IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 February 2023, accepted 3 March 2023, date of publication 10 March 2023, date of current version 15 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3255985

RESEARCH ARTICLE

Adaptive Network Traffic Reduction on the Fly With Programmable Data Planes

CSABA GYÖRGYI[1], PÉTER VÖRÖS[1], KÁROLY KECSKEMÉTI[1], GÉZA SZABÓ[2], (Senior Member, IEEE), AND SÁNDOR LAKI[1], (Member, IEEE)

[1]Department of Information Systems, Eötvös Loránd University (ELTE), 1117 Budapest, Hungary
[2]Ericsson Hungary Ltd., 1117 Budapest, Hungary

Corresponding author: Péter Vörös (vopraai@inf.elte.hu)

This work was supported in part by the Project Strengthening the European Institute of Innovation and Technology (EIT) Digital Knowledge Innovation Community in Hungary under Grant 2021-1.2.1-EIT-KIC-2021-00006; and in part by the Ministry of Innovation and Technology of Hungary from the National Research, Development and Innovation Fund under Grant 2021-1.2.1-EIT-KIC. The work of Péter Vörös was supported by the ÚNKP-22-4 New National Excellence Program of the Ministry for Culture and Innovation of the National Research, Development and Innovation Fund. The work of Sándor Laki was supported by the National Research, Development and Innovation Office—NKFIH under Grant FK_21 138949.

ABSTRACT Industrial networks rely on standard real-time communication protocols, such as ProfiNet RT. These real-time protocols use cyclic data exchange between IO devices and controllers. Each IO device reports its internal state to the controller at a predefined frequency, even if the state of the device is unchanged. These reports are essential to accurately monitor the health of the devices, but network resources are limited and it is not advisable to overload the network with unnecessary packets. The traffic generated by a single device is insignificant, but in an industrial site with hundreds of such devices, the number of packets to be transmitted adds up. As cloud-based industrial controllers (e.g., cloud-based soft-PLCs) become more prevalent, all generated IO device traffic must be forwarded over the access link to edge computing or private/public cloud infrastructure. Wireless (e.g. 5G radio) transmission of many small packets leads to spectrum efficiency issues and high power consumption. In this paper, we propose an in-network solution to significantly reduce industrial network traffic by cooperating with two P4 programmable network elements deployed on both sides of an access link. Excess traffic is filtered out and new data content is cached at both ends while detecting both link and device failures in real-time. The adaptive mechanism introduced allows the system to automatically optimize its efficiency and performance by dynamically enabling and disabling traffic filtering/caching. Our measurements show that the method can significantly reduce the wireless link load while being seamlessly deployable in existing industrial environments without modifying the protocol, IO devices, and controllers.

INDEX TERMS Industrial communication, Internet of Things, SCADA systems, software-defined networking.

I. INTRODUCTION

Over the past few decades, industrial controls have evolved from boards full of control relays to modern programmable logic controllers (PLCs). Other types of specialty controllers such as distributed control system controllers for the process control industry have also developed as industrial controls became more and more automated. Nowadays the whole industrial network is changing quickly. More and more cloud-based solutions appear to replace the traditional industrial controller hardware. With the advent of 5G, a stable, highly reliable network connection can be established over the radio similar to the wired counterparts. The communication between different parts of an industrial site or other remote locations, such as private and edge cloud nodes became a viable option over the radio.

While the idea is functional, connecting programmable logic controllers and IO devices via 5G radio presents a number of new challenges. Sensors and PLCs are currently implemented to send status signals with predetermined

The associate editor coordinating the review of this manuscript and approving it for publication was Renato Ferrero.

frequencies usually between 1 and 1000 Hz. This amount of messages leads to a high overhead affecting both spectral and energy efficiency. Analyzing the contents of these packets, we found that most of the data is redundant, they do not contain any new sensor information. While these packets are redundant, they also take a crucial part in the connection, because both PLCs and IO devices are very sensitive to packet loss and jitter. In our work, we classify these messages and treat redundant packets as simple life-signals, while using the valuable sensor information to operate the system.

In this paper, we focus on real-time industrial protocols that implement a communication behavior called cyclic data exchange (for example ProfiNet [1]). In such protocols, each IO device and the appropriate PLC cyclically exchange data packets with a predefined frequency. This predefined frequency communication does not allow us to simply change existing wired networks to wireless, simply because hundreds or thousands of such devices will deplete the radio spectrum very quickly. To overcome the radio limitation we propose a new method, using the concept of in-network computing to substantially reduce the network traffic that we need to transmit over the radio link. By installing a programmable switch before the radio transceiver at both ends of the communication, we can get rid of the unnecessary data transmission from IO devices whose internal state is unchanged. Each programmable switch is capable of storing the device states, and also is able to detect missing packets. Since both switches have this information, the sender switch only needs to transmit updates and error information over the radio. The receiver switch, if it does not receive any status updates or error information from the radio, will continuously generate and send out the life signals to the PLC.

Our solution adds a layer of logic on top of existing protocols and does not require any modifications to the protocol itself, the PLC, or the IO devices. Seamless deployment of this system is possible by placing two instances of the P4-programmable switches (or smartNICs) at the two sides of the critical link for example the radio. No further changes in the infrastructure or modifications in the configurations are required. The proposed solution provides a real-time reaction to link and device failures but still does not introduce significant computational and memory overhead. The reduction in network load achieved by eliminating redundant traffic opens up new uses for wireless communications that were previously not possible due to the limitations described above, without the need to implement new communication protocols. Our solution makes it possible to convert existing systems using different real-time communication to wireless without the difficulties of deploying new protocols.

This paper extends the work of [2] with the following additional features making the method more efficient in non-static environments: 1) We introduce an adaptive mode of operation, where the traffic reduction module dynamically switches on and off for each device depending on the number of state changes. 2) We propose a missing event detection method to recognize both device and link failures quickly. Our pipeline implements a mechanism that exchanges information about the availability of IO devices and also about the state of the radio link. 3) We present the integrated pipeline implemented in P4 and evaluate it on an Intel/Barefoot Tofino-based hardware switch. 4) We have also extended the evaluation with several new scenarios, various network settings, and further aspects.

II. RELATED WORK

To provide computer networks with a high degree of flexibility and scalability Software Defined Networking (SDN) [3] introduced a new way of programming abstractions by decoupling the data and the control plane functionality. While the literature on control plane programmability has a rich past, difficulties of programmable and portable data planes have just started gaining attention in recent years. To offer network developers the desired flexibility, specific programming languages have evolved. These languages let experts describe the entire packet processing pipeline in a protocol-independent way from a high-level abstraction. P4 [4] is one of the language propositions, which has achieved the most influential community support, backed by members from both industry and academia. The language has numerous compilers for diverse software and hardware targets, ranging from general-purpose processors, NetFPGAs [5] and SmartNICs, to custom-designed sets of ASICs such as Intel Tofino.

Protocol-independent network programming opens up the fields for a new era, in which switches are more than simple packet-forwarding tools. By offloading low-latency processing rules to in-network devices, network hardware can also take part in calculations on the application level during communication. These newborn paradigms are called in-network computing and edge computing, where server-based computations are offloaded partly or completely to the programmable switches. In contrast with cloud computing where computing servers are located far away from end-users, edge computing and in-network computing offer computation very close to the endpoints. Doing so will minimize the response time, therefore satisfying the low-latency constraints in various domain-specific applications became viable.

The integration of SDN into industrial networks has proven to be beneficial in terms of increased reliability, scalability, and cost-efficiency. A number of scientific papers have been published over the years on the topic, highlighting its advantages as well as potential challenges that need to be addressed [6], [7]. In particular, research has focused on improving network performance, reducing latency, increasing the reliability, and security of industrial networks [8]. Other research topics have included the development of algorithms for better traffic management, optimizing energy consumption in industrial settings, and enabling mobility across various industrial applications [9]. Kim et al. demonstrate a prototype implementation for in-band network telemetry [10] with P4 language, using a software switch as the implementation platform. They show how their implementation

can be used to diagnose various performance problems. Jin et al. [11] implement NetCache, which is a new rack-scale key-value store architecture that leverages in-network caching to provide dynamic load balancing across all storage servers. Bremler-Barr et al. [12] show how a simple L7 load balancer over Software-Defined Networks can work. NETHCF a line-rate in-network system using programmable switches to design a novel defense against spoofed IP traffic is introduced in [13]. Laki et al. [14] present that with the advent of P4, description, validation, and evaluation of AQM algorithms in a generic framework has become possible since the different drop policies applied by these methods can be implemented in ingress and/or egress control blocks of a P4 program.

Not originally developed for the execution of complex algorithms, these devices are restrained by their processing capability, and the rules to be executed on them are limited. However several studies showed, those in-network devices can effectively be used to execute simple control algorithms for example [15] controlling an inverted pendulum. Cesen et al. [16] offer an emergency action execution platform to overcome the latency problems caused by the possible connection problems between the devices and the controller. The authors produce emergency packets, with stop commands, directly from the data plane. The emergency action can be triggered if the switch detects a packet with a specific payload. In Industrial IoT, the infrastructure needs to handle the data generated by millions of sensors, and support the control of production processes in real time. Mai et al. [17] proposed an edge- and in-network computing architecture for industrial IoT, where they use complex event processing to transform the application-specific functions into operation units and offload them to the network devices. Security is also a serious concern in industrial IoT. Wusteney et al. [18] in their paper, analyze the arising problems and quantify the delay and jitter impacts caused by using firewalls and packet filters in TSN networks.

III. SYSTEM OVERVIEW

Assume a cloud-assisted industrial environment where the data communication between IO devices (sensors and actuators) is deployed in the industrial site and software PLCs running in the cloud (public, private, or edge). The industrial site has a 5G radio access that transmits all the traffic of the real-time protocol between the software PLCs and the IO devices. Each PLC continuously queries the state of the IO devices with a predefined frequency. The typical update period fits into the range of 1ms-10s and may vary from device to device. In our system model, we assume two operational phases of each device: 1) Active phase when the reported IO data continuously changes, representing the case when the IO device (e.g., an actuator) performs an industrial task or its environment is not static (e.g., in case of a sensor), and 2) Passive phase when the IO device is in an idle state (e.g., a robot arm waiting for the next product) or its environment is static (e.g., a temperature sensor in the hall), leading to traffic with constant IO data. In passive phases, packets report the same IO data, but this traffic is still needed for PLCs to maintain the aliveness of controlled IO devices. Note that industrial protocols (such as ProfiNet RT) do not tolerate significant packet losses. In our system model, passive and active phases alternate during the operation of each IO device. When an IO device is active, all the IO data carry important information needed for the continuous monitoring of the industrial process, but in passive phases, a large amount of redundant data may be transmitted toward the PLCs polluting the radio link. In such a case, traffic reduction on the wireless access link would have several advantages including improved spectrum efficiency and reduced power consumption.

To avoid the transmission of unnecessary packets over the radio link, we propose an in-network traffic reduction method. Accordingly, we assume two P4-programmable switches at the two sides of the radio link (as shown in Fig. 1). The two programmable switches cooperate to keep track of the latest state (reported packet data) of IO devices, filter out redundant traffic, recognize whether a device is in an active or a passive phase and turn traffic reduction on and off accordingly, and detect the device and radio link failures. As illustrated in the figure, each P4-switch emulates the presence of the devices (either IO devices or PLCs) located on the other side of the radio link, responding to packets of the local (deployed on the same side of the radio) devices. Both P4-switches implement the same data and control planes and the proposed method has the advantage that no modification in the IO devices and the protocol is needed.

IV. IMPLEMENTATION

The proposed approach combines three key functionalities: 1) algorithms for storing and updating packet data and reducing data traffic over the radio link, 2) methods for monitoring the status of the radio link and the different IO devices, 3) dynamic enable and disable of the traffic reduction, to adapt to the network characteristics. In this section, we discuss these functions separately and show how they can be implemented in P4 programmable data planes as a single packet processing pipeline. An overview of the whole pipeline is shown in Fig. 2.

A. DATA TRAFFIC REDUCTION

In industrial real-time protocol communication, an IO device and the responsible PLC continuously exchange state information at a predefined frequency. This frequency is negotiated between the PLC and the IO device in advance, using control packets. Traditionally PLC communication is considered not feasible over wireless links, due to the enormous number of tiny packets generated by the IO devices. Our goal was to create a model, which we can use in real-world PLC communication regardless of the frequency, and reduce IO traffic on such a scale that allows wireless access links to interconnect the industrial site and the cloud. By caching IO

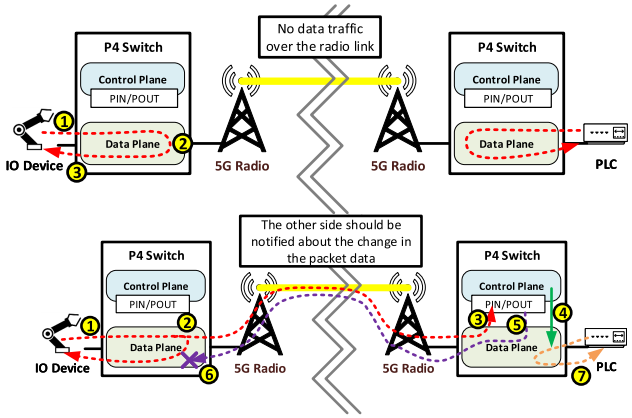


FIGURE 1. The latest packet data is stored by the switch used to generate the response expected by the other end. If the packet data of a particular device is changed, the packet is forwarded to the switch on the other side to update the stored data content.

data contents in programmable switches, we are able to filter redundant traffic and decrease the load of the critical link.

The proposed pipeline has two main tasks: 1) caching IO data and sending automatic responses and 2) packet filtering and state change detection. Each IO device expects replies to their messages, therefore traditional traffic filtering (dropping packets) is not feasible in our scenario. The P4-switch has to handle the incoming data packets from local IO devices, store their carried contents, and send back a response on behalf of the PLC controller on the other side of the radio network, without transmitting data over the wireless link.

The packet filter is responsible for detecting a change in the reported device state (IO data) and for notifying the P4-switch on the other side of the radio link where the stored state information needs to be updated. Fig. 1 depicts our model, and the key steps of handling an incoming packet. We deploy a P4-programmable switch on both sides of the radio, and these switches in cooperation will handle the previously described traffic reduction and automatic reply functionality.

When the state of an IO device is unchanged, it periodically sends redundant data. Packet processing in this case consists of the following steps: 1) The source IO device *src* generates a packet to the destination PLC *dst*. The packet arrives at the P4 capable switch deployed before the radio link. 2) The switch first calculates a hash from the packet data. If the newly calculated hash equals the hash value stored in a register for device *src*, the switch will not forward the packet through the radio link. Instead, it generates a reply for that packet identical to a valid reply from the PLC *dst*, and returns the packet to the IO device. The reply can be created from the data packet by modifying the protocol header fields: setting the device identifier to the ID of *dst*, updating the cycle-counter from a register, and filling the IO data with last seen bytes of *dst* from a match-action table. 3) Finally, it sends the packet back to *src*. Device *src* will treat the received packet as if the actual sender was *dst*.

After describing how the P4-switch can handle the idle stages of an IO device, take a look at the process of managing device state changes. 1) Source IO device *src* generates a packet to the destination PLC *dst* as previously. 2) The P4-Switch before the radio link hashes the packet data, compares it to the cached value, and recognizes that it is different, indicating state change in the *src*. The packet is cloned and the switch transmits one copy over the radio link, while the other packet is returned to *src* with changes identical to the previous case. 3) The P4-switch on the *dst* side of the radio link receives the packet containing the status change. This switch notifies the control plane (CP) about the status change and forwards the packet to the CP. 4) The control plane fills the match-action table responsible for storing the packet data of IO devices with the new data content of *src*. 5) The CP then sends the packet back to the data plane marked with a CP-ACK flag. The data plane directs it towards the radio link. 6) The packet flagged with CP-ACK arrives back to the P4-Switch at the *src* side. After recomputing the hash of the data, it updates the stored hash value of *src* (stored in a register). Finally, it drops the CP-ACK packet. 7) *dst* side P4-switch will use the updated packet data of the IO devices, therefore the automatic responses will contain the latest information.

This pipeline uses two register arrays indexed by a device ID to store the sequence numbers called CycleCounters, and hash values to check the validity of the cached packet data. Cached IO data is stored in a match-action table where the keys are the device IDs, and the table action after a match is used to overwrite the data bytes of the packet with the latest observed values.

While the abovementioned functionality works well in most cases, there are a few scenarios where additional logic is needed. The first problem with our model happens if the delay between the interacting P4-Switches is bigger than the send frequency, so the delay is higher than the cache update time. Let us show an example when the IO device starts alternating its IO data between the same two values denoted by "A" and "B" (e.g., A A A B A B A B ...). In this case, the first "B" is detected as expected. However, if the radio delay is big enough, so the CP-ACK does not arrive back in time, and the cache will still contain the value "A". The next "A" is then not forwarded over the radio since its IO data is identical to the cached value.

To solve this issue, a slight modification is needed in the original algorithm. Accordingly, when the method detects that the hash is changed, it immediately overwrites the stored hash with an extremal value, so false positive caches cannot occur, and for every individual status change the other side will be notified. While this extension handles the previously stated case, it is still not robust enough to handle every delay-sensitive situation shown in Fig. 5. This example illustrates the case when the CP-ACK feedback for a previously sent "B" packet arrives back late, just before the IO device moves from state "A" to "B". The CP-ACK updates the hash, resulting in a situation where the "B" packets will not trigger the notification of the other side about the changed state. The transition from state "A" to "B" is only recognized later when the CP-ACK for the first "A" packets arrives back.

Considering the limited capabilities of P4 programmable devices, we extended our model with an additional register array, that stores the hash values of the latest observed IO data packet for each device. The P4-switch when receiving a packet stores the calculated hash value in the new register. CP-ACK-s are only committed if the acknowledged hash matches the hash of the latest data stored in the new register. With this final extension, our model can handle the delay-sensitive problems, while the pipeline is still not too complex to be able to deploy on P4 programmable hardware.

B. MISSING EVENT DETECTION

To make the application of the proposed data reduction method transparent to IO devices, it is important to react to failures as soon as possible. To this end, the proposed pipeline contains a mechanism that exchanges information about the availability of the devices and the state of the radio link. If failure is detected, the automatic responses described in Sec. IV-A need to be stopped by the appropriate P4-Switch to ensure the original behavior of the industrial network. Each received packet expresses the viability of the sender device by setting a bit in a bitmap register. This status bit needs to be continuously confirmed. If a device becomes unavailable, the P4-Switches stop the transmission of the automatic responses that acted as life-signals. The main operational steps shown in Fig. 3 are the following: 1) A packet generator engine (hardware or software) generates a probe packet at a predefined frequency. 2) The probe is then filled with the status bitmap of known IO devices by the data plane. This bitmap carries information about the aliveness of devices at the local side of the radio link. To monitor the radio state both P4-Switches maintain a radio time-to-live (TTL) counter that is decremented by 1 whenever a probe is sent. If it reaches zero, the radio becomes unavailable. 3) The probe is sent over the radio link. 4) If a probe is received from the other side of the radio link, the radio TTL in the recipient is set to the initial value (3 in our case). The bitmap of device states is refreshed according to the received information, and the probe packet is dropped. Note that both P4-Switches apply the same probing process, exchanging information asynchronously and independently.

The aforementioned bitmap that gets inserted into the probe packet is created with the use of a primitive queue-like data structure. It comprises a number of registers equal to the number of consecutive probe packets (2 in the evaluation scenarios) during which the aliveness of IO devices is checked. When the probe packet is received at the other side of the access link, the transmitted bitmap is stored and taken into account during the generation of automatic responses. No response message is generated on behalf of non-available devices.

C. ADAPTIVITY

In Sec. V, we show an evaluation scenario where an IO device is in an active phase and thus the proposed traffic reduction method doubles the load on the radio link instead of reducing it. This is caused by the overhead of CP-ACK and Probe packets. To improve efficiency, we extended the traffic reduction pipeline with an adaptivity mechanism that can recognize active and passive phases and dynamically enable and disable traffic reduction. Note that a control plane monitoring the device states can also implement such an adaptivity method by constantly re-configuring the tables and registers. However, this may result in a significant CPU load and introduce extra delays in the system. To avoid these disadvantages, we provide a sole data plane solution for this task.

Originally, each packet received on the radio port is sent to the control plane which learns it by modifying the appropriate table entries. In the adaptive case, there are three possible options upon arrival of an IO data packet: OP1) delivering it to the recipient device, OP2) sending it to the control plane only, and OP3) delivering it to the recipient and sending a copy to the control plane. The appropriate action is selected based on the value of a new register used for measuring how frequently the IO data of a given device changes. At packet arrival on the radio port, the register value is incremented (representing the case when a change occurred) and is decremented whenever an automatic response is sent (no change). These adaptivity-related components are visualized in the pipeline diagram Fig. 2 with a red background color.

Option 1 delivers the packet to the recipient IO device but does not result in CP-ACKs. Thus, packets are simply forwarded without applying the traffic reduction method. Option 2 is the initial behavior. Option 3 delivers the packet, but also learns the new IO data content and sends CP-ACKs back to the other side of the radio link. The latter mechanism allows us to re-enable the traffic reduction method for a given flow fast. Note that for Options 1 and 3, the generation of automatic response messages is disabled (via a bitmap register). To demonstrate the benefits and flexibility of the proposed adaptivity extension, we describe two example configurations.

1) ALTERNATING SHORT AND LONG TASKS

Suppose a robot arm alternates between active and passive phases. During the active phase, its state is constantly changing while in the passive phase, it remains unchanged. In our example, the short task results in an active phase of 10 packets long, while the longer task is active for 200 packets. By creating a set of rules that only allows traffic reduction in the passive phase, the network can be significantly offloaded. Then the decision logic between the three options can be configured as follows: Traffic reduction is on (OP2), if r_change_counter $<$ 11. Traffic reduction is off (OP1), if 10 $<$ r_change_counter $\leq$ 200. And preparation for re-activating traffic reduction (OP3), if 200 $<$ r_change_counter.

2) LOW LATENCY USE-CASE

Assume a sensor that typically shows the same value and where it is vital to be able to notify the PLC immediately if the

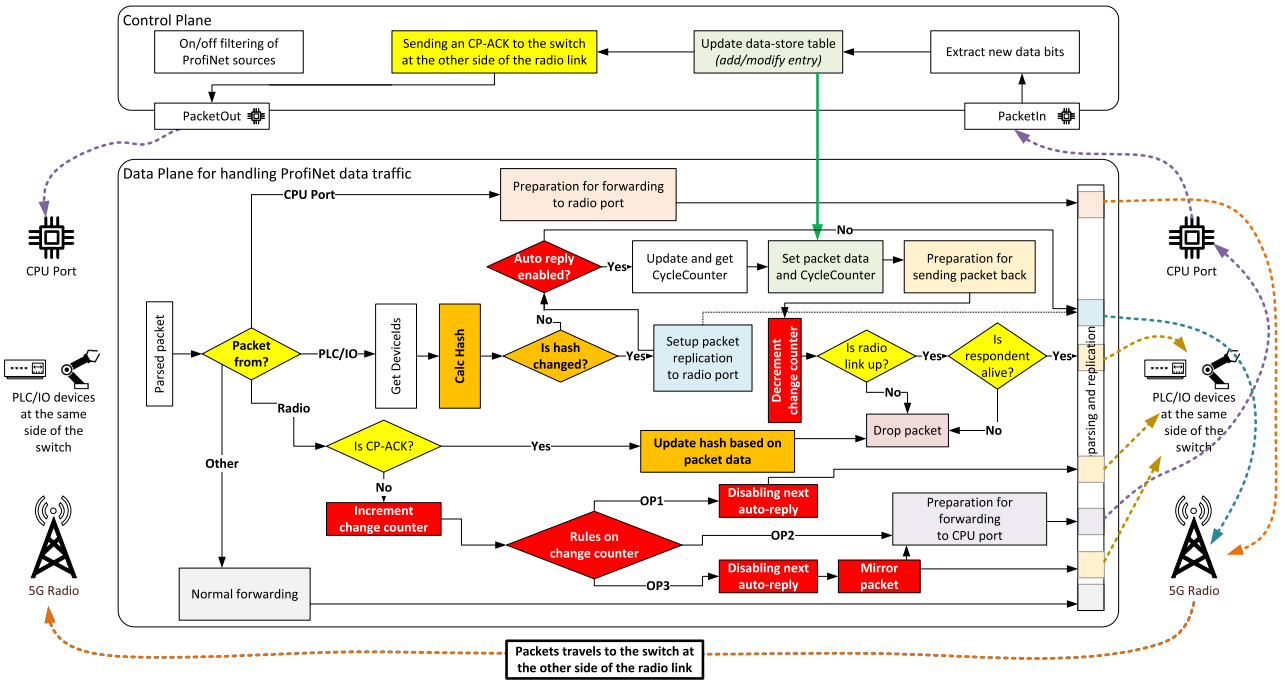


FIGURE 2. The proposed P4 switch pipeline, showing the traffic reduction functionality, visualizing the adaptivity components with red background.

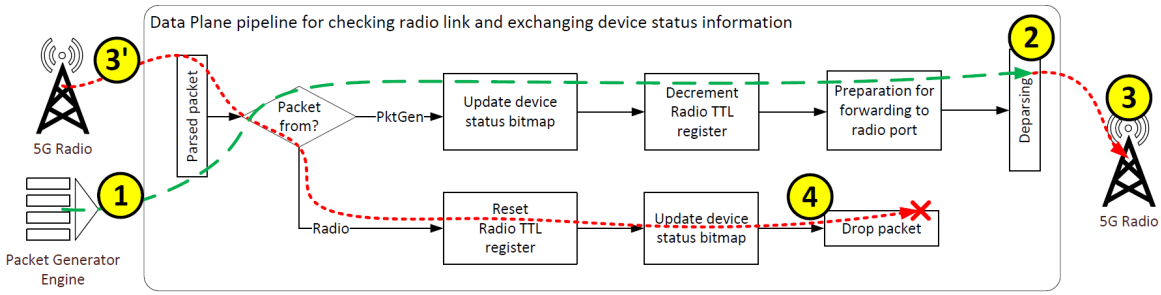


FIGURE 3. Operation of the missing event detection component.

value changes. For this case, a configuration can be provided where the new sensor value arrives at the PLC as quickly as in the original system, while there is no traffic on the wireless link during the long passive phases. In this example, we show another example rule set that aims at minimizing the reaction time. The default rule is to apply OP1 meaning that the packet received from the other side of the radio link is simply forwarded to its destination. However, for every 25th packet received from the given IO device, OP3 is selected that in addition to the default behavior sends a copy to the CPU for learning the IO data content. In this case, a CP-ACK is also generated and sent back to the other side of the radio, enabling traffic filtering. One can observe that the two sides of the radio link operate differently. On the device side, traffic reduction is on for all the packets, but if a new sensor value is observed, the packet is immediately forwarded through the radio link. On the PLC side, the packets coming from the other side are always forwarded to the PLC without any delays. In addition, the method periodically (25 cycles) sends a CP-ACK back to turn traffic reduction on at the device side.

This configuration ensures that the new sensor value arrives at the PLC as fast as in the original system where traffic reduction is not implemented while in long passive phases, there is no traffic on the wireless access link.

D. EFFECT ON INFORMATION DELAY

In this section, we estimate the effect of our algorithms on the information delay. d_{plc} is the delay of the wired connection between the PLC and the corresponding switch, d_{dev} similarly. d_{radio} is the delay between the two switches including the wireless radio connection. d_{cpu} is the delay of the control plane processing and h is the time between two consecutive IO data packets assuming lossless communication.

Suppose the IO device sends packets with arbitrary content until a given point of time called starting point. After this point, it transmits the same data over and over again. The

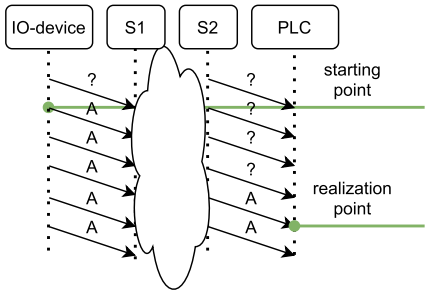


FIGURE 4. Illustration for the definition of information delay.

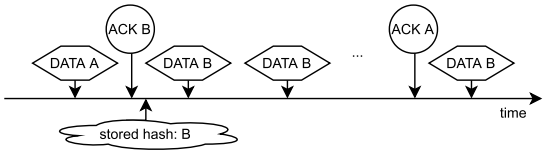


FIGURE 5. The worst case delay sensitive scenario in traffic reduction on the receiver side.

point of time when the PLC receives this repetitive data for the first time is what we call the realization point. We define the information delay as the difference between these two time points as illustrated in Fig. 4.

Using this model, the delay of the original system without the use of our traffic reduction method is always $d_{dev}+d_{radio}+d_{plc}$. Without any adaptivity features, the optimized flow has a delay of $d_{dev}+d_{radio}+d_{cpu}+d_{plc}$ because instead of delivering the packet the P4-Switch forwards it to the control plane and learns its content.

The worst-case scenario is when the change is not recognized immediately. For example, a CP-ACK arrives just before the starting point with the same data the IO device will send in the next cycle (Fig. 5). The total delay sums up to $3d_{radio} + 2d_{cpu} + 2h + d_{plc} + d_{dev}$ in the worst case. It can be proved that the proposed traffic reduction strategy always recognizes the change immediately. Thus, the worst case scenario equals to $d_{dev} + d_{radio} + d_{cpu} + h + d_{plc}$. If we configure the adaptivity feature in a way that it uses only direct forwarding and delivering with a copy to the CPU (Option 1 and 3), then the information delay can be reduced to the original $d_{dev} + d_{radio} + d_{plc}$.

We studied how dropped packets impact the operation on a lossy channel. First, note that for proper operation, the channel should be error-free because real-time protocols demand very strict delay and jitter constraints, and if the channel cannot satisfy that the whole operation is at risk. However, if packet loss occurs it can be divided into two categories. If the acknowledgment is lost, it has no impact on the information delay, because it means that only the data packet will be retransmitted and reacknowledged. In this case, the recipient already has the changed data in time. If a data packet gets lost, then the packet will be retransmitted, which will introduce a delay d_{radio}.

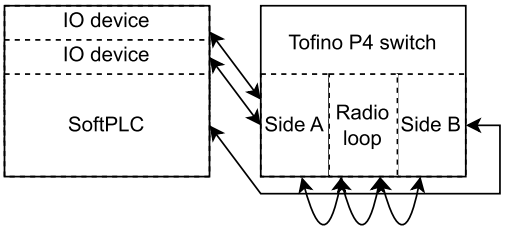


FIGURE 6. Hardware topology diagram with a Tofino-based P4-Switch, x86 servers running Codesys SoftPLC, and IO devices.

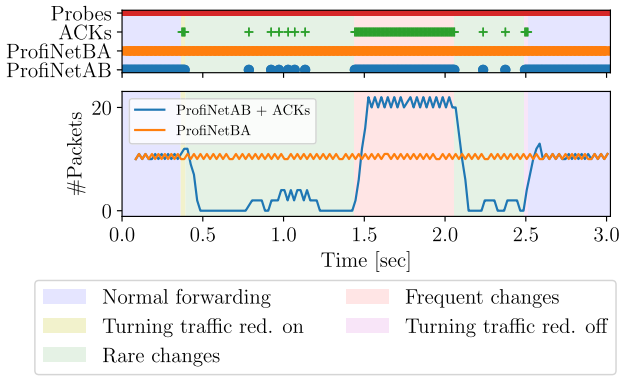


FIGURE 7. Individual packets and the number of packets transmitted over the radio link without adaptivity rules (moving average with 85 ms window).

V. EVALUATION

We have implemented the proposed solution in P4 using the Tofino Native Architecture (TNA) and carried out the evaluation in our testbed using a Tofino-based P4-Switch (Stordis BF2556X-1T) and x86 servers running Codesys IDE, Codesys SoftPLC,[1] and an IO device based on the open-source ProfiNet device stack of RT-Labs[2] as it can be seen on Fig. 6. Note that we use a single P4 capable switch in the evaluation representing the two instances (two pipes) of the proposed method at the two sides of the radio link. We emulated the reliable radio link using a loop directly interconnecting two ports of the switch.

A. TRAFFIC PATTERNS

Fig. 7 presents an example of traffic patterns observed on the radio channel. Packets belonging to four different categories: individual ProfiNet IO data (*ProfinetAB*, *ProfinetBA*), CP-ACK (*ACKs*), and probe (*Probes*) packets. In the case of *ProfinetBA* P4-Switches only do simple forwarding. Thus, it can be used as a baseline, representing the data communication if the proposed traffic reduction method is not applied. For comparison purposes, both *ProfinetAB* and *ProfinetBA* send IO data packets every 8 ms. One can also observe that the Probes introduced in Sec. IV-B are continuously present on the channel. The P4-Switches send them to monitor the

[1] https://store.codesys.com/codesys-control-for-linux-sl.html
[2] https://github.com/rtlabs-com/p-net

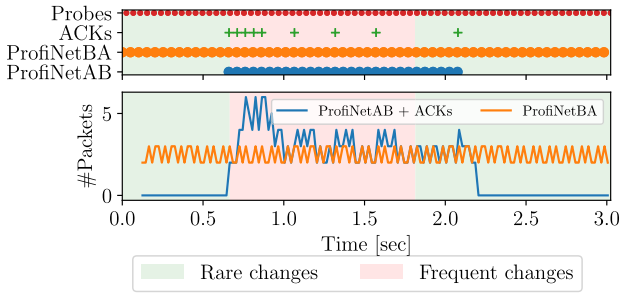


FIGURE 8. Individual packets and the number of packets transmitted over the radio link using adaptivity rules (moving average with 85 ms window).

state of the radio link and exchange information about the availability of devices deployed on the different sides of the critical link. Note that this traffic represents a constant overhead of the proposed method which is independent of the number of IO devices, PLCs, and their load in the system.

The timeline of the measurement mentioned above includes seven regions. 1) In the beginning, the traffic reduction method is not applied. *ProfinetAB* packets are forwarded by the switches like *ProfinetBA*. Thus, there are no CP-ACK packets in the system. Because of that, the traffic levels are the same for *ProfinetAB* and *ProfinetBA*. 2) The second interval is a short transitional phase where we turn the proposed traffic reduction method on. It begins when the switch at the other side of the radio link starts memorizing the IO data and sending CP-ACK responses. The transition ends when the other switch starts filtering out the unchanged *ProfinetBA* data packets. 3) The third phase illustrates the best scenario for our traffic reduction method, resulting in packet transmission over the radio link only if the packet data was changed. The data rarely changes, only six times in this region. One can observe that a CP-ACK packet is transmitted for each transmitted IO data packet, leading to two transmitted packets instead of one. However, the number of transmitted packets is negligible even with these CP-ACK packets compared to the reference traffic *ProfiNetBA*. 4) In the fourth phase, each packet carries new data bytes (active phase), illustrating the case when achieving traffic reduction is impossible. Moreover, CP-ACKs double the total number of packets transmitted over the radio. Note that we can mitigate this worst-case by dynamically turning on and off the proposed solution. Thus, the traffic can be maximized at the original level (same as *ProfiNetBA*). 5) In the next phase is the same as the third one. The device turns back to a passive state with few changes in the data. 6) We stop the traffic reduction method and switch back to normal forwarding of *ProfiNetAB* traffic. 7) The traffic reduction method is turned off like in the first phase.

Fig. 7 shows an evaluation scenario where the adaptivity feature of our method is not applied. As a result, one can observe a two times higher traffic level in the region of frequent changes (rose area). In Fig. 8 we present a scenario using the adaptivity feature to dynamically turn on and off

	30 relax 20 change		10 relax 20 change		random intervals		20% prob. of change	
	1 ms	*5 ms*	*1 ms*	*5 ms*	*1 ms*	*5 ms*	*1 ms*	*5 ms*
optimised	84%	88%	140%	147%	139%	144%	40%	70%
adaptive	64%	66%	93%	101%	88%	92%	41%	72%

FIGURE 9. Comparison of traffic levels in different scenarios.

traffic filtering. The applied rule set turns traffic reduction off after 5 consecutive changes. In order to have a chance to enable traffic reduction when the IO device becomes passive again, the P4-Switch at the PLC side sends CP-ACKs at the end of a cycle interval whose length doubles until the reduction is enabled again or the maximum interval length is reached. The effect of this rule is visible in the figure; the traffic level stays high even after the period of frequent changes until the next CP-ACK. These "false-positive" changes are the price of adaptivity. Packet filtering does not take effect until the latest IO data is learned and acknowledged.

B. TRAFFIC REDUCTION ABILITY

As shown previously in this section, the achievable traffic reduction capabilities highly depend on how frequently the IO data to be transmitted changes. However, other factors also influence the result: the frequency at which IO devices send data exchange packets, and the response time of CP-ACK packets (the time between forwarding a new data packet through the radio link and receiving the corresponding CP-ACK).

To compare the performance of original (without traffic reduction), optimized (with traffic reduction) and adaptive (traffic reduction with adaptivity) approaches with a different setting, we implemented simulation scenarios, using SimPy,[3] a discrete event simulation library.

We consider four different traffic patterns. In the "30 relax 20 change" scenario, the device sends 30 packets with the same content and then 20 packets with constantly changing content (longer relaxed and medium-long changing phases). This pattern is repeated in the scenario. Thanks to the longer relaxed and not so long active phase, the optimization can filter out many packets. The adaptive approach can even further optimize the medium-long changing phase by reducing the number of CP-ACKs sent during passive phases.

The "10 relax 20 change" scenario describes a similar case with a short relaxed phase (10 packets with the same content and then 20 packets with constantly changing content). Since the changing phase is twice as long as the relaxed phase, the bad performance of the non-adaptive optimization is not surprising. We can also see that the adaptive case is close to the original even in this not-ideal setting.

The "random intervals" chooses the length of the relaxed and changing phase randomly between 1 and 50 cycles. The total lengths of relaxed and changing phases are close to each

[3]https://simpy.readthedocs.io/en/latest/

other which explains the difference between the original and optimized performances. The adaptivity can still improve the efficiency in longer passive phases.

The ''change with 0.2 probability'' scenario changes the packet content with a 20% probability every time. The similarity between the optimized and adaptive results can be explained by the following two observations. First, the probability of very long changing periods is low, and the two approaches behave similarly in passive phases. Second, even if a longer active period appears, the subsequent packet changes or remains the same independently.

Our evaluation with different radio delay values shows similar results. Note that the adaptivity extension requires proper settings. If it is misconfigured (e.g., waiting too long before turning traffic reduction on again), the method generates excess traffic instead of a reduction. We can conclude that our adaptive algorithm can deal not only with long non-changing values, but it can handle longer changing periods too. Even if the traffic reduction is not so significant all the time, it can help in reducing the overhead of the traffic reduction method.

C. RESOURCE USAGE

Our implementation has a relatively low footprint. In the proposed method there is a linear relation between SRAM usage and the number of IO devices, while it does not require any expensive TCAM. The current implementation only uses the ingress pipeline of the P4-Switch, requiring 12 stages (including the adaptivity extension).

Register usage can be a key question. The traffic reduction component uses four different register arrays for the: hashes of the acknowledged packets; last seen packets; list of blocked devices; list of modified flows (once we alter a flow, we have to maintain the cycle counter by ourselves). The adaptivity extension requires at least two more registers: one for the counter register that is the input of the decision-making; and one for preventing the next automatic response. The missing event detection needs an additional n register: for the n elements of the queue like data structure. Lastly, one additional register is used to monitor the availability of the radio.

D. DEPLOYABILITY AND LIMITATIONS

One of the main advantages of the proposed solution is that it does not require changes in other infrastructure elements. It works as a plug-and-play extension to an existing system. The method can dynamically be turned on and off and applied on a set of devices. Our approach can also be used with various other industrial protocols that use end-to-end state transfer. Such open IoT protocols include OPC Unified Architecture or MQTT SN publish/subscribe messaging protocol. Ethercat is a counterexample, which is not suitable for such a system due to its daisy chain architecture. Not only because the daisy chain is not ideal for a system like ours, but also because the traditional implementation of the daisy chain over a wireless link is pointless.

The ProfiNet IO data can be up to 1440 bytes. However, our implementation supports only the minimum-sized packets with 40 bytes of data.

Naturally, the question arises of whether background traffic affects the solution or not. The traffic management engine of today's P4 switches supports strict priority scheduling and queueing, unfavorable drops and higher delays can be prevented by processing the ProfiNet traffic with the highest priority.

VI. CONCLUSION

The communication between IO devices and PLCs produces countless small packets, so it was not feasible to build a radio link between the devices and the cloud-based soft-PLCs. In this paper, we proposed a model in which device states are managed locally to minimize communication over critical links. Using in-network computing, we have presented our prototype that can integrate into real-time industrial environments without affecting the required reliability and without the need to modify the protocol, PLC, or IO devices. We have also shown that by managing redundant sensor data locally, we can significantly reduce the average load on critical links, such as 5G radios between different parts of an industrial site. Through our evaluations, we have demonstrated the benefits of the proposed method in several aspects. We have quantified the impact of traffic reduction both in the standard scenario and using our adaptive switching. We found that our implementation has no significant impact on information latency while also having relatively low resource requirements.

REFERENCES

[1] R. Pigan and M. Metter, *Automating With PROFINET: Industrial Communication Based on Industrial Ethernet*. Hoboken, NJ, USA: Wiley, 2008.

[2] C. Györgyi, K. Kecskeméti, P. Vörös, G. Szabo, and S. Laki, ''In-network solution for network traffic reduction in industrial data communication,'' in *Proc. IEEE 7th Int. Conf. Netw. Softwarization*, Jun. 2021, pp. 191–195.

[3] M. Karakus and A. Durresi, ''A survey: Control plane scalability issues and approaches in software-defined networking (SDN),'' *Comput. Netw.*, vol. 112, pp. 279–293, Jan. 2017.

[4] 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, pp. 87–95, Jul. 2014, doi: 10.1145/2656877.2656890.

[5] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, ''NetFPGA—An open platform for gigabit-rate network switching and routing,'' in *Proc. IEEE Int. Conf. Microelectronic Syst. Educ. (MSE)*, Jun. 2007, pp. 160–161.

[6] T. Kobzan, I. Blocher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, ''Configuration solution for TSN-based industrial networks utilizing SDN and OPC UA,'' in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2020, pp. 1629–1636.

[7] D. Henneke, L. Wisniewski, and J. Jasperneite, ''Analysis of realizing a future industrial network by means of software-defined networking (SDN),'' in *Proc. IEEE World Conf. Factory Commun. Syst. (WFCS)*, May 2016, pp. 1–4.

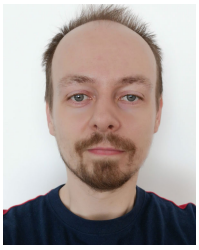
[8] M. Cheminod, L. Durante, L. Seno, F. Valenza, A. Valenzano, and C. Zunino, ''Leveraging SDN to improve security in industrial networks,'' in *Proc. IEEE 13th Int. Workshop Factory Commun. Syst. (WFCS)*, May 2017, pp. 1–7.

[9] M. Ehrlich, J. Jasperneite, D. Krummacker, C. Fischer, R. Guillaume, S. S. P. Olaya, A. Frimpong, H. de Meer, M. Wollschlaeger, and H. D. Schotten, "Software-defined networking as an enabler for future industrial network management," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2018, pp. 1109–1112.
[10] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *Proc. ACM SIGCOMM*, 2015, pp. 1–2.
[11] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing key-value stores with fast in-network caching," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 121–136.
[12] A. Bremler-Barr, D. Hay, I. Moyal, and L. Schiff, "Load balancing memcached traffic using software defined networking," in *Proc. IFIP Netw. Conf. (IFIP Networking) Workshops*, Jun. 2017, pp. 1–9.
[13] M. Zhang, G. Li, X. Kong, C. Liu, M. Xu, G. Gu, and J. Wu, "NetHCF: Filtering spoofed IP traffic with programmable switches," *IEEE Trans. Dependable Secure Comput.*, early access, Mar. 22, 2022, doi: 10.1109/TDSC.2022.3161015.
[14] S. Laki, P. Vörös, and F. Fejes, "Towards an AQM evaluation testbed with P4 and DPDK," in *Proc. ACM SIGCOMM Conf. Posters Demos*, Aug. 2019, pp. 148–150.
[15] J. Ruth, R. Glebke, K. Wehrle, V. Causevic, and S. Hirche, "Towards in-network industrial feedback control," in *Proc. Morning Workshop Netw. Comput.*, Aug. 2018, pp. 14–19.
[16] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, and G. Pongracz, "Towards low latency industrial robot control in programmable data planes," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2020, pp. 165–169.
[17] T. Mai, H. Yao, S. Guo, and Y. Liu, "In-network computing powered mobile edge: Toward high performance industrial IoT," *IEEE Netw.*, vol. 35, no. 1, pp. 289–295, Jan. 2021.
[18] L. Wusteney, M. Menth, R. Hummen, and T. Heer, "Impact of packet filtering on time-sensitive networking traffic," in *Proc. 17th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Jun. 2021, pp. 59–66.

CSABA GYÖRGYI received the M.Sc. degree in computer science from Eötvös Lorand University, Hungary, where he is currently pursuing the Ph.D. degree. He is also a Visiting Researcher with the University of Vienna. He is also participating in EIT Digital Doctoral Programme working with Ericsson Hungary.

PÉTER VÖRÖS received the M.Sc. and Ph.D. degrees from Eötvös Loránd University (ELTE), Budapest, in 2014 and 2019, respectively. He is currently working as an Assistant Professor. Recently, he has been working on projects on network security, traffic analysis, and programmable data planes.

KÁROLY KECSKEMÉTI received the B.Sc. degree in computer science from Eötvös Lorand University, Hungary, in 2021. He is currently pursuing the M.Sc. degree in cybersecurity.



GÉZA SZABÓ (Senior Member, IEEE) is currently pursuing the Ph.D. degree in informatics. He joined Ericsson Research as an undergraduate student, in 2005. He presented the M.Sc. thesis comparing various application traffic classification methods, in 2006. Since 2017, he has been working in the field of industrial 4.0, developing robot cells into cyber-physical production systems with the help of 5G and AI. He is a delegate in 3GPP SA6 and the Vice Chair of IEEE P2940 Standardization Group.

SÁNDOR LAKI (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Eötvös Loránd University, in 2007 and 2015, respectively. He is currently an Assistant Professor with the Department of Information Systems, Eötvös Loránd University. He has authored over 40 peer-reviewed papers and demo papers, including publications at IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, INFOCOM, ICC, and SIGCOMM. His research interests include active and passive network measurements, traffic analytics, and programmable data planes and their applications for new networking solutions.