Towards Low Latency Industrial Robot Control in Programmable Data Planes

Fabricio E Rodriguez Cesen*, Levente Csikor†, Carlos Recalde*, Christian Esteve Rothenberg* and Gergely Pongrácz‡
*University of Campinas (UNICAMP), Sao Paulo, Brazil
‡Ericsson Research, Budapest, Hungary
†National University of Singapore, Singapore

***Abstract*—Due to the advanced control and machine learning techniques, today's industrial robots are faster and more accurate than human workers in well-structured repetitive tasks. However, in case of sudden changes in the operational area, such as unexpected obstacles or humans, robots have to be continuously monitored by powerful controllers for swift interventions (i.e., send emergency stop signals). As in the case of many verticals (e.g., transportation, shopping), the proliferation of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) has started to captivate industry 4.0 as well in order to benefit from low infrastructure costs, and flexible management and resource provisioning. Besides all the advantages of the centralized approach, however, in critical situations (e.g., possible collisions, actuator damages or human injuries) the required ultra-low latency between the robots and the controller becomes an all-important factor, and one of the main concerns, at the same time, for industry leaders making the decision towards this paradigm shift.**

In this paper, we argue that by relying on recently emerged stateful and programmable data planes, it is possible to fill this gap by offloading latency-critical applications to the network, thereby bringing some intelligence much closer the robots. We present the first in-network robotic control application that is capable to intercept the communication between the robot and the controller and craft responses immediately if needed. In particular, we show that we can detect position threshold violations entirely in the data plane, close to the robot, and deliver emergency stop commands within no time with full compliance to the actual TCP session and application states.

***Index Terms*—P4, Low Latency applications, Robotic control**

I. Introduction

Nowadays, industrial robots with high degree-of-freedom (DOF) are more intelligent than their predecessors decades ago. Advanced control and machine learning techniques have enabled the robots to be faster and more accurate than human workers in well-structured repetitive tasks. Robots can calculate their own trajectories between two instructed positions. Furthermore, by having external sensors (e.g., camera, motion sensor) robots can observe sudden changes in the operational area (i.e., in the robotic cell) such as unexpected obstacles or even humans. In such critical circumstances, not reacting in time (e.g., sending emergency stop signals, adjusting movements) can entail a severe impact including collision, actuator damages or, more importantly, human injuries. In order to resolve this, each robot is directly connected to a powerful controller for continuous monitoring, and swift and precise interventions (see Fig. 1).

The evolution of Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have already revolutionized many industries (e.g., IT, transportation, shopping) resulting in that enterprises increasingly offload their business-critical workloads to the (public) cloud to benefit from low infrastructure costs, high availability and flexible resource management. This booming paradigm shift has recently started to captivate further verticals, including industry 4.0 [1] and robotic control enabling each individual robot controller to be centralized (or even offloaded to the cloud) to significantly improve operational management (e.g., controller programming, firmware upgrades) and decrease capital expenditures. However, have to face with the unreliable (i.e., lossy and congested links, variation in delays) nature of today's network.

While the proliferation of 5G provides improved network capacity and ultra-low latency, due to interference, backscattering, or even jamming, wireless signals can still be undeterministic disabling 5G alone to be feasible for latency-critical applications such as robotic control. Particularly, if a stop message generated by the centralized controller suffers from delays or even packet losses, there is going to be an error between the intended and actual position, potentially causing serious hazards mentioned above (cf. Fig. 4).

Recently emerged programmable and stateful networks, on the other hand, have given rise for in-network solutions (e.g., [2]–[4]) enabling simple calculations to be offloaded to the network itself. By enabling some tasks (i.e., requests) to be resolved entirely in the data plane (inherently, much closer to the requester) without practically reaching the centralized controller, we can significantly reduce latency and reaction time without the need to overcome possible congestion and delays in the core of the network. However, in order to realize a fully in-network solution, we have to properly maintain the underlying TCP session and application states.

In this paper, we take the first steps towards this direction and investigate whether the industrial robot controllers can benefit from the revolutionary network paradigm shift by overcoming the possible pitfalls via an in-network solution. In particular, we present the first in-network robotic control

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A., Brazil under grant agreement UNI.66.

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

application (i.e., offloaded to the programmable data planes) that is capable to react (i.e., respond to the robot) on sudden changes in the robotic cells. To reach this end, we leverage the key capabilities of network programmability, and we design and implement custom functions to parse and analyze the TCP communication between the robot and the controller, and perform basic calculations (e.g., distance, average filtering) in-network[1]. Particularly, when we detect a position threshold violation in the data plane (by intercepting the continuous status messages sent by the robots to the controller), a custom reply packet is crafted at the first hop in the network to deliver an emergency stop command to the robot arm within no time.

To simulate the robot arm, we use Universal Robots Simulation (URSim)[2], a widely used simulator tool for programming and manually controlling the movement of high DOF robotic arms. To intercept, analyze and manipulate the unencrypted TCP communication between URSim and a controller in the data plane, we prototype our solution in P4 [6]. We show that our publicly available application[3] can efficiently manipulate the packets within a TCP session in fully compliance to the actual TCP session and application state.

II. Background and Related Work

A. Programmable Data Planes

Lately, many solutions have been proposed to make the network programmable and stateful. Among them, P4 [6] became the most attractive solution. By being a domain-specific language with features that helps to disaggregate the network stack, P4 provides a high level of networking abstraction and supports target-independent implementations. The P4 pipeline includes a parser, *match+action* tables, and a deparser; packet headers are parsed upon arrival, then processed in a multi-table pipeline, and deparsed to be finally forwarded. The main advantage of P4 is that due to its constraints and limited instructions set, all P4 application is guaranteed to run in wire-speed irrespective to the underlying P4-enabled hardware. With the use of table lookups, different hardware-independent packet processing algorithms can be implemented. It is possible, furthermore, to create custom matches to perform various actions, such as modifying the packet fields or generate new messages. Moreover, the processing algorithms can be optimized by means of resource mapping and management capabilities (e.g., allocation and scheduling).

B. Universal Robots Simulator

Universal Robots (UR) provides a free simulator called URSim to explore different robot arm configurations in an open environment. URSim is a virtual representation of a real robot arm and the simulator can work as a client device to establish a communication that allows a robot to interact with external devices (e.g., a controller) using TCP sockets[4].

[1]The main idea and proof of concept were first introduced in [5]

[2]https://www.universal-robots.com

[3]https://github.com/ecwolf/p4_robot.git

[4]https://shar.es/a3Z9MR

C. Robot Arm

The UR robot arm has six joints that can rotate 360 degrees. URSim defines three types of movements: joint space, linear, and circular blends (i.e., `movej`, `movel`, `movep`)[5]. Each movement command has a specific structure. Listing 1 presents the `movej` message, including the message sent to the robot.

```
1  movej([Base, Shoulder, Elbow, Wrist1, Wrist2,
2         Wrist3], a=acceleration, v=speed)
3  movej([1.545, −2.35, −1.31, −2.27, 3.358,
4         −1.22], a=4, v=5)
5  p[1.545, −2.35, −1.31, −2.27, 3.358,
6   −1.22, 4, 5]
```

Listing 1: URSim movej message.

D. Robot Controller

The robot controller can be a physical device or a programmed script. After establishing the connection with the robot, the controller can send commands to the arm, e.g., setting positions and actions. At the same time, the robot is constantly sharing its actual position through the bidirectional TCP channel.

E. Related Work

Recent research in the literature aims at providing in-network action developments [7]. Some authors have driven the development of offloading cloud actions to local networks. Jan Rüth et al. [8] offload from the cloud to a local network box, tasks of control and communication deploying in P4. Angelo Lapolli et al. [4] implement a sophisticated security logic on the data plane device. René Glebke et al. [9] bring computer vision tasks to the network by deploying a P4-based system capable of identifying patterns in images and performing actions accordingly. B. G. Nagy et al [10] evaluate an environment with robot arms and a human collaboration using a Virtual/Augmented Reality (VR/AR) application. They planted a solution using cloud-native programs and CPU core allocation demonstrating the impact of delays during the deployment.

A similar approach came from the Mobile Edge Computing (MEC) architectures bringing cloud-computing capabilities to the network edge [11]. Arising from a MEC approach [12], if the robot actions are programmed in the P4 switch, we can reduce the processing of sending information to the MEC, and having a direct response from the edge device (P4 switch).

III. System Overview

As mentioned in Sec. I, the main goal of our in-network robot arm control is to investigate the potential of a programmable data plane (e.g., P4-based device) deployed close to the target robot components (e.g., arm, sensor) in a centralized network with undeterministic connections. Particularly, we explore how to reduce the latency of critical actions (e.g., emergency stop, alarm notification, synchronization) when needed.

[5]https://www.zacobria.com/

Fig. 1: Traditional robot control.

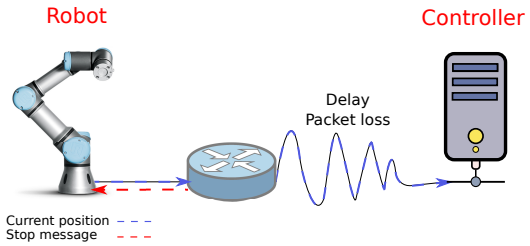


Fig. 2: System overview.

Accordingly, the system being investigated encloses a robot and a controller, where the robot arm is programmed to do the well-structured repetitive tasks. In contrast, the controller role encloses activities of verification, failure response, synchronization process, etc. Figure 2 presents a high-level overview of the system architecture and the experimental setup. On the left-hand side, the robot (URSim) executes the commands received from the controller, and sends its current position and further status messages back to the controller. The robot is configured with a start position, velocity, acceleration, and step size for its movements (Listing 2).

On the right-hand side, the controller (which we implemented in Python) establishes a TCP communication to the robot and analyzes all incoming messages afterward. The controller is configured to send a stop message to the robot when it violates a threshold position. Between the controller and the robot, a P4 device is used to forward the corresponding traffic in both directions and to analyze (parses + table matches) certain parts of the payload to obtain the necessary information (i.e., actual position).

```
Loop var_1 ?= False
  var_1 ?= socket_open("10.1.1.27",30000)
movej([1.545, -2.35, -1.31, -2.27, 3.358,
    -1.22], a=1.4, v=1)
var_6 ?= 1.545
Loop var_8 ≠ 0
  movej([var_6, -2.35, -1.31, -2.27, 3.358,
    -1.22], a=4, v=5)
  joint_position ?= get_actual_joint_positions()
  socket_send_string(joint_position)
  var_7 ?= socket_read_ascii_float(1,t=0.003)
  var_6 ?= var_6+0.01
  If var_7[0] ?= 0
    var_8 ?= 1
  Else var_8 ?= var_7[1]
```

Listing 2: Robot Script.

tcp_match.payload	Action	Data
2.063	tcp_payload_match	(0)

Fig. 3: P4 table match parameter

To realize the offloaded in-network control application in P4, we define a custom header to identify the messages from the robot (Listing 3) following the `movej` data structure (Listing 1).

```
header tcp_match {
    bit<8> robot_msg;
    bit<40> payload;
    bit<240> payload_end;  }
```

Listing 3: Robot Header.

After parsing the message and extracting the payload content, the match+action pipeline starts. One of the tables (Listing 4 and Fig. 3) is defined to *match* a specific value (`tcp_match.payload`) and take the corresponding actions (Listing 5). For instance, we can use the detection of a specific robot position as a trigger to craft a new packet and send it back to the robot with a new payload containing a "magic word" command to stop the robot movement immediately.

```
table tcp_exact {
    key = { hdr.tcp_match.payload : exact;  }
    actions = { tcp_payload_match; NoAction; }}
```

Listing 4: Robot TCP match table.

```
action tcp_payload_match(bit<8> robot_msg, bit
    <40> payload_match, bit<9> port) {
    hdr.tcp_match.robot_msg = robot_msg;    // (
    hdr.tcp_match.payload = payload_match; // 0)
    meta.modified = 0x1;
    standard_metadata.egress_spec = port; }
```

Listing 5: Robot Action.

A. Avoiding the link to the controller

Since the connection to the controller can suffer from external factors (e.g., delay, packet loss), they can affect the correct behavior of an application with ultra-low latency (ULL) requirements.

Figure 4 presents a traditional scenario where the robot controller handles the robot actions in the traditional way. In this scenario, the network has different unreliable properties (e.g., delay, packet loss) that affect all messages with ULL actions (e.g., emergency stop, synchronization). The robot will send its current position ❶ to the controller. If this position value reaches a defined threshold (in the controller), the controller generates a stop message ❷. When the stop message passes through, it suffers from a delay that highly affects the time of the instruction (i.e., the stop message). This results in errors measured as the difference from the threshold to the actual stop position, potentially causing a collision or further hazards.

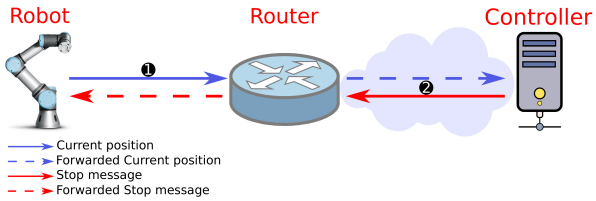


Fig. 4: Traditional scenario without in-network control.

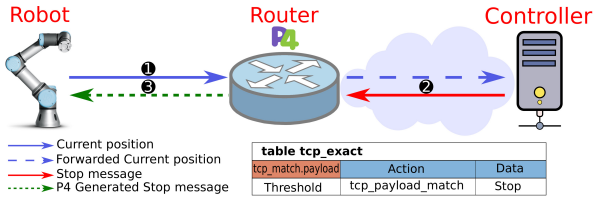


Fig. 5: In-network P4-based implementation.

B. Switch-in-the-middle TCP handling

One challenge of in-network solutions arises from messing up the TCP session. If we manipulate or create any message by the P4 device, we need to sync all the acknowledgment (ACK) and sequence (SEQ) numbers adequately to keep all TCP parameters valid for both end-points.

When the robot sends a message with its current position (Fig. 5) ❶ and it matches with a defined threshold, the P4 router generates an automatic replay with a calculated ACK and SEQ numbers to the robot ❸, without any interaction of the controller. If the robot controller is not informed about this interaction, the synchronization with the robot will fail. The ACK and SEQ numbers are not going to be in line with the robot and the controller, causing a TCP disconnection.

To handle the TCP synchronization problem, alternative strategies can be taken in the P4 program. One option is to update and synchronize the ACK and SEQ numbers of future packets from the controller. To this end, it is necessary to maintain registers in P4 to update these values in the future. Keeping a correct value in the registers can be a complicated process and eventually can fail.

In our implementation (Fig. 5), we use an approach similar to TCP veto [13] (see Fig. 6). If the robot sends a position message ❶ that matches with the predefined threshold, the P4 device will forward the message to the robot controller and, at the same time, it will create a new message to the robot with valid ACK and SEQ numbers ❸. Thereby, when the controller sends the stop message ❷, it will be discarded by the router as duplicates, with synced ACK and SEQ numbers.

IV. Experimental Evaluation

In this section, we present experimental results of the scenario described in Fig. 5. The main objective beyond functional validation is to evaluate the error in the position when the stop message to the robot is generated in the network (delay near to 0) compared to a traditional controller approach subjected to longer and/or variable delays (i.e., 5 ms to 100 ms). Note, the delay components include the propagation (around 5 ns per meter) plus the transmission, queuing, and processing along the path and the controller SW/HW.

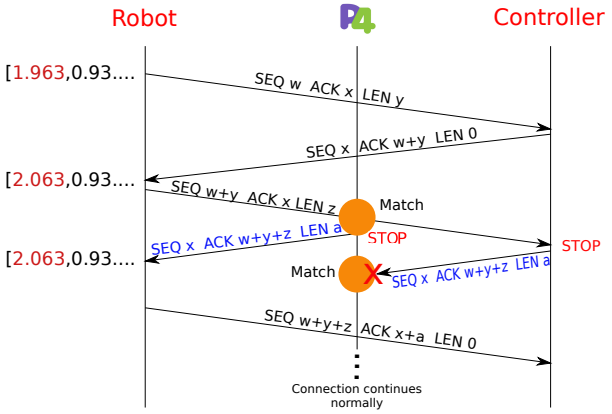


Fig. 6: TCP session approach.

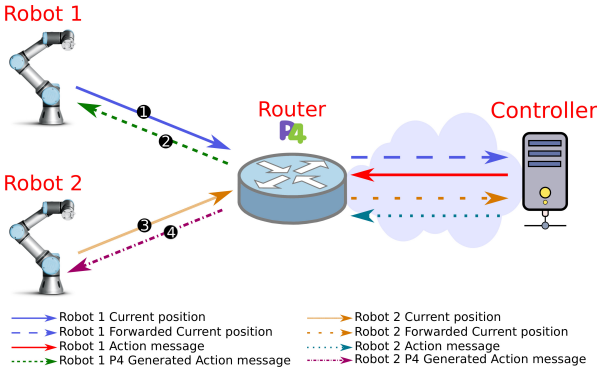


Fig. 7: In-network robot control collision control use case.

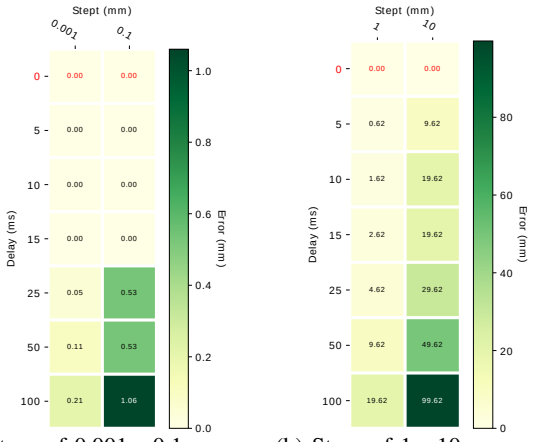
The testbed includes three servers with Intel Xeon E5-2620v2, dual-port 10G Intel X540-AT2 NIC, and 64GB of memory running Ubuntu 16.04. The first server (Robot) is running URSim v3.12.0 with a basic script (Listing 2) to send the position in real-time. The second server acts as a Router and runs the P4 program using BMv2. The third server (Controller) executes a python based script to establish the TCP session and process the information from the robot. For the link between the Router and the Controller, we use Linux Traffic control (tc)[6] to set different delays for the experiments.

We define different delays (5 ms to 100 ms) to evaluate the performance in terms of error (mm) from the target threshold position to the actual stop position. We also explore the impact of the movement length (step of 0.001 mm to 10 mm). We compare the results with the P4 router (0 ms for illustration purposes) carrying out the in-network actions.

Figure 8 presents the results of the experiments. As expected, the delay, and step size affect the error to stop the robot. The color scale used in Fig. 8 correlates darker green with higher errors. In order to minimize the error, the link delay needs to be reduced as much as possible.

The results from the in-network approach (Fig. 5) correspond to the top entries for delay values of 0 ms (red label), since the propagation latency (5 ns every 10 meters) between the robot and the P4 device is negligible compared to the ms

[6]http://man7.org/linux/man-pages/man8/tc.8.html



(a) Steps of 0.001 - 0.1 mm (b) Steps of 1 - 10 mm

Fig. 8: Stop position error without in-network actions. Acceleration of $(30°/s^2)$.

orders of magnitude. Likewise, the line-rate processing time of an actual hardware P4 pipeline would be even more negligible.

Altogether, our experimental findings suggest that an in-network approach can play a critical role in ULL applications. The proof of concept evaluation using a P4 router to send rapid actions to the arm demonstrates that programmable devices could unlock new ways to offload robot controller actions to the network. We also confirm that with P4, we can effectively match payload information from the robot and use it to craft valid in-network packets acting as fast controller actions.

V. Conclusions and Future Work

In industry 4.0 applications, it is common to have more than one device working in synchronization, and their information must be shared. We believe that apart from looking for use cases with one robot, future research should look for a scenario where two or more robots are working nearby in a shared space. If one of the robots is in a collision course with others (including objects or humans detected by collaborative sensors), it is necessary to act rapidly. Programmable network devices may react to such events and send a message of a new course or even a stop action. These activities are required within a short time to effectively prevent possible accidents.

Considering the scenario described in Fig. 7, with two robots working closely. Robot 1 is sending its current position ❶ to the network, as well as Robot 2 ❸. The P4 switch can be programmed to perform actions to synchronize the robots or to prevent any collision (without the intervention of the robot controller). If it is necessary to send any message to one of the robots, the P4 switch can craft them (❷ or ❹) and synchronize with the controller. In such cases, it is important to consider that the P4 switch needs to perform some calculations (e.g, distance, area), thus we have to be aware of the limitations of the underlying hardware that may require the use of external functions.

Overall, we presented the design a P4 implementation of in-network actions to control a robotic arm by offloading some controller functions to a programmable edge network device itself. We described scenarios where the use of a programmable device makes the difference in terms of ultra-low latency response. We showed how to effectively manipulate the content of the messages to craft new replies within an established TCP session. Our experimental evaluation demonstrated the impact of the delay in the connection to an accuracy stop position. The kinematic configuration of the robot (i.e., acceleration, step size) also affects the error (stop position difference) to apply an action.

Programmable data planes in delay critical scenarios open up new opportunities for a different range of applications (e.g., sensors monitoring, data filtering, thresholds matching).

The realization of network actions of the multiple robot synchronization is still in progress. Currently, we are implementing a testbed using a hardware P4 device and a real robotic arm. We are also investigating how to parse messages of URSim using nested package structures as well as additional robot control protocols and real-time UDP sockets.

All in all, our P4-based robotic arm control experience suggests intriguing opportunities that are not limited to robots only but are applicable to a wide range of applications (e.g., industry 4.0, automation, real-time control, synchronization) where low and deterministic latency is paramount.

References

[1] K. Zhou, T. Liu, and L. Zhou, "Industry 4.0: Towards future industrial opportunities and challenges," in *2015 12th International FSKD*. Zhangjiajie, China: IEEE, Aug 2015, pp. 2147–2152.

[2] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings SIGCOMM*, ACM. CA, USA: ACM, 2017, pp. 15–28.

[3] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of SOSR*, ser. SOSR '17. ACM, 2017, p. 164–176.

[4] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on IM*. VA, USA: IEEE, April 2019, pp. 19–27.

[5] F. Rodriguez, C. E. Rothenberg, and G. Pongrácz, "In-network p4-based low latency robot arm control," in *Proceedings CoNEXT*, ser. CoNEXT '19. NY, USA: ACM, 2019, p. 59–61.

[6] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[7] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, "In-network computation is a dumb idea whose time has come," in *Proceedings HotNets*, ser. HotNets-XVI. NY, USA: ACM, 2017, pp. 150–156.

[8] J. Rüth, R. Glebke, K. Wehrle, V. Causevic, and S. Hirche, "Towards in-network industrial feedback control," in *Proceedings of NetCompute*, ser. NetCompute '18. New York, NY, USA: ACM, 2018, pp. 14–19.

[9] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, and K. Wehrle, "Towards executing computer vision functionality on programmable network devices," in *1st ACM CoNEXT Workshop ENCP*, 2019.

[10] B. G. Nagy, J. Dóka, S. Rácz, G. Szabó, I. Pelle, J. Czentye, L. Toka, and B. Sonkoly, "Towards human-robot collaboration: An industry 4.0 vr platform with clouds under the hood," in *IEEE ICNP*, 2019, pp. 1–2.

[11] Y. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5g," *ETSI*, vol. 11, pp. 1–16, 2015.

[12] W. Wang, "5g mobile platform with p4-enabled network slicing and mec," https://www.opennetworking.org/wp-content/uploads/2019/09/2pm-Wilson-Wang-5G-Mobile-Platform-with-P4-Enabled-Network-Slicing-and-MEC.pdf, September 2019.

[13] J. T. Hagen and B. E. Mullins, "Tcp veto: A novel network attack and its application to scada protocols," in *2013 IEEE PES ISGT*. Washington DC, USA: IEEE Computer Society, Feb 2013, pp. 1–6.