Demo Abstract: Towards In-Network Processing for Low-Latency Industrial Control

Jan Rüth, René Glebke, Tanja Ulmen, Klaus Wehrle
Communication and Distributed Systems, RWTH Aachen University
{rueth,glebke,ulmen,wehrle}@comsys.rwth-aachen.de

***Abstract*—Traditional Networked Control Systems control physical machinery in factories and production environments via centralized on-site network controllers. This architecture is increasingly challenged by rising amounts of sensory and actuator control data and reconfigurable, highly dynamic production sites. A recent trend in Network Control tries to move control algorithms towards remote environments such as the cloud rather than closer to the controlled machinery. While this may satisfy the growing demand for computation resources in industrial settings, simply moving logic into the cloud introduces unpredictable latency and jitter that may break intricate control loops. In this demo, we present an approach that offloads latency-critical parts of the control logic to in-network elements. We combine the P4 network programming language with industrial control algorithms and execute both in eBPF virtual environments on the in-network elements. This technique enables to lower latency and jitter in production environments, providing a significantly higher quality of control even in adverse conditions.**

I. Introduction

For years, communication research has focused on provisioning ever more bandwidth to meet the growing demands of users and applications. However, the recent advent of Cyber-Physical Systems (CPSs) has brought a set of requirements into focus that traditional communication and control architectures cannot meet: Modern production sites are envisioned to consist of a large number of sensors and actuators that generate context information and manipulate the physical environment. Software-based controllers running in cloud entities then orchestrate the operation of the components via networks. In such scenarios, devices and networks have to handle ever-increasing amounts of data and may change their configuration and requirements frequently during operation. On the other hand, controlling physical processes via Networked Control Systems (NCSs) requires a verifiable and highly reliable behavior of all components. High latencies or jitter in the signaling path between sensors and controllers can cause the control algorithms to work on data which is already outdated the moment it arrives, yielding false assumptions about the current status of a so-called System Under Control (SUC) [1]. Likewise, actuator control signals from controllers have to arrive in a timely and reliable fashion at the SUC to guarantee the desired operations to be fulfilled and to prevent physical damage in emergency situations.

Network latency and jitter are challenged by uncertainties in packet processing in two regards. First, the traditional abstraction between layers adds a *vertical overhead* as every signal needs to traverse the network stacks of the affected devices. Second, each hop that a signal needs to take in a network adds a natural *horizontal overhead* by physical distance and queuing on the intermediary network elements.

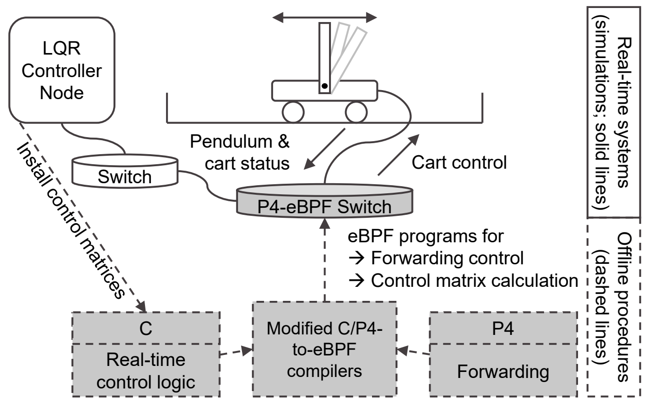


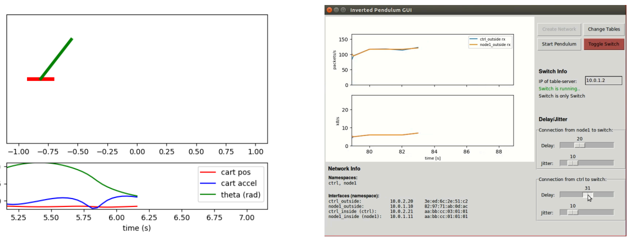
Fig. 1. Overview of our system demo. We compile P4 forwarding rules and a C version of the online control algorithm for an inverted pendulum to an eBPF program and install it as a switch. The switch then controls the pendulum in real-time. Our contributions are marked in gray.

In this demo[1], we show how we can reduce both vertical and horizontal overheads in NCSs with remote (cloud) controllers by partly removing the abstraction induced by the layering concept and by moving control functionality into the network. Our approach is to combine switching functionality expressed using the P4 network programming language [2] with industrial control algorithms and executing both in Extended Berkeley Packet Filter (eBPF) virtual environments [3] on in-network elements. We explain our approach in Section II and then discuss our demo setup and first evaluation results in Section III, before concluding in Section IV.

II. Approach and Example Use-Case

We explain our approach for reducing horizontal and vertical overhead along Figure 1. The top part of the figure shows an exemplary NCS, consisting of at least one controller node, a network of several switches, and at least one SUC, in our example case, an *inverted pendulum*. The goal is keeping the pendulum in an upright position by moving the cart on which it is mounted. For this, information on the status of the pendulum and the cart are sent to the controller node via the switched network in the form of a *state vector*. The controller node runs an instance of a two-phase control algorithm. The first, computationally heavy phase is offline and uses the physical

[1]Video: https://www.comsys.rwth-aachen.de/short/infocom18-demo



Switch Info
IP of table-server: 10.0.1.2
Switch is running..
Switch is only Switch

Delay/Jitter
Connection from node1 to switch:
Delay: 20
Jitter: 10
Connection from ctrl to switch:
Delay: 31
Jitter: 10

Network Info
Namespaces:
ctrl, node1

Interfaces (namespace):
```
ctrl_outside:           10.0.2.20    3e:ed:6c:2e:51:c2
node1_outside:          10.0.1.10    82:97:71:ab:0d:ac
ctrl_inside (ctrl):     10.0.2.21    aa:bb:cc:03:01:01
node1_inside (node1):   10.0.1.11    aa:bb:cc:01:01:01
```

Fig. 2. Screenshot of the pendulum and network control visible in the demo.

properties of the SUC to derive a Linear Quadratic Regulator (LQR) expressed as control matrices. In the second phase, these matrices are used and multiplied with the state vector yielding a cart control vector that is then sent back to the cart to counteract the tilting of the pendulum in real-time.

The lower and gray parts of Fig. 1 show our modifications to the initial setup in order to counteract the effects of the two kinds of overheads. Our core idea is the following: We leverage the control design's split and leave the computationally heavy first phase in the resourceful cloud but outsource the second phase to in-network elements. In our example, we move the LQR's matrix multiplication. Once moved, we instruct the network to redirect all required inputs (the state vector) through this network element while leaving the remainder of the information flow between SUC and the controller unchanged.

In our implementation, we use the P4 network programming language [2] to express forwarding logic (bottom right of Fig. 1). While P4 is well suited for forwarding, we found that it is not expressive enough to handle various control functions. Hence, in order to express the LQR's online phase, we combine P4 forwarding with a C version of the matrix multiplication by compiling both to a single eBPF program that is flexibly configurable with control matrices further allowing to selectively apply the control matrices to matching traffic. eBPF execution environments are currently available for Linux and recently, programmable network hardware [4]. Once installed on a capable in-network element, the controller can install matrices in the network element similar to the network controller installing forwarding rules. The eBPF program then applies the LQR control to configured and matching flows and immediately sends new control instructions back to the cart.

We next give a description of our demo setup and show the benefits of our approach by providing first experimental results for our inverted pendulum.

III. Demo Setup and Initial Results

Our demo (see Fig. 2) modifies a freely available simulation of an inverted pendulum [5] as our SUC. Afterward, we use Mininet [6] to instantiate a virtual network on our demonstration machine that resembles the network depicted in Fig. 1. Once the matrices are generated, we install them in the eBPF network element through its southbound API. The unmodified pendulum controller runs as one process on our machine and communicates with the pendulum simulation in another process through a UDP traversing the new virtual network. Our demo allows to flexibly add delay and jitter on the link between the two switches using *netem* (see Fig. 2). The blue line in Fig. 3 shows the effect of the overhead to the quality of control in terms of the mean *force error*, i.e. the difference between the desired force exerted on the cart as calculated by an optimal control and the actual force (in percent). The light area around the line shows the standard deviation of the force error over 30 repeated runs. The system constantly oscillates and the controller is unable to bring the pendulum into an upright position. The lower red line in Fig. 3 shows the new quality of control that we can achieve using our approach: Since the real-time critical part is now executed on the gray switch, the costly hop on the link between the switches on the path to the controller node can be omitted and the pendulum is stabilized after roughly 0.5 s, with no noticeable deviations. Since we left the remainder of the communication between controller and SUC intact, non real-time critical functionality of the controller can still be executed on the controller node, thus not impairing the general functionality of the controller or the network.

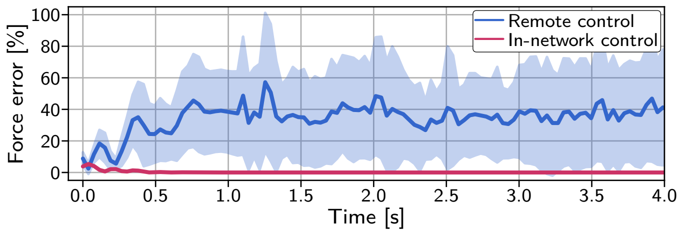


Fig. 3. Force error and standard deviation (area) in our demo setup (Fig. 1). Without in-network control, the system cannot be stabilized. With in-network control activated on the gray switch, the system stabilizes after roughly 0.5 s.

IV. Conclusion and Future Work

Our demo shows that by moving real-time critical parts of machine controllers to in-network elements for execution as eBPF programs, we can significantly reduce the impact of latency and jitter that signals to and from the controller experience. This results in a significantly higher quality of control in conditions which are likely to occur when parts of the controllers are offloaded to remote environments. As future work, we would like to study the applicability of our approach to more difficult problems than our inverted pendulum example, as well as the possibility of investigating the impact of feedback mechanisms between the offloaded parts of the control algorithms and the remaining controller nodes.

Acknowledgment

This work has been partially supported by DFG within Priority Programme 1914 Project REFLEXES.

References

[1] Goldschmidt et al., "Cloud-Based Control: A Multi-tenant, Horizontally Scalable Soft-PLC," in *IEEE CLOUD*, 2015.
[2] Bosshart et al., "P4: Programming Protocol-independent Packet Processors," *SIGCOMM CCR*, Jul. 2014.
[3] Steven McCanne and Van Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in *USENIX*, 1993.
[4] Netronome, "Agilio CX SmartNICs." [Online]. Available: https://www.netronome.com/products/agilio-cx/
[5] Todd Sifleet, "Inverted Pendulum." [Online]. Available: http://www.toddsifleet.com/projects/inverted-pendulum
[6] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *SIGCOMM Hotnets*, 2010.