P4.

3.2 LQR Control in P4

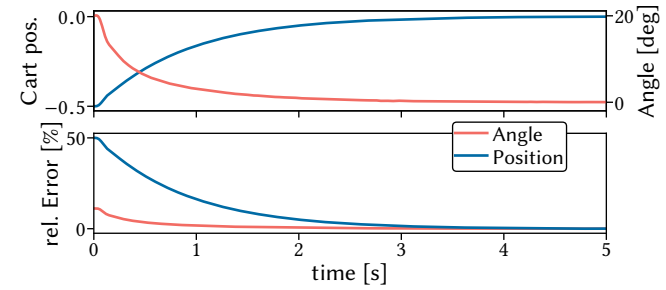
An LQR controller output is rather simple to compute. The whole controller is described by a vector K which is multiplied by the current system state vector x (over the reals or using signed floating-point numbers, respectively). This yields the output u to apply next to the plant:

$$u = -K^T \cdot x$$

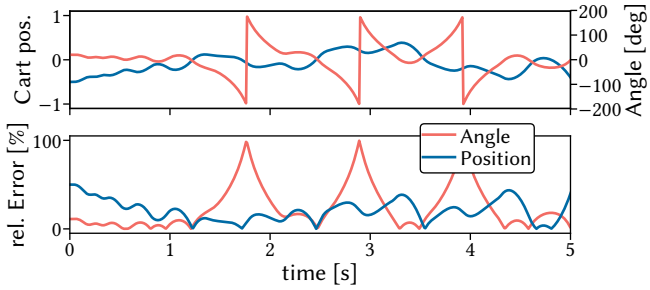
We implement a switch in P4 that is able to perform this simple multiplication for a state vector x that is transmitted in UDP datagrams. To this end, our implementation parses each incoming packet and looks for UDP datagrams of the correct length. In case such a header is found, we apply a P4 table trying to match the source IP and port as well as the destination IP and port, and look up the control vector K in the table, similar to looking up destinations in a routing process. We then calculate the desired LQR output u using K and x and send a packet with the new output back to the plant by swapping source and destination IPs and ports and by replacing the original payload of the packet with u.

As it turns out, this is not necessarily as straight forward as one would imagine. First of all, P4 and also BPF have no support for floating point arithmetics which however are used in many control problems. This is not a real problem as many real-world sensors only operate on integers in the first place and other floating point systems can be transformed to integers using fixed-point arithmetics. Yet, when assuming fixed-point arithmetics, our LQR-control computation effectively scales the output by the square of the fix-point. K and x are both scaled by the fix-point, thus the product of both is scaled by the square of the fix-point. Thus, we are required to divide the output of the multiplication by the fix-point to scale the value to the original fix-point again. However, the most recent P4 standard does not define divisions on signed numbers requiring to first convert negative numbers to positive ones, perform divisions and finally negate them again. Even though BPF would allow signed divisions, we strive for P4 compatibility. Furthermore, multiplying fix-points further requires performing the calculations with larger bit-depth such that integers do not overflow, a requirement that could also not be met on different platforms according to the P4 documentation.

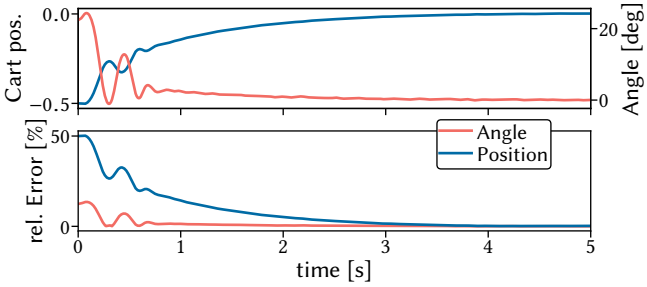
After having transformed the input to an output value we fix up checksums and length fields and we apply another table that looks up destination IPs to Ethernet next hop addresses and we rewrite the MAC header accordingly. Finally, P4 de-parses the packet, yet, our UDP payload changed such that we construct a different header for the UDP payload than that we previously parsed.



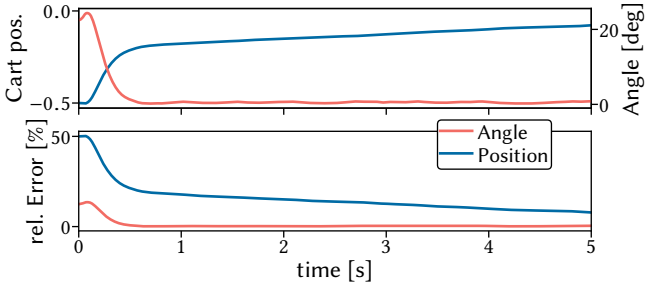
(a) Local control not subject to delay.



(b) Control as in (a) when introducing 5 ms RTT.



(c) Control accounting for 20 ms RTT at 20 ms RTT.



(d) Control accounting for 20 ms RTT and wobbling.

Figure 3: Controlling the pendulum over a network. (a) the control is directly connected to the pendulum. (b) control breaks down when we introduce only 5 ms of RTT. (c) adjusting the controller design to account for delay, and, (d) adjusting the control design while guaranteeing no wobbling.

4 INVERTED PENDULUM IN-NETWORK CONTROL

To investigate the effectiveness of in-network control, we implement a real-time simulation of the inverted pendulum problem (cf. Figure 2). The system state (cart position, first derivative of the cart's position, pendulum angle, first derivative of pendulum angle) is periodically sampled and transmitted using UDP to a controller that is attached to a mininet. We extend mininet such that XDP programs can be attached to the virtual interfaces of a mininet switch. This way, we can easily test our implementation fully virtualized or we only use it to emulate delay and attach the XDP program to a PC equipped with a Netronome Agilio CX to which we can also offload the BPF program[1]. In the end, this results in a simple linear topology, with the XDP-enabled switch connected to the real-time simulation and to the mininet hosting the controller that would normally be executed in the cloud. The XDP-enabled switch provides interfaces to populate the P4 tables, thus by inserting a control vector K with a corresponding flow four-tuple we can activate and deactivate the switch performing LQR-control for certain flows.

[1]Netronome supplied us with an alpha version of their upcoming BPF-capable firmware.

4.1 Evaluation

To investigate the effectiveness of our approach, we must evaluate the control performance when offloading control to in-network elements. Control performance can be evaluated under various aspects. At the example of the pendulum, typical evaluation metrics are: how fast does the controller reach its goal, what is the amplitude of the pendulum when it is moved, does the pendulum oscillate around the equilibrium? To answer these questions, we setup the real-time simulation such that the cart is positioned left to the center ($x = -0.5$) and the pendulum is tilted to the right ($\theta = 20°$) of the upright position ($\theta = 0°$). Thus, when control does not apply a force to the cart moving it to the right, the pendulum will fall over (and will continue rotating below the cart). We setup the LQR controller such that its goal is to balance the pendulum at the center ($x = 0$) in the upright position ($\theta = 0°$). To measure the quality of control that we can achieve, i.e., answering the previous questions, we investigate the distance from the system's state to the desired equilibrium at ($x = 0, \theta = 0°$).

Baseline. To set a baseline, we first evaluate the classic scenario, i.e., the control is local to the pendulum and is not outsourced to a cloud environment. Figure 3a depicts the achievable quality of control. The upper subplot depicts the

absolute difference to the equilibrium for the cart's position (left axis, blue line) and the pendulum's angle (right axis, red line). Below, we show the relative error given the horizontal bound of -1.0 ...1.0 and at most 180° distance from the upright position. We observe that the controller is swift in correcting the angle: After roughly 4 s, controller has reached the equilibrium. Furthermore, we observe no oscillations in the pendulum's angle or the cart's position attesting a smooth control.

Introducing Delay. When we start to outsource the control to a distant controller, e.g., located in some factory cloud or in a datacenter, delay is introduced into the communication path. Even when assuming low latencies on an Internet scale, i.e., through an IXP, with an RTT of 5 ms, the same control starts to break as shown in Figure 3b. When looking at the first second of the control, we can see that the controller is able to move the system towards the equilibrium. Yet, we can already observe that the system starts to oscillate. Continuing, the oscillations—caused by measurements as well as control signals arriving too late—become to severe such that the system becomes completely unstable. Indicated by the angle of the pendulum, the pendulum starts to rotate by 360° applying force (and thereby movement) onto the cart that the controller is unable to manage.

In summary, these first results show the high correlation of delay and control quality.

As stated earlier, control is of course able to account for delays. We incorporate up to 20 ms RTT into the controller design allowing for jitter and delay. As shown in Figure 3c this re-enables control to finally stabilize the pendulum in the upright position. However, we observe that the pendulum is now subject to severe wobbling as indicated by the angle of the pendulum swinging back and forth in the first second. The feedback control is able to stabilize the pendulum by adjusting the force on the cart as indicated by the change in the cart's position. Yet finally, control is again able to reach the equilibrium after roughly 4 s.

However, depending on the actual control goal such a back and forth might be undesired, e.g., imagine a controller governing the transport of liquids without causing disturbance or sloshing. We can however adjust control further to also incorporate this as indicated by Figure 3d. Now, we have eliminated a wobbling, however, as indicated by the relative error to the equilibrium, control had to slow down the cart movement to achieve the desired goal. Thus, from another perspective the quality of control is still suboptimal, e.g., leading to increased production times. While this shows that delay can be incorporated to a certain degree, this incorporation often leads to some sort of control degradation.

Using In-Network Control. However, when we utilize our approach of in-network control, we can restore the original control quality. Figure 4 shows the quality-of-control using

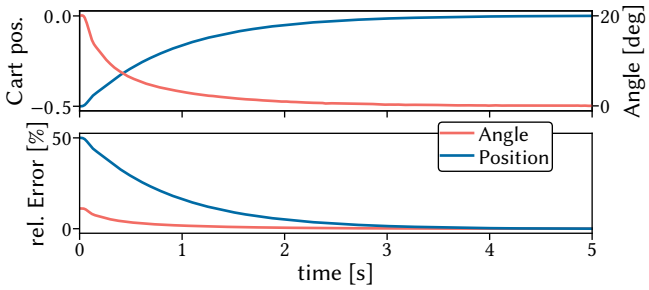


Figure 4: Outsourcing control to the network enables to restore the original quality of control while maintaining central management in the distance.

our XDP-offloaded LQR control. We observe that in-network processing enables to utilize the ultra-low latencies of local communication as the pendulum can be moved into stable position as if we were using a controller directly attached to the pendulum as in Figure 3a. This also highlights that the latencies of locally connected devices that are typically order of magnitude lower than the sampling rates of sensors do not influence the control.

5 FUTURE CHALLENGES

While this work highlights that a common class of control problems can be offloaded to network elements, there are still technical and conceptual challenges. First, we assume that it is rarely required to account for delay when control is outsourced to the local network, yet, it raises some conceptual questions if it is still required. When accounting for delay during the controller design, one has to bloat the state-space to include past control outputs and keep them local to the control process. One has to keep the last n control outputs to account for n times the sampling rate of the sensor data, i.e., to account for 20ms of delay when sampling with 500 Hz, one has to keep the last ten computed values. Currently, P4 is not able to store such values computed on the datapath even though the $P4_{16}$ standard acknowledges that future versions *might* support this. In contrast, BPF programs can store these values, even though when offloaded to our SmartNIC, storing values from the datapath is also currently *not* possible, future version are planned to include atomic add operations to store statistics, yet these would not be enough.

When looking at other control problems, P4 and BPF can be further limiting: P4 first completely parses a packet to memory, in case of BPF, this is challenging as the BPF stack is limited to 512 byte which is also shared with other local variables. Assuming, e.g., audio data capturing machine vibrations on which signal detection and subsequent actuation should be performed, 1 ms of 16-bit PCM audio is already 88 byte, several milliseconds of audio can easily overflow

BPF's stack. Visual processing, e.g., tracking or detecting objects, even in ultra-low resolution images might require both, per flow storage that outlives a single packet, as well as being able to process larger packets.

6 RELATED WORK

Utilizing cloud computing in industrial automation has been proposed before, e.g., [20, 21, 23, 24]. Most works target higher level functionality such as monitoring, configuration and maintenance, not the latency-critical control- and field level [9]. Jitter and delay in cloud controller settings have gained further attention [7, 13], with the [13] suggesting an adaption of the controllers or state prediction techniques for mitigation. We, in contrast, analyze and partially offload existing controllers into network hardware. By utilizing SDN- and packet processing-specific languages (P4 and BPF), our approach reduces jitter and delay to a minimum *by design*.

Our approach relates to the edge computing paradigm in which computations are also offloaded [14, 19]. In line with aforementioned works, the large-scale survey [3] reveals only little concrete work regarding edge computing for industrial control. Hence, to the best of our knowledge, our work is among the first to adapt the concept of SDN and in-network processing and apply them to challenges in automation and control.

7 CONCLUSION

This paper introduces networked control as a new application domain for in-network processing. By combining control and communication, a new flexibility for automation and control is introduced that enables to outsource and thereby optimize control to distant, resourceful environments. Our implementation of control and communication in P4 highlights the applicability of today's technologies. However, we also show its limitations for more involved problems that demand more than is currently offered. Further, by synthesizing P4 to BPF running in XDP on SmartNICs or general purpose hardware we further gain flexibility in deployment. Our evaluation highlights how control is challenged by even small amounts of delay and that in-network processing is a solution worth exploring.

REFERENCES

[1] Cilium Authors. 2018. Cilium — API-aware Networking and Security for Containers based on BPF. https://github.com/cilium/cilium

[2] J. Baillieul and P. J. Antsaklis. 2007. Control and Communication Challenges in Networked Real-Time Systems. *Proc. IEEE* 95, 1 (2007), 9–28.

[3] A. C. Baktir, A. Ozgovde, and C. Ersoy. 2017. How Can Edge Computing Benefit From Software-Defined Networking: A Survey, Use Cases, and Future Directions. *IEEE Communications Surveys Tutorials* 19, 4 (2017), 2359–2391.

[4] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM CCR* 44, 3 (2014), 87–95.

[5] Cloudflare. 2016. BPF Tools - packet analyst toolkit. https://github.com/cloudflare/bpftools

[6] Andreas Doerr, Christian Daniel, Duy Nguyen-Tuong, Alonso Marco, Stefan Schaal, Marc Toussaint, and Sebastian Trimpe. 2017. Optimizing Long-term Predictions for Model-based Policy Search. In *Proceedings of Machine Learning Research*.

[7] P. Ferrari, E. Sisinni, D. Brandão, and M. Rocha. 2017. Evaluation of communication latency in Industrial IoT applications. In *IEEE International Workshop on Measurement and Networking (M&N)*.

[8] Mojgan Ghasemi, Theophilus Benson, and Jennifer Rexford. 2017. Dapper: Data Plane Performance Diagnosis of TCP. In *ACM SOSR*.

[9] O. Givehchi, H. Trsek, and J. Jasperneite. 2013. Cloud Computing for Industrial Automation Systems – A Comprehensive Overview. In *IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*.

[10] D. Goswami, R. Schneider, and S. Chakraborty. 2011. Co-design of Cyber-Physical Systems via Controllers with Flexible Delay Constraints. In *16th Asia and South Pacific Design Automation Conference*.

[11] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. 2016. HULA: Scalable Load Balancing Using Programmable Data Planes. In *ACM SOSR*.

[12] Petr Lapukhov. 2016. Internet-scale Virtual Networking with ILA. NANOG 68 Presentation.

[13] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandström, and M. Behnam. 2017. Delay Mitigation in Offloaded Cloud Controllers in Industrial IoT. *IEEE Access* 5 (2017), 4418–4430.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[15] Anirudh Sivaraman, Changhoon Kim, Ramkumar Krishnamoorthy, Advait Dixit, and Mihai Budiu. 2015. DC.P4: Programming the Forwarding Plane of a Data-center Switch. In *ACM SOSR*.

[16] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. 2017. Heavy-Hitter Detection Entirely in the Data Plane. In *ACM SOSR*.

[17] H. Takatsu and T. Itoh. 1999. Future Needs for Control Theory in Industry—Report of the Control Technology Survey in Japanese Industry. *IEEE Transactions on Control Systems Technology* 7, 3 (1999), 298–305.

[18] Cheng-Chun Tu, Joe Stringer, and Justin Pettit. 2017. Building an Extensible Open vSwitch Datapath. *ACM SIGOPS Oper. Syst. Rev.* 51, 1 (2017), 72–77.

[19] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. 2016. Challenges and Opportunities in Edge Computing. In *IEEE International Conference on Smart Cloud (SmartCloud)*.

[20] Alexander Verl, Armin Lechler, Stefan Wesner, Andreas Kirstädter, Jan Schlechtendahl, Lutz Schubert, and Sebastian Meier. 2013. An Approach for a Cloud-based Machine Tool Control. *Procedia CIRP* 7 (2013), 682 – 687.

[21] A. Vick, C. Horn, M. Rudorfer, and J. Krüger. 2015. Control of Robots and Machine Tools with an Extended Factory Cloud. In *IEEE World Conference on Factory Communication Systems (WFCS)*.

[22] VMware. 2018. Backend for the P4 compiler targeting XDP. https://github.com/vmware/p4c-xdp

[23] H. Wu, L. Lou, C. C. Chen, S. Hirche, and K. Kuhnlenz. 2013. Cloud-Based Networked Visual Servo Control. *IEEE Transactions on Industrial Electronics* 60, 2 (2013), 554–566.

[24] Xun Xu. 2012. From Cloud Computing to Cloud Manufacturing. *Robotics and Computer-Integrated Manufacturing* 28, 1 (2012), 75–86.

[25] Wei Zhang, M. S. Branicky, and S. M. Phillips. 2001. Stability of Networked Control Systems. *IEEE Control Systems* 21, 1 (2001), 84–99.