Programmable Data Planes as the Next Frontier for Networked Robotics Security: A ROS Use Case

Diego Rossi Mafioletti*†§, Ricardo Carminati de Mello†,
Marco Ruffini*, Valerio Frascolla‡, Magnos Martinello†, Moises R. N. Ribeiro†
*Trinity College Dublin, Dublin, Ireland – Emails: rossimad@tcd.ie, marco.ruffini@tcd.ie
†Federal University of Espírito Santo, Espírito Santo, Brazil
Emails: ricardo.c.mello@ufes.br, magnos@inf.ufes.br, moises@ele.ufes.br
‡Intel Corporation, Deutschland GmbH – Email: valerio.frascolla@intel.com
§Federal Institute of Espírito Santo, Espírito Santo, Brazil

***Abstract*—In-Network Computing is a promising field that can be explored to leverage programmable network devices to offload computing towards the edge of the network. This has created great interest in supporting a wide range of network functionality in the data plane. Considering a networked robotics domain, this brings new opportunities to tackle the communication latency challenges. However, this approach opens a room for hardware-level exploits, with the possibility to add a malicious code to the network device in a hidden fashion, compromising the entire communication in the robotic facilities. In this work, we expose vulnerabilities that are exploitable in the most widely used flexible framework for writing robot software, Robot Operating System (ROS). We focus on ROS protocol crossing a programmable SmartNIC as a use case for In-Network Hijacking and In-Network Replay attacks, that can be easily implemented using the P4 language, exposing security vulnerabilities for hackers to take control of the robots or simply breaking the entire system.**

***Index Terms*—In-network computing, security, P4, ROS**

I. Introduction

Envisioned as a key factor for the upcoming generation of service robots, cloud-enabled robots will also play important roles in areas such as eHealth and Industry 4.0 [1]. Given resource constraints imposed by embedded hardware, the possibility of offloading processing into a programmable element closer to the robots (e.g. edge) allows more cost-effective robots cooperating in unstructured environments [1].

Network-related issues (e.g., latency) may prevent further advances of cloud robotics, and edge computing techniques have the potential to alleviate such constraints [2]. Nevertheless, edge computing is an expensive architecture solution when compared to the cloud and may not suit some latency-sensitive and critical applications on production environments (e.g., lower-level controllers). Thus, there is room for exploiting this market using state-of-the-art network programmability. In-network computing is a promising field that uses the capabilities of programmable network devices (e.g., programmable switches and NICs) to offload computing to the network [3].

Figure 1 presents the concept of In-Network Edge (INE), in which a programmable data plane connects robots to edge and cloud servers, and allows for robotic functionality to be instantiated within the network. Leveraging in-network applications based on a consolidated network programming

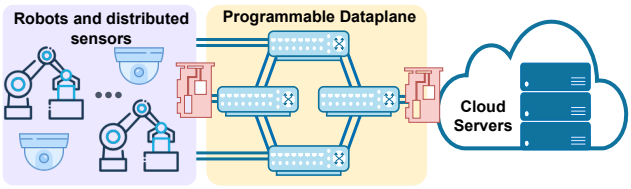


Fig. 1: Cloud Robotics and Programmable Data Plane.

language, such as the P4 language, may enhance management and control at the edge. Since network devices are in physical proximity with robots and distributed sensors, the use of in-network computing for robotics also reduces the overall latency, which is specially interesting for time-critical applications. In this context, INE may be enabled by the P4 language and a NFV framework [4], with potential to unleash real-time functionality in networked robotics.

Nevertheless, routing data from networked robotic systems via programmable network devices opens another window of opportunity for attackers trying to get access to the system. In this sense, in-network computing is a double-edged sword: vulnerabilities in how data is transmitted and interpreted can be explored from within the network. Thus, in a threat model in which malicious applications are running inside programmable devices, aspects related to data security, integrity and validity must be taken into further consideration. Multiple works have advocated for the use of in-network computing for robotics (e.g., [5]–[7]), but to the best of our knowledge this is the first work to address security concerns introduced by programmable network devices to networked robotic systems.

In this paper, we argue that most current robotic systems are vulnerable to simple attacks in a programmable data plane. We discuss two threat models in which robotic systems might be driven to unstable conditions by a compromised network device. We demonstrate such vulnerabilities using a networked robotics system based on the Robot Operating System (ROS), which is currently the most adopted robot development framework, communicating over a P4-enabled network device. The main contribution of this work is to cast a light in how attacks that are well described in the literature

978-3-903176-36-2 ©2021 IFIP

can be refactored to be launched from the network itself.

The remainder of this paper is as follows. In Section II, we further describe the technologies involved in the scenario presented in the introduction, drawing a parallel of our view with related research works. Then, in Section III we present two threat models for networked robotics in programmable data planes, demonstrate possible attacks and discuss related issues. Finally, in Section IV we present a final discussion and the future directions of this research.

II. Background and Related Work

As with other embedded systems, robotics manufacturers place a high priority on safety, development cost, speed to market, and providing customer features. Cybersecurity is a lower, and sometimes, forgotten priority in part because security is not a primary consideration for customers [8]. The end-user demands more concerns on cost, usability, features, and functionality [9]. However, due to their direct interaction with human beings, robotics applications must be required to be more secure and safe than other embedded systems. Overall, various security concerns, issues, vulnerabilities, and threats are constantly arising, including the malicious misuse of these robots via cyberattacks, which may result in serious injuries and even death [10], [11].

A. Programmable Data Planes

Data plane programmability means that the data plane with its algorithms can be defined by the end users, algorithms which are responsible for processing all the packets that cross through a network device. Thus, they ultimately define the functionality, performance, and the scalability of such systems. When data plane programming is provided to end users, it qualitatively changes their power, for both good and evil [12].

The rise of data plane programmability has been leveraged by domain specific languages like P4 [13] for programming high-speed packet processing. In that sense, a new set of in-network applications has emerged such as heavy-hitter detection [14], machine learning classification [15] and caching for distributed services [16]. More recently, cloud robotics related applications [17], [5], [6], [7] aiming to achieve low latency by programmable network devices positioned nearest to the robotic fleet.

From an architectural point of view, a new RMT architecture [18], which evolved to PISA architecture, proposes a flexible parser and a customisable match-action engine. To process packets at high speed, this architecture has a multi-stage pipeline where packets flow at line rate. Each stage has a fixed amount of time to process every packet, allowing fast lookups table operations (e.g. TCAM), manipulating packet metadata and stateful registers. Such technology, which employs the P4 language as basis for programming, led to the creation of new efficient programmable network devices, overriding the past limitation.

B. Placement of Embedded Network Function and its Security

In NFV, the VNF placement is normally defined by where and how many instances of each network function should be placed and allocated. When using this paradigm in conjunction with programmable data planes, we have to define the most desirable place to execute a specific network function, in order to optimise both hardware and software resources.

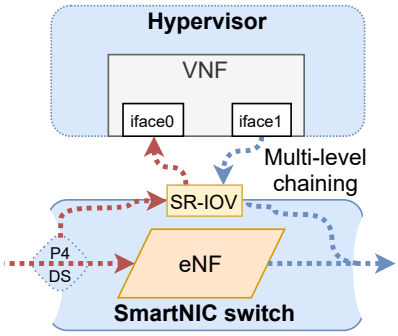


Fig. 2: PIaFFE and multi-level chaining concept.

A couple of works has used the P4 programming language targeting the deployment and placement of micro-applications in different data planes. T4P4S [19] is a software P4 target that relies on interfaces for accelerated packet processing, providing a compiler that translates P4 programs into target-independent C code that interfaces a network hardware abstraction library. Flightplan [20] is a target-agnostic, programming tool-chain that helps with splitting a P4 program into a set of cooperating P4 programs and maps them to run as a distributed system formed of several data planes.

PIaFFE [4] is a framework that uses P4-language for decomposing and deploying Virtual Network Functions (VNFs) into small embedded Network Functions (eNFs) on in-network processors, allowing the correct placement and balance between hardware capabilities from a programmable NIC, using the flexibility of traditional VNFs running on virtual hosts. As depicted by Figure 2, PIaFFE framework can be used to deploy micro-services into programmable data planes, creating small applications that can cope with network-related services (i.e. firewall, routing) and/or high-level network applications (i.e. data encryption, telemetry), enabling the inference on the upper application stack into the network packets. As long as the network traffic arrives at the SmartNIC, PIaFFE employs a P4 Data Structure (P4 DS) – which can be a hash table, a bloom filter, for instance – to steer traffic through the eNF or send it up to the VNF at the virtualisation layer, using SR-IOV as an efficient communication channel.

In this work, we propose the use of PIaFFE for creating ROS micro-applications that are able to interact with the network packets in a transparent fashion, processing network packets when forwarding them, exposing the potential disruption that can be achieved using a programmable data plane with a malicious embedded code.

C. Vulnerabilities in Programmable Network Devices

As desirable properties for securing network communications, we can list confidentiality, message integrity, end-point authentication and operational security [21]. Together,

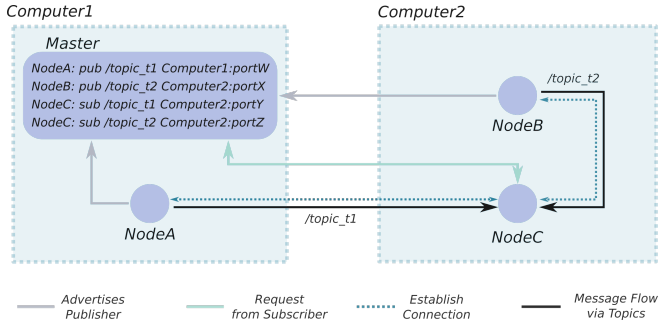


Fig. 3: ROS operation: the master registers and provide information for publishers and subscribers to establish connections and allow nodes to exchange information directly via topics.

these terms define a desirable secure environment for devices to ensure that the messages exchanged between them are authentic and to increase the network's confidence.

Programmable devices are extremely effective in improving performance and giving flexibility to the data plane, by the use of high-level programming languages and compilers which could exploit that emerging programmable data planes devices. However, they bring novel security concerns, such as targeted denial of service and state exhaustion attacks, data plane attacks [22], which need to be exposed both to programmers and the end-user.

A compromised device can disrupt the overall network's confidence, since all packets need to be routed through this type of device, and giving the location in the premises, which is generally considered safe, even a small malicious exploit inserted on it may result in disastrous problems. Notwithstanding, non-programmable network devices are also susceptible to these problems [23], [24], and now with the possibility to customise the data plane using high-level programming languages, these constraints can escalate to a new level, leaving a room for new types of exploits that can be pre-inserted and programmed to catch information or simply to disrupt all network traffic passing through these devices.

D. Security in ROS Systems

ROS [25] is the *de facto* standard middleware for robotics. It separates application management issues and the communication of data, which is abstracted by the middleware. This decentralisation of components pairs well with networked robotic systems and ROS can be used as a base to connect multiple components, even when parts of the system are in the cloud. Given its widespread use, ROS is ideal to demonstrate how programmable data planes can be used to explore vulnerabilities in robotic systems.

To briefly explain how ROS works, the pieces of software that compose the system (i.e., nodes) are executed on top of an operating system (OS) and use the underlying ROS infrastructure to communicate with other nodes. Communication is carried by transmitting well-defined data structures (i.e., messages) via topics in a pub/sub architecture. A ROS master is implemented using XMLRPC, a stateless HTTP-based protocol, and provides naming and registration services to the nodes in the system. The master is used to connect nodes and, once a publishing node locates a subscriber they can communicate peer-to-peer. This process is illustrated in Fig. 3: upon startup, nodes *NodeA* and *NodeB* advertise to the master that they will publish messages to topics named */topic1* and */topic2*, respectively; to subscribe to those topics, *NodeC* communicates with the master to register two subscribers, each associated with one topic. Then, as there are both publishers and subscribers registered to each topic, the master sends instructions to the nodes to establish a connection and start communicating with one another.

Traditionally, custom protocols based on TCP/IP or UDP/IP are used in ROS (TCPROS and UDPROS, respectively) [25]. Upon initialisation, a publisher node provides the ROS master with the topic name and associated data structure (i.e., ROS message type). The master then informs the node about all of the other publishers and subscribers. When a node becomes a publisher on a topic, it will connect to subscribers to that specific topic and there is no access control for topics beyond the data type MD5 hash [26].

In general, no security mechanisms regarding confidentiality, integrity or authenticity are implemented out of the box. Data is transmitted unencrypted and the only information needed to decode intercepted ROS packets is the type of the message being transmitted, which describes the associated data structure and is usually standardised in the ROS framework. Thus, ROS messages can be intercepted and decoded either by direct inspection of the payload. Notwithstanding, the stateless API does not take account of what is happening in the network.

The Secure ROS (SROS) was introduced to add cryptography and security measures to ROS, thus addressing some its vulnerabilities [27]. The SROS enables TLS support for encrypted communication within ROS, access control policies and user-space tooling to generate node key-pairs. Nevertheless, the use of the SROS hampers performance and such a trade-off must be taken into account. Alternatives to the SROS have been presented but none seem to have gained traction. Dieber *et al.* [28] identified that compromised user space libraries can break communication's confidence in ROS. We argue that a compromised programmable data plane can be exploited to the same end.

It is worth noting that ROS' evolution, the ROS2 tackles some of the security concerns with ROS. It uses DDS as the messaging layer, which supports a security standard for protecting messages between parties with access control enforcement. The SROS2 is the current initiative to integrate DDS security and the ROS2 and has been found to significantly affect system performance [29]. ROS' user base is much larger than ROS2', making it more relevant to the security concerns we raise, some of which also apply to ROS2 and will be addressed in our future work.

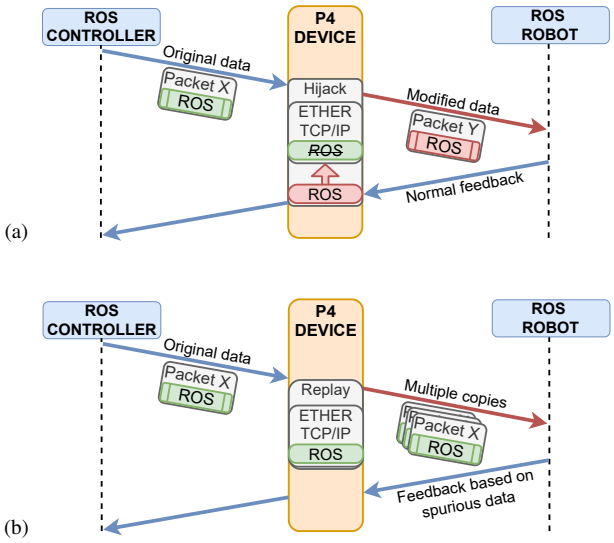


Fig. 4: Threat models: a) hijacking; b) replay attack.

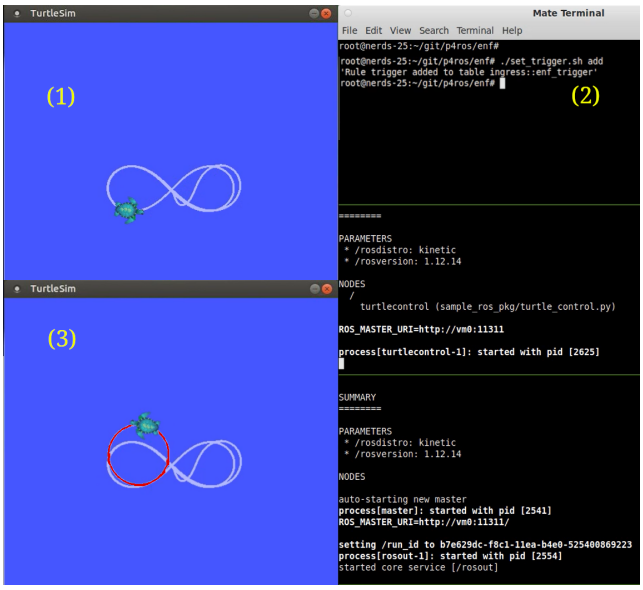


Fig. 5: TurtleSim simulator: (1) Normal robot behaviour; (2) Setting a trigger via P4 table; (3) Abnormal robot behaviour (red track)

III. Threat Models Exposed

In this section, we explore two vulnerabilities in the use of ROS within the paradigm of programmable data planes. Given the projects described in the literature and in the open-source community, the majority of ROS systems would be vulnerable to such attacks.

A. In-Network Hijacking: Man-in-the-middle Attack

Despite ROS' distributed nature, security aspects in communication are not implemented. Common message types encapsulate raw data and it's safe to assume that most ROS' users take no further steps into securing it. Thus, if one can isolate a packet flow and identify the associated data structure interpreting the data is straightforward. More than that, it becomes easy to tamper with the flow by directly modifying the data being transmitted; this can be done in a coherent manner, replacing the actual data for feasible – but incorrect – data, thus making it harder for any automated function to detect that the system has been compromised.

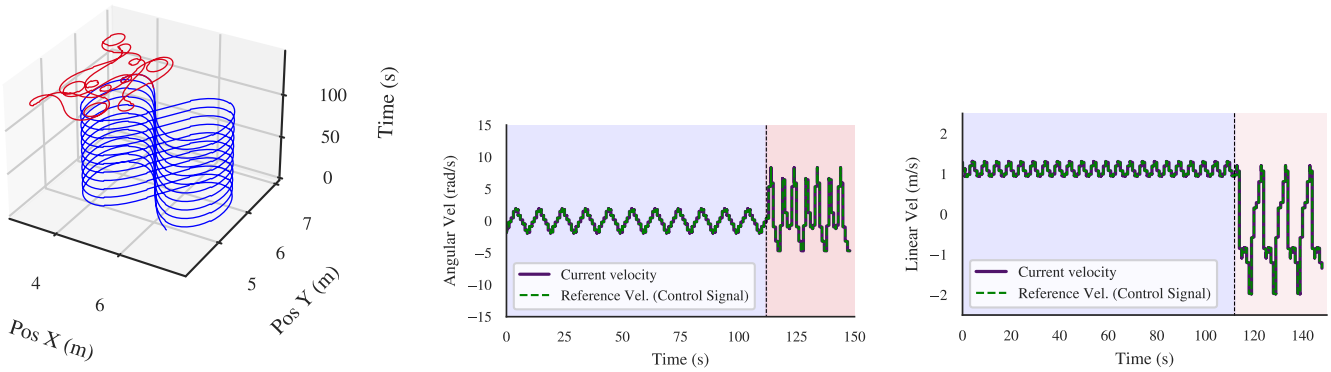
We consider that programmable network devices have transparent access to data exchanged in a ROS system. Such devices may be used to identify a given flow and alter the data being transmitted. There are three assumptions: (i) ROS' standard libraries and messages are used; (ii) ROS' nodes are distributed among different machines, and; (iii) transmitted data is not encrypted. All of these assumptions are compatible with the standard ROS operation and are present in most systems.

To demonstrate how to explore such vulnerabilities using programmable network devices, we implement a ROS system composed of two VMs communicating over a P4-enabled SmartNIC. One VM simulates a robot and the other VM instantiates a navigation controller. In this feedback loop, the robot sends its current position to the controller, which generates velocity commands to the robot. In the SmartNIC, we implement a P4 library relating packet size and part of the payload to standard message types. The SmartNIC parses its traffic to single out packets identified as messages coming from the simulated robot's controller, as exemplified in Figure 4a. Once the target flow is identified, its payload is altered to a given valid instruction, thus hijacking the robot's motion.

We use the *TurtleSim* package to simulate a robot being controlled to follow an eight-shaped trajectory. Given an external trigger, the hijacking takes place and the messages sent from the controller to the robot are modified to trick the robot into following a spurious trajectory. Figure 5 illustrates the outcomes of our demonstration. The controller tracks an eight-shaped trajectory and, once the attack begins, the robot receives tampered instructions to make it follow a circular path. By changing the payload directly, it is possible to inject any instruction to pose as legitimate control output. A similar approach could be used to modify sensor data, introducing artificial noise to hamper system performance without leaving clear signs of an attack.

In our demonstration, we rely on live per-packet detection of a given ROS flow. A more sophisticated approach would be to identify ROS' control packets exchanged among nodes and the ROS master to identify publish/subscriber pairs and the type of message exchanged among them. Thus, to identify the flow associated with a given pair of nodes, one only needs to parse the TCP header in such packets.

Encryption could be used in all communications within a ROS system to overcome such a threat, especially in production environments. Nevertheless, encryption algorithms introduce processing and bandwidth overheads that must be considered since they can degrade the overall performance. In case the internal network is considered to be secure and encryption is only used when communicating with the cloud,



(a) (b) (c)

Fig. 6: Replay attack: a) robot position on plane varying with time, normal (blue) versus disturbed (red) operation; b) angular and c) linear velocities of the robot under normal operation (blue zone) and under attack (red zone); the red dashed line marks the beginning of the attack.

a compromised programmable network device within the local network might also be exploited to a similar result.

B. In-Network Replay Attack

A replay attack occurs when an individual eavesdrops on a secure network communication, intercept the packets, and then delays or re-sends it to misdirect the receiver into doing what the attacker wants, or simply messing with the final proposes of the overall communication. Thereafter, programmable network devices are not only capable of modifying packet data but can also cloning or creating packets, inserting them into a given flow. Thus, it becomes possible to clone valid ROS messages to mislead subscribers, whether data is encrypted or not. Since ROS' subscribers rely on a middleware-level buffer for relaying incoming messages, overflowing the buffer may lead to undefined behaviour in the ROS system. This could be achieved by inserting considerably lower levels of throughput than it would be necessary to disrupt the robot network. By being done transparently, from the inside of the network, such an attack could be hard to be detected and could demand for direct inspection of incoming packets in the robot.

By default, neither ROS or ROS2 implement mechanisms to verify if the arriving data is duplicated. A sequence field is present in the header of some messages in ROS, but it was deprecated in earlier versions. This means that, unless the developer explicitly implements methods to avoid message repetition, every message extracted from the overflowed buffer would be considered valid. One way to mitigate such a threat without directly addressing it is to discard messages with old timestamps but, even if a short time-to-live mechanism is present, the buffer size would have to be tuned accordingly to limit the amount of accepted duplicate data.

For this threat model, we consider that a programmable network device may insert packets in publisher/subscriber flows in ROS systems, as illustrated in Figure 4b. A given flow can be targeted and have its packets cloned to overflow subscribers' buffers. There are two assumptions: (i) ROS' nodes are distributed among different machines, and; (ii) subscriber nodes do not implement any mechanisms to discard duplicate messages. These assumptions are compatible with the standard ROS operation and are present in most users' ROS systems.

We use the same setup described in the Subsection III-A to demonstrate how to generate unstable behaviours in robot systems by flooding a ROS subscriber. In the SmartNIC, we implement a P4 library relating packet size and part of the payload to common standard ROS message types. The SmartNIC is now programmed with P4 code that parses its traffic to single-out a given flow and replicate its packets. In separate experiments, we target the subscriber in the controller node, which receives the current robot position, and the subscriber in the robot's motion node.

As we can see in Figure 6a, the desired robot trajectory (blue) was disturbed by the in-network replay attack, forming a different pattern (red) due to the overload generated by the eNF on the network device. Following the Figure 6b and 6c, we also have the angular and linear velocities of the TurtleSim, captured before and during the attack, showing the erratic behaviour of the system as a result of the disruption caused by the attack. in a real scenario of robotics, this could lead to a catastrophic outcome, since the robot would supposedly be receiving orders from the controller, which in turn would not be aware of what the robot was doing.

IV. Conclusion and Future Works

In this paper, we discuss concerns about the effects of jeopardised programmable devices upon networked robotic systems. We show that micro-services can be inserted into the data plane to intercept and modify network packets. In particular, we implement such micro-services to interact with packets of ROS' systems, causing problems for both controller and the robot fleet. We also discuss the main threat models,

pointing at some aspects in the current implementations of networked robotics, showing the results using P4-enabled network hardware. Our results confirm the possibility of exploiting programmable network devices as attack vectors towards robotics systems.

As future works, we plan to ensure the confidentiality and reliability of the data, adding and checking the information via hash algorithms and/or cryptography using an embedded Network Function. We also intend to explore complex robotic systems based on ROS 2, making use of the DDS machine-to-machine communication protocols.

ACKNOWLEDGMENT

Financial support from Science Foundation Ireland grants 14/IA/2527 and 13/RC/2077 is gratefully acknowledged.

This study was financed in part by CAPES - Finance Code 001. The authors also thank CNPq and FAPES for the financial support granted to this work.

REFERENCES

[1] O. Saha and P. Dasgupta, "A Comprehensive Survey of Recent Trends in Cloud Robotics Architectures and Applications," *Robotics*, vol. 7, no. 3, p. 47, Sep. 2018.

[2] R. Mello, M. Jimenez, M. R. N. Ribeiro, R. Guimarães, and A. Frizera-Neto, "On Human-in-the-Loop CPS in Healthcare: A Cloud-Enabled Mobility Assistance Service," *Robotica*, pp. 1–17, Feb. 2019.

[3] B. Nour, S. Mastorakis, and A. Mtibaa, "Compute-less networking: Perspectives, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 259–265, 2020.

[4] D. R. Mafioletti *et al.*, "Piaffe: A place-as-you-go in-network framework for flexible embedding of vnfs," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[5] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, and K. Wehrle, "Towards executing computer vision functionality on programmable network devices," in *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ser. ENCP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 15–20.

[6] F. E. R. Cesen, L. Csikor, C. Recalde, C. E. Rothenberg, and G. Pongrácz, "Towards low latency industrial robot control in programmable data planes," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 165–169.

[7] I. Kunze, R. Glebke, J. Scheiper, M. Bodenbenner, R. Schmitt, and K. Wehrle, "Investigating the applicability of in-network computing to industrial scenarios," in *Proceedings of the 4th IEEE International Conference on Industrial Cyber-Physical Systems*. IEEE, 2021.

[8] G. W. Clark, M. V. Doran, and T. R. Andel, "Cybersecurity issues in robotics," *2017 IEEE Conference on Cognitive and Computational Aspects of Situation Management, CogSIMA 2017*, 2017.

[9] S. H. Mirjalili and A. K. Lenstra, "Security observance throughout the life-cycle of embedded systems," *Proceedings of the 2008 International Conference on Embedded Systems and Applications*, pp. 186–192, 2008.

[10] L. A. Kirschgens, I. Z. Ugarte, E. Gil-Uriarte, A. M. Rosas, and V. M. Vilches, "Robot hazards: from safety to security," *CoRR*, vol. abs/1806.06681, 2018. [Online]. Available: http://arxiv.org/abs/1806.06681

[11] Ángel Manuel Guerrero-Higueras, N. DeCastro-García, and V. Matellán, "Detection of cyber-attacks to indoor real time localization systems for autonomous robots," *Robotics and Autonomous Systems*, vol. 99, pp. 75–83, 2018.

[12] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, "A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research," 2021. [Online]. Available: http://arxiv.org/abs/2101.10632

[13] P. Bosshart, D. Daly, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "Programming Protocol-Independent Packet Processors," 2013. [Online]. Available: http://arxiv.org/abs/1312.1719

[14] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 101–114.

[15] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[16] 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 *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 121–136. [Online]. Available: https://doi.org/10.1145/3132747.3132764

[17] J. Rüth *et al.*, "Towards in-network industrial feedback control," in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, ser. NetCompute '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 14–19.

[18] P. Bosshart, G. Gibb, H. S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Computer Communication Review*, vol. 43, no. 4, 2013, pp. 99–110.

[19] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "T4p4s: A target-independent compiler for protocol-independent packet processors," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8.

[20] B. T. L. Nik Sultana, John Sonchack, Hans Giesen, Isaac Pedisich, Zhaoyang Han, Nishanth Shyamkumar, Shivani Burad, André DeHon, "Flightplan: Dataplane Disaggregation and Placement for P4 Programs," *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021.

[21] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Boston, MA: Pearson, 2016.

[22] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A Survey on the Security of Stateful SDN Data Planes," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[23] "Vulnerability details : Cve-2020-9115." [Online]. Available: https://www.cvedetails.com/cve/CVE-2020-9115/

[24] J. Wang, R. Cai, and S. Liu, "Research on the protection mechanism of cisco IOS exploit," *Journal of Physics: Conference Series*, vol. 1584, p. 012045, jul 2020.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," *ICRA Workshop on Open Source Software*, vol. 3, 01 2009.

[26] S. Rivera, S. Lagraa, C. Nita-Rotaru, S. Becker, and R. State, "Ros-defender: Sdn-based security policy enforcement for robotic applications," in *2019 IEEE Security and Privacy Workshops (SPW)*, 2019, pp. 114–119.

[27] R. White, D. H. I. Christensen, and D. M. Quigley, "Sros: Securing ros over the wire, in the graph, and through the kernel," 2016.

[28] B. Dieber, R. White, S. Taurer, B. Breiling, G. Caiazza, H. Christensen, and A. Cortesi, *Penetration Testing ROS*. Cham: Springer International Publishing, 2020, pp. 183–225.

[29] J. Kim, J. M. Smereka, C. Cheung, S. Nepal, and M. Grobler, "Security and performance considerations in ROS 2: A balancing act," 2018.