2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit) | 978-1-6654-1526-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUCNC/6GSUMMIT51104.2021.9482549

A Security Monitoring Architecture based on Data Plane Programmability

Amir Alsadi † amir.alsadi@studio.unibo.it
Davide Berardi † davide.berardi@unibo.it
Franco Callegati † franco.callegati@unibo.it
Andrea Melis † a.melis@unibo.it
Marco Prandini † marco.prandini@unibo.it

† Department of Computer Science and Engineering University of Bologna, Bologna, Italy

***Abstract*—Software Defined Networking has put the accent on the implementation of effective, sophisticated algorithms for the control plane, running on centralized devices. Pure centralization, however, also introduces inefficiencies and limitations in many scenarios, often negatively affecting security. Network applications could benefit from data plane programmability, e.g. implementing the increasingly popular P4 language. In this paper, we show that P4-enabled switches can run simple yet significant tasks that enhance the cooperation with the control plane, improving traffic analysis functionalities of practical relevance for security monitoring purposes. We also show how this P4-based solutions can be integrated into an SDN architecture acting as an Intrusion Detection System.**

I. Introduction

The massive growth in cloud computing and IoT applications is driving networks towards more and more complex architectures. Software Defined Networking is essentially the only way to cope with this trend. Its most "clean" adoption foresees the placement of all the intelligence on a centralized control plane, leaving to the data plane devices the only roles of applying simple flow rules to known packets or reporting unrecognized ones. However, under many conditions, the latency of controller-switch communication, as well as the sheer amount of messages that have to be exchanged between them, could hinder the ability of the controller of timely detecting anomalies and providing mitigation plans. Without jeopardizing the model, it is possible to improve the performance of the SDN by evolving the switches, not necessarily in the sense of bringing back control logic into them, but allowing them to execute flexible algorithms for packet manipulation and enabling the implementation of richer interfaces to interact with controllers. This concept of Programmable Data Plane introduces many benefits, including: the drastic reduction of switch-to-controller traffic for monitoring purposes, by sending cumulative metrics; the dynamic (re)definition of packet processing algorithms to suit changing conditions, instead of being limited to act on flow tables; the optimization of switch performances and reduction of their attack surface, by implementing the minimum set of functionalities required at any given time; the list could go on.

In this paper, we want to show that the general benefits of programmable data planes can yield specific improvements to security. To substantiate this claim, after providing the necessary background on models and technologies and specifying the scope of their security applications in Section II, we proceed straight to the presentation in Section III of our SDN-based IDS architecture, composed of three practical solutions we devised. These are basic but significant applications in which programming the data plane allows to achieve results that are beneficial for security purposes, ranging from improving link delay measurement, to detecting suspicious traffic asymmetries, to sharing the load of deep packet inspection. Section IV concludes the paper.

II. Background and motivation

A. SDN, Programmable Data Plane and P4 Language

Software-Defined Networking (SDN) is an enabling technology that gives operators programmatic control over their networks. In SDN, the control plane is physically separate from the forwarding plane, and one controller can manage, using the so-called SouthBound Interface, multiple forwarding devices. Forwarding devices can be programmed with different behaviors, having a common, open, vendor-agnostic interface that enables a control plane to configure them.

The evolution of protocols followed the success of the model. The most commonly used data plane interface at this moment, i.e. OpenFlow [1], started simple, with the abstraction of a single table of rules that could match packets on a dozen header fields. As SDN deployment in large data centers became more common, required features grew in number and complexity, calling for the release of updated versions of OpenFlow [2], which now supports the management of more than 50 header features [3]. This trend appears to have reached a limit. The community agrees [4] that, rather than repeatedly extending the OpenFlow specification, the future switches should support flexible mechanisms to parse packets and match header fields, allowing controller applications to leverage these capabilities through a common interface.

As a result, a new programming language for the data plane arose: *P4*, an open-source programming language that allows to describe how networking gear should process packets. The idea of programmable switches is not a new one. The turning point was marked by the works of [5], in which recent chip designs demonstrate that such flexibility can be achieved in custom ASICs, therefore making the programmable switches perform as well as the traditional ones.

Basically *P4* has three main goals:

- Protocol independence. The switch should not be tied to specific packet formats. Instead, the designer should be able to specify a packet parser to extract any header field, associating it with a proper name and type, and to configure a set of actions to be applied to packets whose header fields match specific conditions.
- Reconfigurability. The controller should be able to install and update the packet parsing logic and processing rules, dynamically in the field.
- Target independence. The controller programmer should not need to know the details of the underlying switch. A *P4* compiler should be provided to translate the program requests into architecture-dependent actions based on the target's capabilities.

Although *P4* can be rightly be defined a programming language, the structure of programs must respect some constraints, namely they must follow a model called "match-action pipelines". The core elements of this model, provided by the *P4* language, are:

Header types - A description of the format of each header within a packet, i.e. the set of fields and their sizes.

Parsers - The allowed sequences of headers within received packets, how to identify those header sequences, and the headers and fields to extract from packets.

Tables - The association between user-defined keys and actions. *P4* tables generalize traditional switch tables; they can be used to implement routing tables, flow lookup tables, access-control lists, and other user-defined table types, including complex multi-variable decisions.

Actions - Code fragments that describe how packet header fields and metadata are manipulated. Actions can take into account external data, supplied by the control-plane at runtime.

Match-action units - They perform the following sequence of operations:

- Constructing lookup keys from packet fields or computed metadata;
- Performing table lookup using the constructed key, and accordingly choosing an action (including the associated data) to execute;
- Executing the selected action.

In summary, as described in [6] *P4* creates forwarding rules that operate manipulations (*actions*, which generally consist of simple operations such as copying byte fields from one location to another) based on the lookup results on learned forwarding state (*matches*). *P4* uses the concept of tables to represent forwarding plane state. Since *P4* does not specify any exact protocol to communicate state between the control and data planes, an interface between the control plane and the various *P4* tables must be provided to allow the control plane to inject/modify state in the program. This interface is generally referred to as the "program API".

B. Security Benefits

Data Plane Programmability in *P4* introduces several benefits from a network management point of view. We argue that many of these benefits can be used to create innovative security-oriented solutions. The following list provides a few examples.

- Expressiveness: *P4* makes many packet-forwarding policies expressible as programs, in contrast to traditional switches, which expose fixed-function forwarding engines to their users. More complex logic can thus be applied when making security-relevant decisions, such as filtering or throttling.
- Hardware support:
  - Portability: *P4* can express sophisticated, hardware-independent packet processing algorithms using solely general-purpose operations and table look-ups.
  - Component libraries: Component libraries supplied by manufacturers can be used to wrap hardware-specific functions into portable high-level *P4* constructs.
  - Decoupling hardware and software evolution: Target manufacturers may use abstract architectures to further decouple the evolution of low-level architectural details from high-level processing.

  Security-wise, these properties enables reliable relocation of functions on different devices, to address issues such as rolling out replacement of vulnerable or resource-constrained hardware, thus easing the mitigation of attacks ranging from "pwning" infrastructural elements to denial-of-service.
- Resource mapping and management: *P4* programs describe resources abstractly; compilers map such user-defined fields to available hardware resources and manage low-level details such as allocation and scheduling. Security policies, as it is in their nature, can be kept separated from mechanisms.
- Software engineering: *P4* programs provide important benefits such as type checking, information hiding, and software reuse. Coding errors that could become vulnerabilities are less likely to happen.
- Customization: instead of having a large set of functions, implemented to face any possible scenario but ending up mostly sitting there unused, *P4* switches implement the exact set of needed functionalities. The reduction in "bloat" translates into an increase in reliability, a contraction of the attack surface, and a more efficient usage of resources.

As a whole, it is possible to say that *P4* enables the network administrator to deploy different detection, prevention and reaction mechanisms that otherwise would be impossible. In this paper, a few specific applications are demonstrated, based on the ability to add new features that enhance telemetry. The overall system can exploit the resulting capabilities of the data plane to implement detection and mitigation of malicious

activities, leveraging the timely processing of traffic data at the switch instead of flooding the controller with unmanageable amounts of (consequently delayed) requests.

C. Related Work

Integrating a Programmable Data Plane inside a monitoring architecture represents a new, promising, yet partial advancement in the quest for efficient attack detection solutions. Each of the proposed *P4* programs represents a piece of the puzzle. There is therefore a need for a higher-level structure that is able to correctly interpret the aggregated data and warnings produced by data plane, so that they can be integrated into an IDS or IPS.

To better justify the correctness of the solutions we proposed, we briefly discuss some of the most relevant related works.

Existing proposals in SDN data plane security have been known to suffer from inaccurate adversarial model. To the best of our knowledge and summarized in [7] among existing solutions, only SPHINX and WedgeTail have practical implementations and proposed a realistic integration between a custom Data Plane and network application though an SDN architecture. Published in 2015, SPHINX [8] is a framework to detect attacks on network topology and data plane forwarding. SPHINX is one the very few solutions to secure data-plane of SDN that does not assume forwarding devices as trusted. It detects and mitigates attacks originated by malicious forwarding devices using two main concepts: first, it abstracts the network operations exploiting incremental flow graphs; then it employs pre-defined security policies specified by the security administrator. Also, it checks for flow consistency using a flow-path built from a similarity index metric which must highlight 'good' switches encountered on the path. WedgeTail [9], on the other hand, is a controller-agnostic Intrusion Prevention System designed to 'hunt' for forwarding devices which fails to process packets as expected. Initially it maps forwarding devices as points within a geometric space and regarding packets as 'random walkers' among them. WedgeTail tracks packet paths when traversing the network and generates their corresponding trajectories. Thereafter, in order to detect malicious forwarding devices, locate them and identify the specific malicious actions, it compares the actual packet trajectories with the expected ones. When a malicious forwarding device is detected, WedgeTail forward the warning through the controller according to the administrator-defined policies.

Although these solutions are efficient, we believe they do not fully exploit the potential of a custom data plane in *P4* language. There are otherwise some related work that aimed to interact with the programmable data plane, with P4 language, in a way that is the controller level that decide what information consider at what priority.

P4ID presented in [10] is a good example of such scenario. In this work authors presented an IDS that combine a rule parser, stateless and stateful packet processing using P4, and an evaluation phase with public available datasets. In this case the results from the data plane are used to train a machine learning process.

In [11] otherwise the IDS is structured in two level. The first level consists of flow and Modbus whitelists, leveraging P4 for efficient real-time monitoring. The second level is a deep packet inspector communicating with an SDN controller to update the whitelists of the first level. In this case then, P4 language is used for real-time monitoring, but deep packet inspection and filtering capabilities are left to the controller.

Despite the community effort in this sector, however, we believe that there is still a lack of integration of different aspects of network security under a single architecture. In fact, the proposed solutions work on a single approach to network security e.g. load balancing, inspection, isolation. What is still missing, to the best of our knowledge, is an architecture that tries to unify these different aspects in a single architecture. The purpose of this work is therefore to fulfill this gap, showing how it is possible to integrate an IDS at the controller level through different components that work at the data plane level, with a single technology (in this case P4) that allows it to be managed and modified in a smart way.

III. IDS Architecture

This section describes our proposed architecture in which we exploit *P4* to implement network analysis functions that can be related to Intrusion Detection Systems. The reference architecture is depicted in Figure 1 and is the composition of three basic functionalities developed to address different tasks, each one calling for a specific kind of analysis that a common IDS needs to support. They are meant to demonstrate that a fully P4-based IDS system can be implemented and to showcase the power and flexibility offered by the language, highlighting the advantages gained with respect to II-C solutions.

The architecture is composed of:

- A program for live packet inspection
- A program for network capacity measurement.
- A program for traffic real-time monitoring.

The feeds from the data plane are integrated at the controller level, where the IDS can read the data and the generated warnings, and use it to intervene in case of an attack. The main achieved goals of this work consist of course on the development of the P4 data plane programs, but also the integration of such programs into the controller level.

The data plane has been realized used the Bmv2[1] switch integrated into the mininet[2] tool. The latter result has been made possible thanks to the recent efforts of the ONOS Brigade Community [12] which developed an integration support for P4Runtime API into the ONOS controller. With such integration we were able to create dedicated pipelines at the controller level. With such pipelines we were able to let the P4 program communicate with the correspondent ONOS application. In this way, P4 programs can raise warnings and send specific measurements to the controller level.

[1]https://github.com/p4lang/behavioral-model

[2]http://mininet.org/

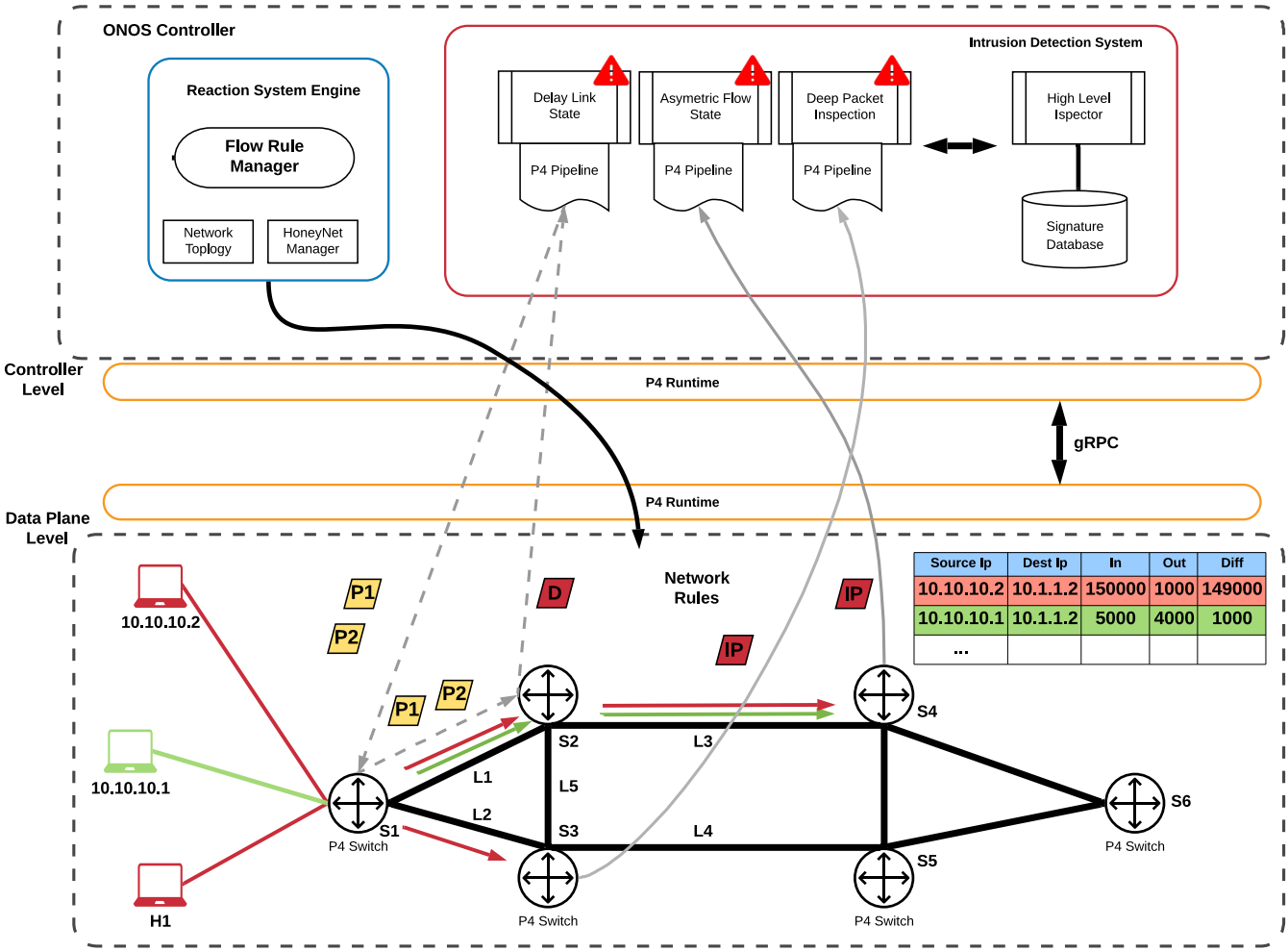

Fig. 1: Reference architecture of the proposed solution to which the Use cases here discusses refer.

In the next sections a detailed description of the data plane programs in P4 is provided.

A. Deep-Packet Inspection

This use case aims at using *P4* as a way to inspect packet payloads. This application wants to provide deep packet inspection capabilities, based on a predefined string. The application is a simple proof-of-concept and it works as follows:

- the payload is loaded as a constant value on the initial program definition;
- in the ingress processing state, we define an action function that takes as input an offset value of the packet header;
- based on which kind of packet we want to inspect, e.g. TCP, UDP ecc, we can call the action function accordingly just specifying the header offset;

In this way, the deep packet inspection is kept as a separate function and can be invoked only if specific conditions on the traffic are verified.
As shown in Figure 1 the switch S3 is the one where the p4 program that implements the deep packet inspector is deployed. He is in charge to notify if the payload in the packet is retrieved, and send to the controller a warning as a packet-in.

In Listing 1, a small block code of the *P4* application performing deep packet inspection is presented.

Listing 1: Proof of Concept of Deep Packet Inspection using *P4* .

```
const bit<32> BAD = 0x626f6d62; /* payload */
.
.
action simple_content_op(bit<32> inparameter)
{
  //meta.mparam = meta.mparam + inparameter;
  hdr.content.word = inparameter;
}

action ipv4_forward(macAddr_t dstAddr,
                    egressSpec_t port)
{
  standard_metadata.egress_spec = port;
  hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
  hdr.ethernet.dstAddr = dstAddr;
```

```
  if (hdr.content.word == BAD) {
    simple_content_op(0x676f6f64);
  }
}
```

B. Link Delay

Availability is one of the pillars of the "CIA" security triad. Measuring responsiveness degradation can be one possible way to detect impending resource saturation. For this purpose, we implemented a precise link delay measurement solution using *P4* to instrument packets at the data plane, and to send the results to the controller. The overall latency a packet experiences when traversing a network is due to different effects [13]:

- Processing delay – time it takes for routers to process the packet header;
- Queuing delay – time the packet spends in routing queues;
- Transmission delay – time it takes to push the bits of the packet into the link;
- Propagation delay – time for the packet-bearing signal to reach its destination.

Some of these quantities are known and unalterable; they either depend on hardware (e.g. the transmission delay is a feature of the network interface) or are physically constrained (e.g. the propagation delay is defined as a function of the distance between the network nodes).

There are no standard techniques to calculate the delay capacity of network. Depending on the level of precision, and at what moment we want to calculate these measures, different techniques have been proposed in the literature, but describing them and evaluating their pros and cons is beyond the scope of this work.

However, we believe that packet probing is the best technique for our requirements. In fact, since the controller knows the network topology of the data plane this technique allows us to work directly at the data plane level with few resources, having a real-time estimate and can be repeated many times in a short period.

The idea, inspired by [14] is defined as following:

1) the Controller targets a link to measure the average delay;
2) the Controller selects the two switches at the ends of the target link, called $S1$ and $S2$;
3) the Controller sends to $S1$ two packets, $P1$ and $P2$, to be sent through the target link to $S2$;
4) the *P4* program installed in $S1$ instruments the packets by adding specific custom headers, that will be used to calculate the delay;
5) when the two packets reach $S2$, the *P4* program there adds the results of the measurement, again as custom headers, and makes $S2$ send them to the controller as Packet-In;
6) the controller gets the information about the delay on the link reading the packets.

With reference to Fig. 1, assuming the target link is $L1$, an example of measurement can be described as follows:

1) the controller sends two packets to $S1$ – let us call them P1 and P2 – with the information about their destination embedded into the custom header *PORT:* in the form of the port number of $S1$ towards $S2$;
2) when P1 goes to the egress queue, a time-stamp is taken and saved in a register entry of $S1$ for $L1$;
3) P1 is forwarded to $S2$, ;
4) when P2 arrives at the egress queue, $S1$ calculates the difference between the current time-stamp and the time-stamp of P1; in this way we get an estimate of the processing time in $S1$, which is called TP_1, and which is saved in the correspondent header field *DELAY* of P2;
5) P2 is also forwarded to $S2$;
6) $S2$ behaves in the same way as $S1$, in calculating the processing delay of the packets, but also knows it is the end node of the measurement from the custom header *WHICH*;
7) the processing time in $S2$, called TP_2 is calculated as in $S1$ and the total time P1 and P2 have been around are also calculated.
8) the delay due to the link is calculated as:

$$Delay = TT(P1, P2) - TP(S1) - TP(S2)$$

   where TT is the total time required by the packets to cross $S1$,$S2$ and the link $L1$, which can be seen as the transmission plus the propagation delay.
9) the calculated value is stored in the custom header *DELAY* of P2 before sending the packets to the SDN controller.

In this way we get a measure of the propagation delay on the link plus the queuing delay, if any.

C. Asymmetric Flow Detection

The third application we implemented is aimed at providing a different real time aid to the identification of possible DoS attacks to the network. DoS attacks are a serious threat to the availability of networks; SDN networks, if not properly deployed, could offer a larger attack surface, including the controller and not just the data plane nodes and terminals.

Typical countermeasures rely on traffic measures to identify anomalies in the traffic profiles, such as for instance [15], [16], [17], [18], [19]. Solutions to safeguard the controller from the risk to be overloaded have also been proposed in the literature [20].

Nonetheless the implementation of DoS detection in the SDN control plane calls for massive data transfer, storage, and analysis overheads [21]. Relying on modifications or extensions of the OpenFlow protocol is a far from effective option, requiring ad-hoc implementations and/or quite slow procedures to modify standards.

By programming the data plane, for instance with *P4*, it is possible to overcome such limitations, implementing a program which is able to produce an aggregated result of a possible DoS attack warning. In this work, we implemented

as an example a *P4* program for asymmetric flow detection. It calculates the ratio between the amount of incoming and outgoing traffic for a specific IP. The basic idea is that a very large asymmetry is an indication of a possible DoS attack.
Figure 1 shows a small example of how this works. The *P4* program on the switches maintains a registry entry for each IP address or IP class of interest. The program works on a predefined time scale, storing packets flowing in both directions between two sets of destinations. If a severe difference is detected between packets flowing one way and packet flowing the opposite way, this is considered a possible anomaly, causing the switch to send a *P4Runtime* packet-in to the Intrusion Detection System to raise a warning of a possible DoS attack. As an example, in the figure, two connections are monitored and only one of them, the one involving 10.10.10.1, is not malicious.
The threshold triggering the warning is implemented in a dynamic way, so that the controller can progressively track the level of the asymmetry and decide which action to perform. The threshold starts at an initial value, and every time the flow reaches it, the value is increased by a factor of 2.

IV. Conclusions

In this paper we explored a new frontier of network programming, with the specific goal of improving security-related functionalities. We made the case for the extensive usage of technologies such as the *P4* language, to exploit the strengths of a programmable data plane in SDN architectures. We discussed how it is possible to program forwarding devices with *P4* to achieve substantial benefits in terms of higher precision in measuring traffic parameters, more timely detection and mitigation of undesired traffic patterns, and more flexible distribution of traffic analysis tasks among the elements of the network. All of these improvements have a direct effect on security, by helping the controller to detect and react to threats to availability and to intrusions.

We showed that different modular P4 programs can be combined into an SDN architecture at the controller level. The principle was tested by coordinating three different metrics computed at the programmable data plane to achieve an overall IDS functionality.
We have implemented a test-bed to verify the viability of the described architecture. We are currently collecting results, and the preliminary findings for credible scenarios show the effectiveness of our approach, and improved efficiency compared to the cited similar proposals.

We argue that network programmability will represent a cornerstone of the network management field, and that integrated approaches of this kind, exploiting the synergy between programmable data planes and smart control planes, will play an important role for security-oriented solutions in the next future.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, p. 69–74, Mar. 2008.

[2] A. O. Basil, "A consistency based research: P4 versus openflow and the future of software defined networks," 2019.

[3] O. N. Foundation, "Openflow switch specification."

[4] O. A. Fernando, H. Xiao, and X. Che, "Evaluation of underlying switching mechanism for future networks with p4 and sdn (workshop paper)," in *Collaborative Computing: Networking, Applications and Worksharing* (X. Wang, H. Gao, M. Iqbal, and G. Min, eds.), (Cham), pp. 549–568, Springer International Publishing, 2019.

[5] 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," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[6] P. L. Consortium, "P4 language specification."

[7] A. Shaghaghi, M. A. Kaafar, R. Buyya, and S. Jha, "Software-defined network (sdn) data plane security: issues, solutions, and future directions," *Handbook of Computer Networks and Cyber Security*, pp. 341–387, 2020.

[8] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: detecting security attacks in software-defined networks.," in *Ndss*, vol. 15, pp. 8–11, 2015.

[9] A. Shaghaghi, M. A. Kaafar, and S. Jha, "Wedgetail: An intrusion prevention system for the data plane of software defined networks," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pp. 849–861, 2017.

[10] B. Lewis, M. Broadbent, and N. Race, "P4id: P4 enhanced intrusion detection," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–4, 2019.

[11] G. K. Ndonda and R. Sadre, "A two-level intrusion detection system for industrial control system networks using p4," in *5th International Symposium for ICS & SCADA Cyber Security Research 2018 5*, pp. 31–40, 2018.

[12] Y. Yetim, A. Bas, W. Mohsin, T. Everman, S. Abdi, and S. Yoo, "P4runtime: User documentation," 2018.

[13] D. Medhi and K. Ramasamy, "Chapter 4 - network flow models," in *Network Routing (Second Edition)* (D. Medhi and K. Ramasamy, eds.), The Morgan Kaufmann Series in Networking, pp. 114 – 157, Boston: Morgan Kaufmann, second edition ed., 2018.

[14] J. Raghavendran and J. Schormans, "Inferring delay variations using packet-pair probing techniques for network measurement."

[15] S. W. Shin, P. Porras, V. Yegneswara, M. Fong, G. Gu, M. Tyson, *et al.*, "Fresco: Modular composable security services for software-defined networks," in *20th Annual Network & Distributed System Security Symposium*, Ndss, 2013.

[16] M. Munir, S. A. Siddiqui, M. A. Chattha, A. Dengel, and S. Ahmed, "Fusead: unsupervised anomaly detection in streaming sensors data by fusing statistical and deep learning models," *Sensors*, vol. 19, no. 11, p. 2451, 2019.

[17] L. Dridi and M. F. Zhani, "Sdn-guard: Dos attacks mitigation in sdn networks," in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*, pp. 212–217, IEEE, 2016.

[18] D. Berardi, F. Callegati, A. Melis, and M. Prandini, "Technetium: Atomic predicates and model driven development to verify security network policies," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–6, 2020.

[19] D. Berardi, F. Callegati, A. Melis, and M. Prandini, "Security network policy enforcement through a sdn framework," in *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–4, 2018.

[20] D. Kotani and Y. Okabe, "A packet-in message filtering mechanism for protection of control plane in openflow networks," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 29–40, 2014.

[21] N. Z. Bawany, J. A. Shamsi, and K. Salah, "Ddos attack detection and mitigation using sdn: methods, practices, and solutions," *Arabian Journal for Science and Engineering*, vol. 42, no. 2, pp. 425–441, 2017.