P-SCOR: Integration of Constraint Programming Orchestration and Programmable Data Plane

Published Version:
Melis A., Layeghy S., Berardi D., Portmann M., Prandini M., Callegati F. (2021). P-SCOR: Integration of Constraint Programming Orchestration and Programmable Data Plane. IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, 18(1), 402-414 [10.1109/TNSM.2020.3048277].

Availability:
This version is available at: https://hdl.handle.net/11585/790409 since: 2021-01-22

Published:

DOI: http://doi.org/10.1109/TNSM.2020.3048277

A. Melis, S. Layeghy, D. Berardi, M. Portmann, M. Prandini and F. Callegati, "P-SCOR: Integration of Constraint Programming Orchestration and Programmable Data Plane," in IEEE Transactions on Network and Service Management.

The final published version is available online at: http://dx.doi.org/10.1109/TNSM.2020.3048277

P-SCOR: Integration of Constraint Programming Orchestration and Programmable Data Plane

Andrea Melis, Siamak Layeghy, Davide Berardi, Marius Portmann *Senior, IEEE*, Marco Prandini, and Franco Callegati *Senior, IEEE*

***Abstract*—In this manuscript we present an original implementation of network management functions in the context of Software Defined Networking. We demonstrate a full integration of an artificial intelligence driven management, an SDN control plane, and a programmable data plane. Constraint Programming is used to implement a management operating system that accepts high level specifications, via a northbound interface, in terms of operational objective and directives. These are translated in technology-specific constraints and directives for the SDN control plane, leveraging the programmable data plane, which is enriched with functionalities suited to feed data that enable the most effective operation of the "intelligent" control plane, by exploiting the *P4* language.**

***Index Terms*—SDN, Constraint Programming, *P4* , Programmable Data Plane, Security**

I. Introduction

The Internet has revolutionized our lives, allowing unprecedented possibilities to exchange information and enabling innovative ways to support many human activities. Arguably, there is no turning back. On the contrary, we observe a steady growth of the existing computing and communication infrastructures, and it is easy to forecast that this trend will continue for the foreseeable future. The recent events connected to the global pandemic just stressed this trend, moving many activities to the virtual world.

The increase in size and centrality of the network brings along formidable challenges. As usual new challenges require new engineering approaches, that emerged in the last decade, mostly based on virtualization and softwarization of network functions. [1]

Software Defined Networking (SDN) [2] and Network Function Virtualization (NFV) [3] are among the most notable examples. SDN aims at achieving full control and user plane separation (CUPS), with the goal of enhancing scalability and allowing more flexible forwarding policies, with cross protocol logic. NFV enables orchestrating virtualized infrastructures, with the goal of automating their life-cycle management. These technologies are also at the basis of the new architectural concepts that can be found in 5G, such as Radio Access Network (RAN) functional splitting and network slicing [4].

SDN and NFV open up new opportunities [5]; in particular the management plane could collect real-time information about the network status and react, by issuing suitable directives to the network orchestrators and/or software controllers. This *closed loop approach* promises to reduce both the duration of events jeopardizing the efficiency (or even the basic functionality) of the network, and the complexity of the necessary reaction. It is therefore particularly well-suited to properly address problems related to the security of the network, or to the guarantee of quality of service. In this manuscript we describe and demonstrate P-SCOR (Programmable data plane-Software-defined Constrained Optimal Routing), a methodology to implement management functions that can self-adapt to the changing network conditions and react to specific observed behaviors by exploiting the following innovative characteristics:

- *a flexible northbound interface*, through which network managers can describe high-level directives that state the goal to be reached, instead of having to provide the technical details of a plan to reach it;
- *a programmable data plane* to collect network measurements and state variables in real time;
- *an AI-based approach* to design network management strategies based on the collected network information;
- *integration* of the AI-driven control plane and the programmable data plane into a self-consistent network management architecture.

The goals of P-SCOR are achieved by integrating a high level orchestrator, based on Constraint Programming (CP), with an SDN control plane, based on ONOS — the Open Network Operating System[1], and a programmable data plane

Manuscript received June 2, 2020; revised October 25, 2020 and December 23, 2020; accepted December 24, 2020. Date of publication XXXXXX XX, 202X; date of current version December 28, 2020. The associate editor coordinating the review of this article and approving it for publication was Wolfgang Kellerer.(Corresponding author: Andrea Melis.)

This work was partially supported by Regione Emilia Romagna, program POR-FESR 2014-2020, Action 1.2.2, Project "I4S: Industria 4.0 Sicura (Cybersafe Industry 4.0)"

Andrea Melis, Davide Berardi, Marco Prandini and Franco Callegati are with the Department of Computer Science and Engineering at the University of Bologna, Italy, e-mail {a.melis, davide.berardi, marco.prandini, franco.callegati}@unibo.it

Siamak Layeghy and Marius Portmann are with the School of ITEE, The University of Queensland, Brisbane, Australia, e-mail siamak.layeghy@uq.net.au and marius@ieee.org

[1] https://opennetworking.org/onos/ visited on October 24, 2020.

exploiting the *P4* programming language[2]. Thanks to existing efforts of the open-source community such as the *P4Runtime* API [6], the proposed integration allows the CP orchestrator to gather real time information regarding the status and the operations of the network from dedicated *P4* programs in the data plane. These are then processed in order to compute solutions that are then implemented via the SDN control plane.

The advantages of the P-SCOR solution are discussed in the next section with reference to the state of the art. We claim that this line of research has theoretical and practical relevance, and that this manuscript provides a valuable addition to the knowledge in this field.

The paper is organized as follows. Section II presents a discussion of the relevant state-of-the-art literature. Section III offers a brief review of the two key technologies used in this work, namely Constraint Programming (CP) and programmable data plane with *P4*, while Section IV summarizes the characteristics of SCOR [7], the background project which laid the cornerstone for this work. Section V describes the main architectural components of P-SCOR and outlines the proof-of-concept scenario we used to validate the proposed architecture in Section VI. Section VII provides the numerical results of the tests, and conclusions are drawn in Section VIII.

II. Related works

To the best of our knowledge, the integration between high level programming, network control and forwarding programming proposed in this work has not been addressed in the past. Therefore there are no specific previous works to refer to. Nonetheless, it is important to briefly review some general concepts that, in general terms, inspired this work. First of all, let us consider that intelligent networks, self aware networks, and autonomic networking are well known terms that have been addressed by the scientific community for many years, in the quest for more efficient networks, as recently pointed out for instance in [8]. It is widely agreed that a major breakthrough in this direction was made with the *softwarization* of networks, thanks to the introduction of technologies such as SDN and NFV [9].

Efficiency is not the sole goal pursued by this evolution; more effective QoS management is another important goal [10], [11]. Furthermore, the capability of the network to self-adapt and react to specific working conditions and/or monitored parameters should become a major driver towards more secure networks, which is an increasingly critical concern, as discussed in recent works such as [12] and [13].

Indeed the centralized control plane approach of SDN allows the implementation of solutions to detect malicious, unwanted or unexpected traffic behaviors, as well as of suitable countermeasures at scale. The general scheme used, among others, in works such as [14], [15], and more recently in [16], exploits the OpenFlow protocol to monitor either packets or network parameters at the SDN controller and implements network-wide countermeasures via OpenFlow in case some malicious behavior is detected. Detection is achieved by matching traffic patterns and network parameters against some pre-defined sets of values, thresholds, etc. or by exploiting some form of formal verification method of behavioral models [17], [18]. While overall effective, these works exhibit a couple of weaknesses.

1) They have to rely on what the OpenFlow protocol allows, both in terms of network measurements and packet analysis. It could not be powerful enough to catch specific network behaviors, since the OpenFlow protocol was designed with forwarding control in mind, not network security.
2) They mainly focus on few known security issues, typically DDoS or similar ones, with the aim to provide higher effectiveness of detection or better automation of countermeasures implementation.

With reference to the former issue, related to network telemetry, it was recently demonstrated, for instance in [19], that more effective and scalable measurement of network parameters can be achieved by exploiting the *P4* language to program the data plane. Nonetheless, a holistic approach, aimed at exploiting the best characteristics of these solutions to provide very flexible network management and control, is still missing. This work proposes and demonstrates the integration of these techniques, with the purpose of overcoming the aforementioned limitations.

We propose to leverage the SDN control plane and the *P4* driven network telemetry to feed a Constraint Programming, high-level application, devoted to the analysis of the network and the design of countermeasures against identified anomalous behaviors. While currently AI is most commonly identified with its predominant sub-field, machine learning, CP rightfully belongs to its scientific domain, as testified, for instance, by being widely represented in the major conferences on the topic, such as the International Joint Conferences on Artificial Intelligence Organization (IJCAI). As briefly discussed in the following section, CP can automate the design of optimal reactions, instructing the controller upon changes in network conditions. On the switch side, the tight integration with a programmable data plane, exploiting the *P4* programming paradigm, allows the design of monitoring solutions that may go beyond what available with the OpenFlow protocol.

To the best of our knowledge there are very few works presented in the literature that exploit CP in combination with NFV and SDN, and none at all that provides an integrated approach with a programmable data plane as proposed and demonstrated here. In [20], CP modelling is used to design an intent-based approach to tackle the problem of Virtual Network Function placement in a multi-domain network. The idea to exploit CP modelling to provide a more user-friendly and technology-agnostic interface to the network manager is in line with the goals of our work, but it was pursued by Liu et.al. to a completely different end and, again, with no integration with the data plane. SCOR [7] also introduced CP as part of a broader SDN architectural refactoring. It is the cornerstone of this work, which enriches it with the integration of a programmable data plane.

[2] https://p4.org visited on October 24, 2020.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3048277, IEEE Transactions on Network and Service Management

III. Enabling Technologies and Background

In this section we briefly review the enabling technologies behind P-SCOR. As already stated, we claim that achieving the integration between these technologies is the main original contribution of this work.

A. Constraint programming

In CP, problems are solved by defining the requirements (constraints) to be applied to the problem variables and the goal is to find a solution that satisfies all the constraints. [21]. The main idea behind CP as stated by E. Freuder is: "Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."

Therefore CP is very suitable to implement a declarative northbound interface, that network operators or network users may exploit to express general constraints and goals in a way that is not bound to the specific underlying technology [22].

The essential step in modelling a real-world problem as a CP model, which can be solved using CP techniques, is to determine the decision variables of the problem and their relationship in terms of constraints, representing in very general terms the restrictions or cross bounds on values that all decision variables can have. Solving a CP model is the action of finding the values of the decision variables that simultaneously satisfy all the constraints. In this case CP problems are also called Constraint Satisfaction Problems (CSP) [23].

However, in many cases, there may be many subsets of the variables domains that satisfy the constraints. The solver program can be tuned to provide the first solution found, without any further processing, or all possible solutions. In addition, if an objective function can be defined on decision variables, the solver program can also be asked to provide a subset of values that maximises or minimises such a function. In this case CP problems are also called Constraint Satisfaction Optimisation Problem (CSOP)[23].

High level languages exists to state problems and constraints, like the open-source Minizinc language[3], as well as high performance solvers that can solve problems in a very efficient way [24]. Indeed the performance of a CP solver depends on its implementation but also on the description of the problem, with a mixed combination of computation efficiency and human optimization in problem design. The ease of implementation, simplicity, expressiveness and compatibility with many solvers has made MiniZinc the de-facto standard CP modelling language.

B. SDN and Programmable Data Plane

As already outlined, SDN aims at separating the network control plane from the forwarding plane. Controllers interact with forwarding devices via the so called SouthBound Interface. OpenFlow is the de-facto standard for southbound interfaces to date [25].

OpenFlow started simple, with the abstraction of a single table of forwarding rules that could match packets on a dozen header fields (MAC addresses, IP addresses, protocol, TCP/UDP port numbers, etc.) but, over the past five years, the specification has grown and OpenFlow is now more complex and feature-rich [26].

In the first version of OpenFlow, just 4 header features were available, now there are more than 50 [27]. This is not necessarily a positive trend. There is a widespread belief that, rather than repeatedly extending the OpenFlow specification, the future switches should support flexible mechanisms for parsing packets and matching header fields directly, allowing controller applications to leverage these capabilities through a common, open interface [28].

A good example has been shown in [29], where data-center network operators increasingly want to apply new forms of packet encapsulation (e.g., NVGRE, VXLAN, and STT), for which they resort to deploying software switches that are easier to extend with new functionality.

The idea of programmable switches has been around for a long time; in the past it was hindered by the performance degradation of programmable switches, due to the fact that the vendor chips had to adapt to different specifications instead of focusing on a subset of features and making them perform at their best. More recently, thanks to the advances in ASICs design, it was demonstrated [30] that programmable forwarding can be achieved at terabit/s speeds, thus making programmable switches comparable to legacy ones. These are the main motivations that inspired the development of a programming language for the data plane: the P4 *language*. *P4* is an open-source programming language which lets the end users describe how the switch should process the packets. It controls silicon processor chips in network forwarding devices, enabling a paradigm change from a "bottom-up" approach where fixed-function switches are built-in, to a programmable "top-down" approach where the user decides which functionalities to install. [31]

Basically *P4* has three main goals.

- Reconfigurability. The controller should be able to redefine the packet parsing and processing in the field.
- Protocol independence. The switch should not be tied to specific packet formats. Instead, the controller should be able to specify a packet parser for extracting header fields with particular names and types and a collection of typed match/action tables that process these headers.
- Target independence. The controller programmer should not need to know the details of the underlying switch. The *P4* compiler should translate the program features into target-dependent capabilities.

IV. SCOR

The proposed CP orchestration discussed in this paper is an extension of the SCOR system introduced in [7].

SCOR was implemented in Minizinc [32]. In addition to its pre-packaged solvers, e.g. Gecode [33], MiniZinc can utilise other available solvers, such as Jacop [34] and ECLiPSe [35].

SCOR provides a new SDN northbound interface with powerful CP-based abstractions that allows complex routing

[3] https://www.minizinc.org

problems to be expressed in only a few lines of code. More importantly, the problem only needs to be declared, while the solving of the corresponding constraints satisfaction problem is delegated to the powerful, general CP solvers, provided all data are available.

As an example, let us consider *minimum delay routing*, where we aim to find the path with the minimum end-to-end delay between two network nodes. As discussed in [36], this routing problem can easily be expressed and solved in SCOR. However, what is missing is the information about link delays in the network. This information is currently not provided by SDN controllers. This is mainly due to the limitation of OpenFlow, which does not provide the required functionality for data plane instrumentation and monitoring. With *P4* we can overcome this limitation, by utilizing its capability for in-band data plane monitoring. By integrating *P4* and SCOR, P-SCOR allows combining the benefits of both.

P4 has the ability to monitor critical link information from the data plane, and can provide this information to SCOR, which in turn uses it to very efficiently implement QoS routing applications, such as those discussed in detail in [7]. The focus of this paper is neither *P4* nor SCOR, but the integration of the two, which is demonstrated in the following sections via two use-case applications.

V. P-SCOR: Programmable Data Plane for Constraint Programming Orchestration

The P-SCOR architecture is summarized in Fig. 1, showing the three components (CP-based orchestrator, SDN controller, and *P4* -based programmable data plane) and their mutual relationships, focusing on one of the key contributions of this work, i.e. the communication channels between the components.

A. The key components

1) Programmable data plane with P4 *:* *P4* is not a protocol or device API for run-time control or configuration, i.e. once a *P4* program is deployed to a device, *P4* does not offer primitives, for example, to add or remove entries in match/action tables, or to read the value of a counter.

To carry out this kind of tasks, the *P4Runtime* API [6] has been developed to interact with the program. The main purposes of this new standard API are:

- enabling run-time control of *P4* -defined switches;
- defining program-independent interaction (the API does not change if the *P4* program is modified);
- enabling to push a new *P4* program without recompiling the switch software stack.

It adheres to a client-server model; the server resides in the data plane, integrated within the switch. A client integrated within a local or remote control plane interacts with the server to load the pipeline/*P4* program, write and read pipeline state (e.g. table entries, meters, groups, etc.) and sends/receives packets. *P4Runtime* uses a gRPC/protobuf-based language to define its own API, called p4runtime.proto. What *P4Runtime* needs in order to work is a set of specifications which are defined in the *P4* program and retrieved at compile time. The typical workflow of such process is defined in Fig. 2.

A significant part of our work regarded this layer of the architecture, in which we exploited *P4* to create an entity able to interact with the upper layer according to the specification required by *P4Runtime*. This *P4* program is then used as a wrapper for the specific data plane functionalities that were implemented and tested, as it will be explained in the following section.

2) SDN control plane: ONOS was chosen to implement the SDN control plane. It offers a number of features, besides the basic SDN controller capabilities, that qualify it as a real Network Operating System. ONOS was used as:

- the network controller, i.e. a component capable of controlling a whole network composed of several nodes and with a general topology;
- the communication channel between the data plane and the CP orchestrator.

This was achieved by exploiting the work of the ONOS *P4* brigade[4] which implemented a component that allows the integration of any *P4* program with ONOS.

The idea behind the *P4* brigade project is shown in Figure 2. The pipeline-agnostic application is any sort of ONOS application that may collect relevant data from the network and apply relevant control plane actions. The *Pipeliner* is a wrapper that gives to the ONOS application the capability to interact with *P4Runtime* without the need to know the details of the *P4* implementation (which would not be case for the pipeline aware application).

In this scenario the deployment of novel software functions can be seen as composed of three steps:

- a new *P4* program is deployed; it becomes immediately available thanks to the *P4Runtime* Standard API but it is not automatically integrated with the controller applications.
- a suitable pipeline is designed and implemented to guarantee a proper communication with the *P4* program (via *P4Runtime*).
- a new ONOS pipeline-aware application can be deployed at any time to access the *P4* program functionalities.

In ONOS we can perform this task defining what is called an "ONOS pipeconf". This component is essentially a regular application that can be loaded in ONOS at run-time and that, once loaded, registers the pipeline, which is the wrapper from the application to southbound API. Once registered, an ONOS application can add this registered pipeline to use a *P4Runtime*-capable device as shown in figure 2. This process gives ONOS the novel ability to create "wrappers" to *P4Runtime*.

We implemented all the necessary pipelines to allow the SCOR level to talk to the *P4* plane. We wrote the pipeconf that tells ONOS the mapping between the table IDs, header fields, and the instructions used in the FlowRule generated by the SCOR apps, and the *P4* table names, match fields, and actions as in the *P4* program, queried through *P4Runtime*.

[4] `https://wiki.onosproject.org/display/ONOS/P4+brigade` visited on May 27, 2020

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3048277, IEEE Transactions on Network and Service Management

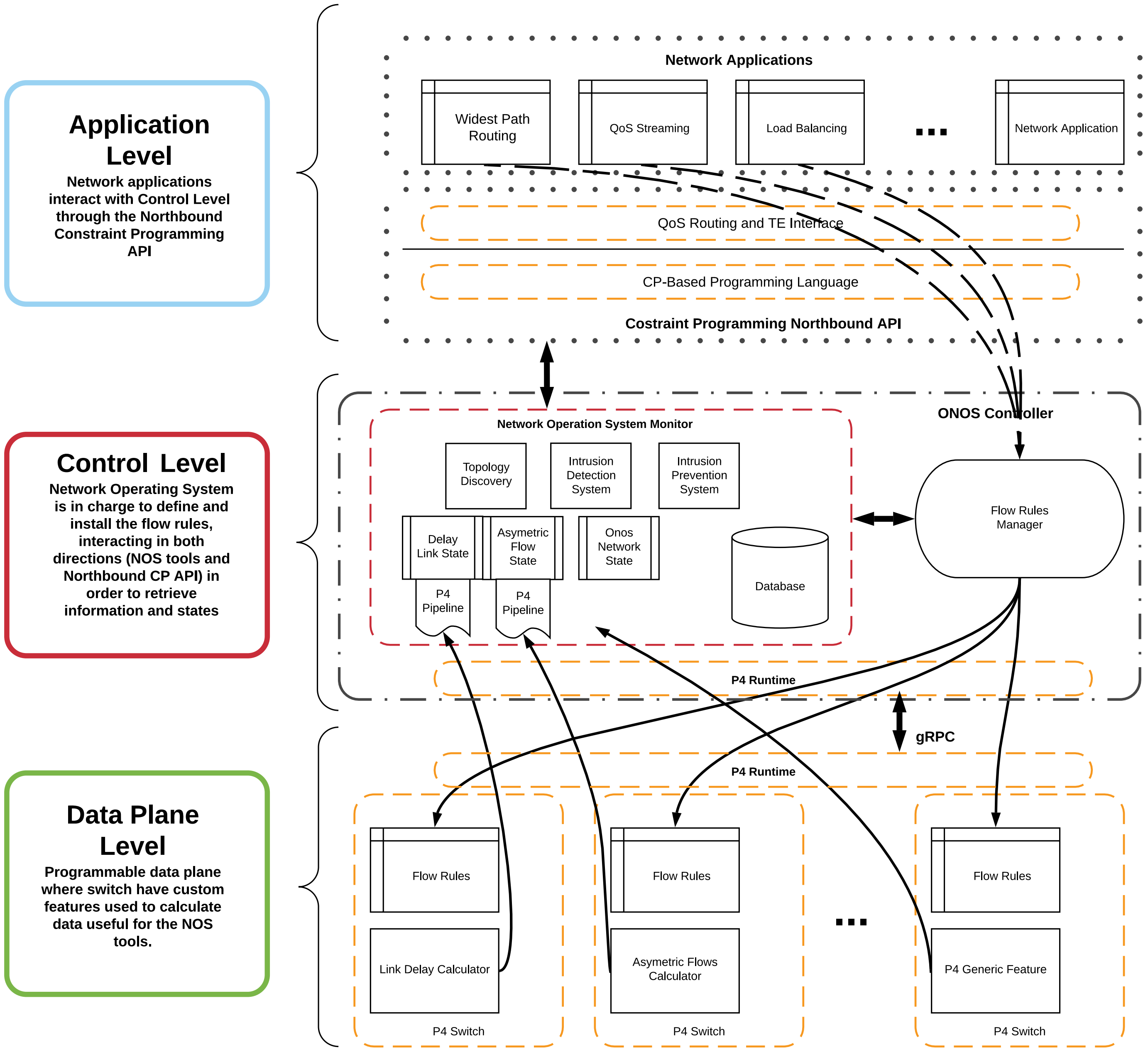

Application Level
Network applications interact with Control Level through the Northbound Constraint Programming API
Network Applications
Widest Path Routing
QoS Streaming
Load Balancing
...
Network Application
QoS Routing and TE Interface
CP-Based Programming Language
Costraint Programming Northbound API
Control Level
Network Operating System is in charge to define and install the flow rules, interacting in both directions (NOS tools and Northbound CP API) in order to retrieve information and states
ONOS Controller
Network Operation System Monitor
Topology Discovery
Intrusion Detection System
Intrusion Prevention System
Delay Link State
Asymetric Flow State
Onos Network State
P4 Pipeline
P4 Pipeline
Database
Flow Rules Manager
P4 Runtime
gRPC
P4 Runtime
Data Plane Level
Programmable data plane where switch have custom features used to calculate data useful for the NOS tools.
Flow Rules
Link Delay Calculator
P4 Switch
Flow Rules
Asymetric Flows Calculator
P4 Switch
...
Flow Rules
P4 Generic Feature
P4 Switch

Fig. 1: P-SCOR Overview of the three main component level

B. Putting it all together

If SCOR interacted only with the SDN control plane, as it happens in the original paper, it could not work on detailed run-time information, because the SDN controller would not have visibility of them. Most commonly, the SDN controller provides only aggregated information on the network behavior and high level information about topology etc.

On the other hand, were SCOR simply integrated with the *P4* layer via the *P4Runtime* interface, it could access run-time information only on a per-device basis, thus missing the overall network view.

By integrating ONOS with *P4* we enabled the collection of additional and more detailed information about network run-time properties in the SDN controller, which can pass them to the CP orchestrator implemented with SCOR. As a result, SCOR has access both to the high level network information provided by the SDN control plane and to the low level run-time and node-based information provided by the *P4* program.

Consequently, CP becomes applicable to solve problems that could not even be stated otherwise because of the lack of the needed variables and constraints. Moreover, SCOR may instruct the SDN controller to inject in the network specific packets, built by means of the *P4* program. The architectural enhancement we achieved, thus, extends its reach to the realm of action on the network flows; it is not limited to a better way of capturing and processing information.

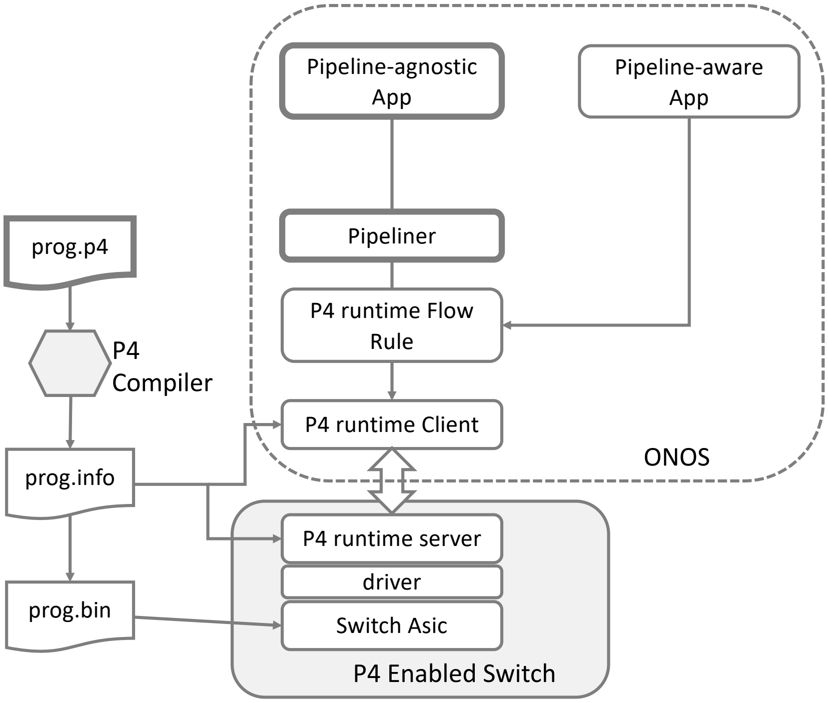


Fig. 2: A block diagram showing the interaction of the various components with the integration between ONOS and *P4* via *P4Runtime* and the ONOS pipeline. The components that have been originally implemented for this specific work are those highlighted with thicker lines.

VI. Examples of Applications: Link Delay and Asymmetric Flow Detection

A full test-bed of the P-SCOR architecture was implemented. The various components described above were all deployed appropriately, and in the data plane two *P4* applications were implemented to test the effectiveness of the proposed approach:

- link delay measurement;
- asymmetric flow detection (possible DoS detection).

The knowledge of the *link delay* is needed by many routing protocols, and in packet networks is indeed one of the main indicators of performance. In a conventional SDN network, analysing packets coming from the data plane is time- and resource-consuming especially if high accuracy is needed [37], [38]. Most SDN controllers do not encompass a built-in application able to do that, and also the original test-bed of SCOR used hard-coded (i.e. emulated, not real time) link delays for the tests.

Asymmetric flow detection can be used as a warning of potentially incorrect network behavior and also to trigger a remediation action (in our test application, raising a warning or rejecting every packet related to a network node identified as responsible). In SDN, the controller is informed only of the first packet of any new flow; this is enough to decide upon packet forwarding. Counting single packets per flow is not a viable feature of the control plane, as it would entail an unbearable overhead. With *P4* we are able to perform such action in the device directly and not at the control plane level, sending only the eventual warning to the CP orchestrator.

A. Link delay evaluation

The overall latency a packet experiences when traversing a network is due to many different contributions [39]:

- 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 packet's bits onto 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 a function of the distance between the network nodes). The processing and queuing delays are the main random components and also the ones that can be controlled – and ideally reduced – by enhanced network protocols, scheduling policies, etc.. For example, MultiProtcol Label Switching [40] was introduced to implement dedicated virtual circuit connection (the Label Switched Paths) and reduce the time needed to take the packet routing decisions.

In general, the delay measurement can be [41]:

- **passive**, i.e. non-intrusive and based on capturing packets, in order to store and collect information from various fields within the packet header.
- **active**, i.e. by injecting probe packets, measuring the performance they experience, and taking it as representative of the performance of all the traffic.

The measurement strategy we opted for belongs to the active category. In standard SDN, active measurement is not

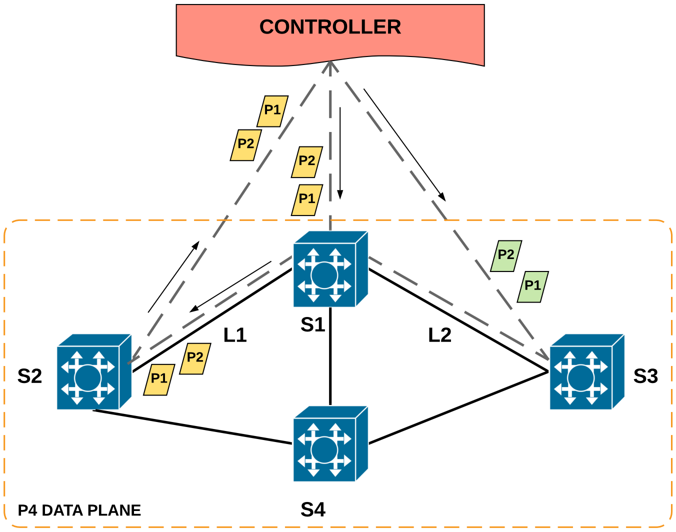


Fig. 3: Delay Link program work-flow

practical, because the switches act just as forwarders and all the intelligence is in the controllers. Therefore any real life packet measure made at the controller is affected by the switch-controller delay that may impair the measure reliability. *P4* shows all its potential in this scenario, because it allows us to perform an active measurement in a simple and automatic way at the switch level.

Since the controller knows the network topology of the data plane, we chose to use a packet-probing technique to estimate the delay of each link of the network, following [42]. The idea is simple:

- the controller targets a link to measure the average delay;
- the controller picks the two switches at the ends of the target link, called $S1$ and $S2$;
- the controller sends to $S1$ two packets, $P1$ and $P2$, to be sent through the target link to $S2$;
- the *P4* program installed in $S1$ instruments the packets by adding specific custom headers, that will be used to calculate the delay;
- 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;
- the controller gets the information about the delay on the link reading the packets.

The process is summarized in Fig. 3.

The custom headers added to the packets are called:

- *WHICH*: a header to identify the packet ordering;
- *WHERE*: a value to specify whether the switch which added this header is the source or the sink of the link;
- *PORT*: the port through which the source switch has to forward the packet to reach the sink of the link;
- *TIME*: the current time-stamp;
- *DELAY*: a time-stamp that is used with a different purpose on P1 and P2, in P1 it is used to calculate the total end-to-end delay of the packet, in P2 it is used to calculate the processing delay to be subtracted from the total delay of P1 to estimate just the link delay.

With reference to Fig. 3, 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:

$$Delay = TT(P1, P2) - TP(S1) - TP(S2) \qquad (1)$$

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.

B. Asymmetric flow detection

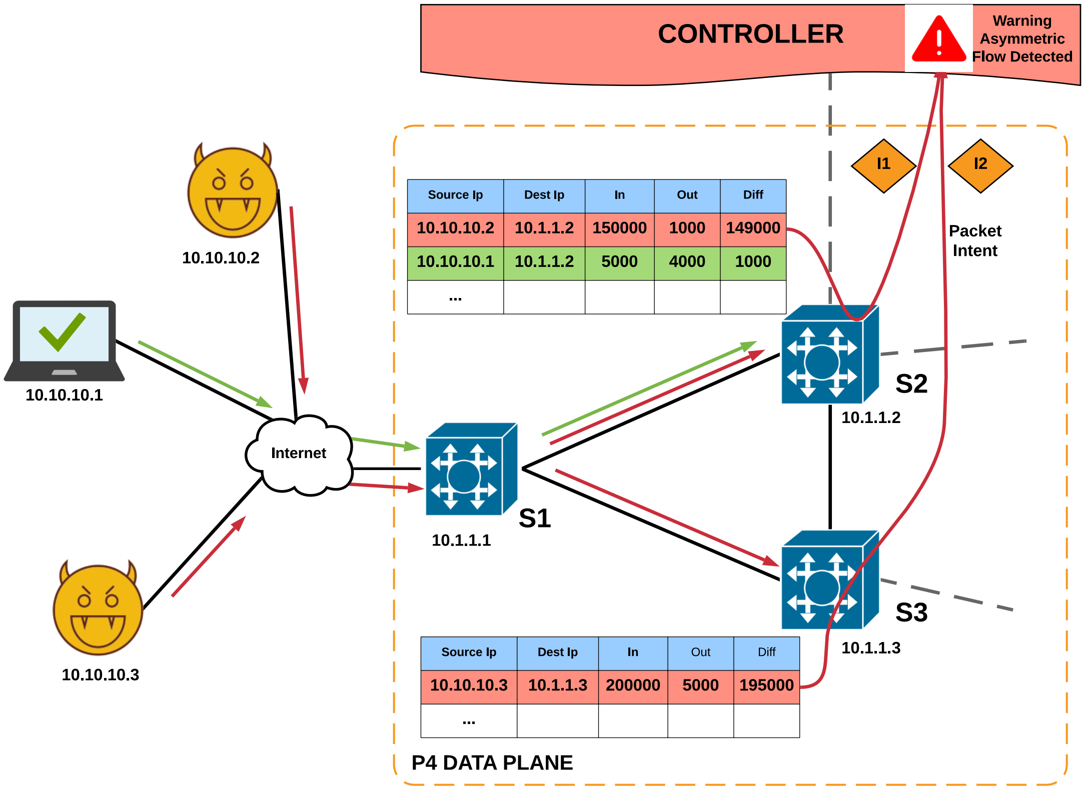
The second application we implemented, as an additional example of the effectiveness of the P-SCOR approach, aimed at providing a real time aid to the identification of possible DoS attacks to the network. This information is passed to the CP orchestrator that will make decisions about possible countermeasures, with an approach similar to the QoS routing already mentioned above.

It is well known that DoS attacks are a serious threat to the availability of networks; even more so in SDN networks where a DoS attack can be mounted against the controller and not just towards the network nodes and terminals.

Typical countermeasures rely on traffic measures to identify anomalies in the traffic profiles, such as for instance [43], [44], [45], [18], [46]. Solutions to safeguard the controller from the risk to be overloaded have also been proposed in the literature [47].

Nonetheless the implementation of DoS detection in the SDN control plane involves data storage and analysis which consumes memory, requires complex computations, and presents the risk of false-positives and false negatives [48]. Additionally, as discussed in Section II, relying on modifications or extensions of the OpenFlow protocol is a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3048277, IEEE Transactions on Network and Service Management



Source Ip	Dest Ip	In	Out	Diff
10.10.10.2	10.1.1.2	150000	1000	149000
10.10.10.1	10.1.1.2	5000	4000	1000
...				

Source Ip	Dest Ip	In	Out	Diff
10.10.10.3	10.1.1.3	200000	5000	195000
...				

Fig. 4: Asymmetric flow detection, example of work-flow. Three data flows are active and the *P4* application is active in all switches. In S2 and S3, anomalies are detected since some flows show sensible asymmetries between packet numbers observed in the two directions.

far from effective option, requiring ad-hoc implementations and/or long lasting 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 entity. The basic idea is that a very large asymmetry is an indication of a possible DoS attack.

Figure 4 shows a schematic 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 pre-defined 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 controller to raise a warning of a possible DoS attack. As an example, in the figure, three 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.
In Listing 1, a small block code of the *P4* application is shown, performing the calculations and the checks for the threshold.

Listing 1: Threshold computation and checks using *P4* .

```
apply
  if (hdr.ipv4.isValid()) {
    ipv4_lpm.apply();
    ...
    window.read(last_time,flow);
    threshold.read(currentThreshold,flow);
    // first time initialize
    ...
    intertime = standard_metadata.ingress_
    global_timestamp - last_time;
    window.write((bit<32>)flow,
      standard_metadata.
      ingress_global_timestamp);
    // check window
    if(intertime > WINDOW){
      restore_flow(flow,flow_opp);
    }
    last_seen.read(last_pkt_cnt,flow);
    last_seen.read(last_pkt_cnt_opp,flow_opp);
    tmp = last_pkt_cnt - last_pkt_cnt_opp + 1;

    if(tmp < (bit<48>)currentThreshold) {
      get_inter_packet_gap(last_pkt_cnt,flow);
    }
    else{
      // threshold is reached, drop it
      // (send packetin to the controller)
  if(currentThreshold > 1000){
     drop();
  }
  // else i increase your threshold,
  // and restore the flow
  else {
   threshold.write(flow,
    currentThreshold+200);
   // increase your threshold,
   // restore the flow
   restore_flow(flow,flow_opp);
  }
 }
}
```

VII. Test Bed and Experimental Results

P-SCOR was implemented in a virtualized environment on a server with Ubuntu Linux 18.04, 8 GB of RAM and dual core CPU.

The components used were:

- ONOS controller, version 1.14, the most recent version supported by the ONOS brigade community;
- *P4* version 16;
- Bmv2[5] as switches supporting *P4*;
- mininet[6] to implement virtual network topologies;
- some well known tools for traffic generation, such as MiniCPS [49], iperf[7], hping[8], and the Python scapy library[9] to forge custom packets.

With this test bed it was possible to run and test P-SCOR with the aim to:

- verify the correct functionalities of the *P4* program implementation and of the architecture as a whole;
- verify the overhead introduced by our solution;
- compare the P-SCOR solution with competing, existing ones.

The tests were performed on a simple ring network topology with three switches, unless otherwise specified.

At first we compared the performance of the switches with the *P4* programs used by P-SCOR with conventional switches and with OpenFlow controlled switches. The goal is to check how much overhead (if any) is introduced by the P-SCOR components.

Following [50] we performed two different types of tests. We measured the average Round Trip Time (RTT) to transmit a set of ICMP packets of size 512 and 8192 Bytes. Each of these tests were performed 10 times for each set of packets, and the results are the average of the 10 rounds. The aim of such test is to measure the forwarding-behaviour performances of each switch. Is important to mention that for tests on the OpenFlow switches we had to limit the bandwidth to 18 Mbit/s. This because it was the maximum amount of stable bandwidth that we were able to obtain for the Bmv2 *P4* switch and we needed a consistent environment for the evaluation tests.

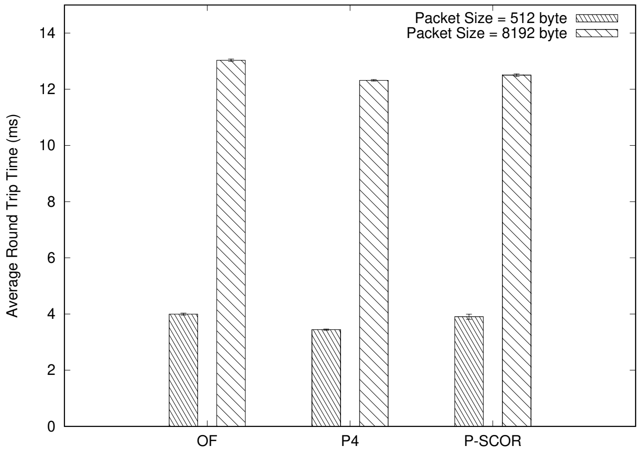


Fig. 5: Comparison of the forwarding performance of an OpenFlow switch, a *P4* enabled switch and a *P4* enabled switch running the P-SCOR related programs.

These results are shown in Fig. 5, where the confidence interval of the measure is also shown as error bar on top of the histograms bars. The confidence interval is very small, meaning that the measures are very consistent. They show the rather obvious fact that forwarding long packets takes longer than forwarding short ones, but they also show that the switch exhibiting the worst performance, albeit for just a few ms, is the OpenFlow-based one.

This was an expected result, since the basic *P4* switch is the lighter one; it implements only the essential forwarding reactive behavior and it does not have all the tables and features that a standard OpenFlow switch has. Otherwise our modified version of the *P4* delay link switch includes only the new feature to process special crafted packets, and the performances are slightly worse than those of the basic *P4* switch but a bit better than those of the OpenFlow one.

This is confirmed by the histogram in Fig. 6, where we measured the Total Transmission Time of bursts of packet of increasing length, from 20 to 200, both for packet size 512 and 8192 Bytes and in the same conditions as before. Again, the performances of the three variants of the switches are very similar; the times needed to send the train of packets are similar as well, since the transmission delay becomes negligible, when compared to the sum of the propagation delay of the all the packets in the burst.

The same comparison has been performed on the asymmetric flow detection switch. The results very closely matched the ones we just illustrated for the delay case, so we deemed not necessary to include an additional graph.

These first experiments therefore allow us to conclude that the performance of our switch is in line with the existing state of the art, despite the fact that we have introduced some important changes.

The second set of tests was used to validate the correct integration of the three layers by means of the two applications implemented in *P4* . The link delay was supposed to be a key input even in the original implementation of SCOR, with the goal to perform QoS-based routing strategies to minimize the

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

[6] http://mininet.org/

[7] https://iperf.fr/iperf-doc.php

[8] http://www.hping.org/

[9] https://scapy.net/

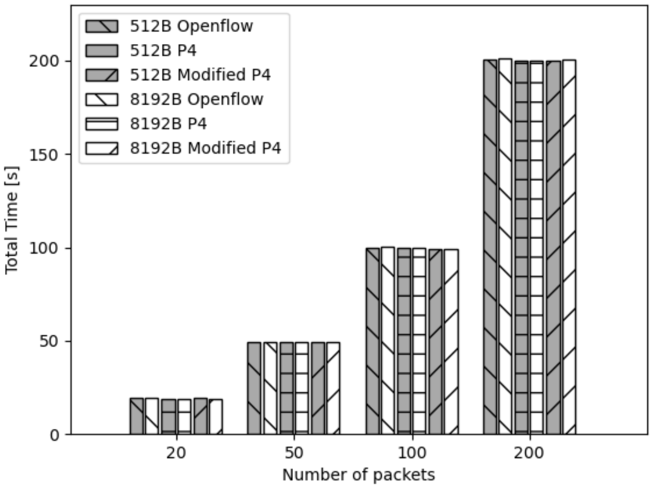


Fig. 6: Comparison of the total time transmission time of a burst of packet (with burst size from 20 to 200)

TABLE I: The link delay calculated by the *P4* program.

Number of Measurements	100
Interval Measurements	1000ms
Min value	6
Max value	110
Average delay	37.42ms
Variance	5676.79

overall packet latency, but it was not implemented with real time measurement. The asymmetric flow detection is used as an input to trigger remediation actions, in this case with packet drop as explained later. The CP orchestration programming will not be discussed here because it follows what presented in [7].

Table I summarizes the first set of tests on link delay measurement. We sent a set of four consecutive pairs of packet probes every second, and we measured the delay of the link. In this case we expected a small delay just due to the packet transfer in the virtualized environment. The average measured delay was 37.42 ms and is taken as a reference for the subsequent tests, in which a delay was introduced on the link, increasing from 1 second to 20 seconds in 1-second steps. The goal was to check that the application could keep measuring the correct delay. The results are shown in Fig. 7.

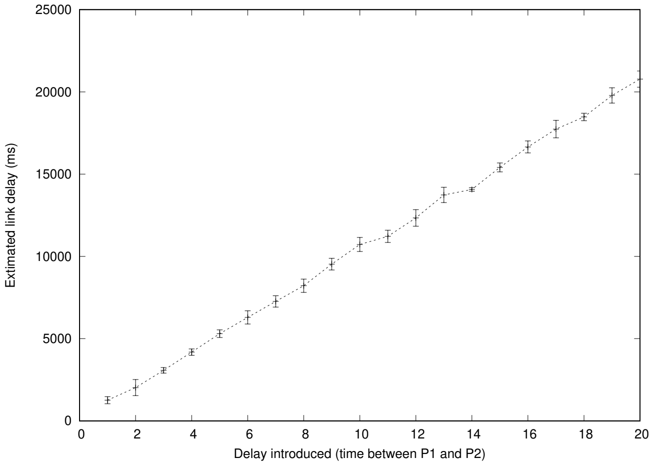


Fig. 7: The link delay calculated by the *P4* program, with an increased delay on the link from 1 to 20 s. The 95% confidence interval is plotted with the average. This confidence interval is quite good and was achieved with 5 experiments per point.

Here every point of the graph represents the average of 5 tests, therefore the 95% confidence interval of the measure is also shown, confirming that the measure is rather accurate.

The tests on asymmetric flow detection were also run in a similar way. In this case, providing evidence of the effectiveness of the operations of P-SCOR is slightly more complicate. We set up the *P4* application performing the tests on the flow asymmetry with a threshold T. A time window of $W = 15$ seconds was set. The time in the example is measured as a multiple of W, therefore $t = 1W$ means $T = 15$ seconds. Every W we use `iperf` to send $N(t)$ packets, a number increasing with t. These packets emulate the asymmetry of the bidirectional connection, i.e. the difference between the number of packets flowing in one direction and the number of packets flowing in the other direction. A warning of asymmetric flow detection is sent, as specified in Section VI, every time the asymmetry of the flow hits the intermediate threshold.

The threshold T starts at 300 packets per W; it is increased by 100 packets at every window in which no warning occurs, up to a value of $T_M = 600$ packets. The CP orchestrator, upon receiving a warning, simply asks the control plane to drop the packets of the flow; the rule is then programmed in the switches.

When the number of packets in the flow decreases, the threshold T is brought back to the starting value and packets are allowed to cross the network again.

This behavior is shown in the example of Fig. 8, where we plotted three curves as a function of time. The threshold T is the dotted curve that increases, reaches its maximum and then decreases again when the traffic falls back within the pre-set limits. The number of packets N sent per W is the continuous line, and goes up from 100 to more than 600. Then it drops to 0 and stays at 0 for a minute ($4W$), which is the flat section of the continuous curve. Then `iperf` starts sending packets again as before. After a minute also T is reset as shown by the dotted curve. The dashed curve in the figure is the bandwidth B used by the flow as measured by `iperf` . As it can be seen, when the number of packets sent overcomes the threshold T, then B drops to 0, meaning that the packets are dropped and `iperf` cannot see any capacity available for the traffic flow. Otherwise, when the network behaves normally, B is constant and equal to the link capacity.

The specific values of these quantities are just an example, to show how the *P4* application works and interacts with the CP orchestrator; in the same way, dropping packets is just one of the possible countermeasures to be taken.

The last set of tests aimed at comparing the proposed solution with similar ones from the literature. The approach

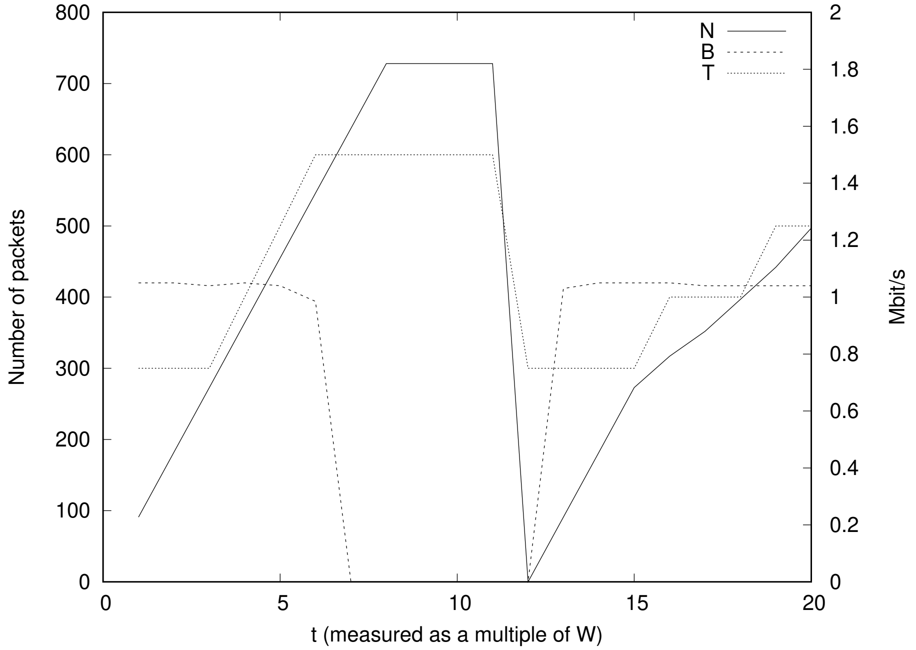


Fig. 8: A graphical example of the application for asymmetric flow detection. The adopted policy is to drop until the number of packet in the overloaded direction decreases. The figure plots the link bandwidth used by the source, the number of sent packets and the threshold. When the threshold is reached and overcome the packets are dropped (used bandwidth goes to 0).

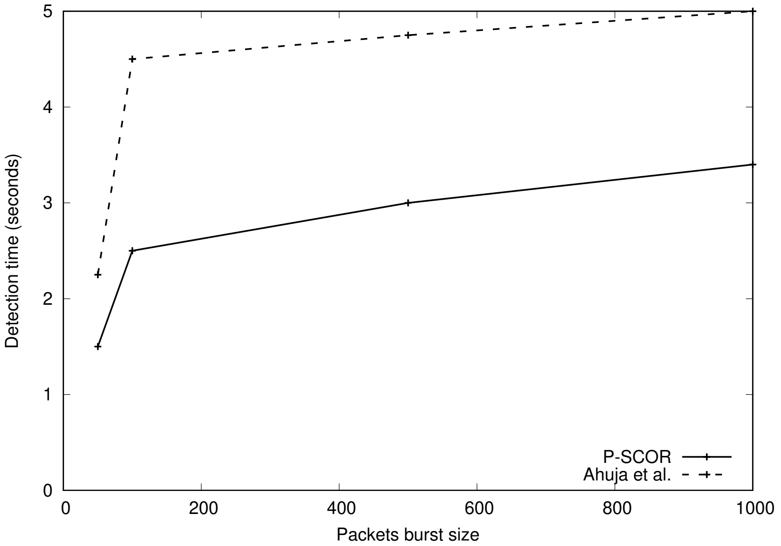


Fig. 9: Detection time of traffic flows very asymmetric of P-SCOR, compared with the detection time reported in [51]. The threshold of detection is set to 100 packets per window.

presented in [51] is the closest to ours: there, a ratio between packet entering and leaving the switch is computed, in order to look for an asymmetry of the traffic flow, but the authors had to implement the calculation at the controller level. To the best of our knowledge, there is no solution like the one we proposed, in which the asymmetric flow calculation is performed directly and dynamically at the data plane level.

Graph 9 presents a comparison between our work and the results reported in [51]. We simulated the same attack traffic with the same ratio threshold of 100 packets per second. The network topology for this test is the same used for the previous ones, i.e. three switches connected in a ring.

What we compare is the time taken to detect the traffic as malicious, before and after reaching the threshold. The graph shows that when the attack rate reaches the threshold, the detection time stabilizes. Comparing the P-SCOR solution with that proposed in [51], we can see the clear advantage coming from not having to send packets to the controller level for ratio computations.

VIII. Conclusions

In this paper we described P-SCOR, an innovative and original architecture for network management, based on the integration of Constraint Programming with SDN at the control plane level and a Programmable Data Plane exploiting *P4*. The core of this work has been the creation of the SDN-based infrastructure that enables the communication between the Constraint Programming Orchestrator, coordinating high level applications through the northbound interface, and the applications written in *P4* and deployed on data plane elements.

To the best of our knowledge there are no solutions demonstrating this kind of integration and of cooperation between planes.

We showcased the benefits and the strength of such architecture by implementing two specific study cases of practical relevance for QoS and network security applications: the real time measurement of the link delay and the detection of asymmetric traffic flows. To validate the P-SCOR proposal, we implemented a test-bed supporting active measurements, which can be considered a further contribution to the field.

The tests confirmed the effectiveness of the proposed approach. Further work will be devoted to systematize and implement additional classes of *P4* applications, aimed at supporting the CP-Orchestrator with sets of enriched data plane information needed to design controls with the maximum efficiency and effectiveness.

References

[1] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, “Clouds of virtual machines in edge networks,” *IEEE Communications Magazine*, vol. 51, no. 7, pp. 63–70, 2013.

[2] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, and S. Shenker, “Rethinking enterprise network control,” *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1270–1283, 2009.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, “Network function virtualization: State-of-the-art and research challenges,” *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[4] D. Borsatti, G. Davoli, W. Cerroni, and F. Callegati, “Service function chaining leveraging segment routing for 5g network slicing,” in *2019 15th International Conference on Network and Service Management (CNSM)*, pp. 1–6, 2019.

[5] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, and A. Manzalini, “Sdn for dynamic nfv deployment,” *IEEE Communications Magazine*, vol. 54, no. 10, pp. 89–95, 2016.

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

[7] S. Layeghy, F. Pakzad, and M. Portmann, “Scor: Constraint programming-based northbound interface for sdn,” in *2016 26th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 83–88, Dec 2016.

[8] E. Gelenbe, J. Domanska, P. Fröhlich, M. P. Nowak, and S. Nowak, “Self-aware networks that optimize security, qos, and energy,” *Proceedings of the IEEE*, vol. 108, no. 7, pp. 1150–1167, 2020.

[9] I. Farris, T. Taleb, Y. Khettab, and J. Song, “A survey on emerging sdn and nfv security mechanisms for iot systems,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 812–837, 2019.

[10] G. Chandwani, S. Behera, and G. Das, “Delay-aware control plane virtual topology design of software defined-elastic optical network,” *IEEE Transactions on Network and Service Management*, pp. 1–1, 2020.

[11] S. Tomovic, W. Cerroni, F. Callegati, R. Verdone, I. Radusinovic, M. Pejanovic, and C. Buratti, “An architecture for qos-aware service deployment in software-defined iot networks,” in *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pp. 561–567, 2017.

[12] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. Pastor Perales, “Adding support for automatic enforcement of security policies in nfv networks,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 707–720, 2019.

[13] B. Martini, P. Mori, F. Marino, A. Saracino, A. Lunardelli, A. L. Marra, F. Martinelli, and P. Castoldi, “Pushing forward security in network slicing by leveraging continuous usage control,” *IEEE Communications Magazine*, vol. 58, no. 7, pp. 65–71, 2020.

[14] A. Lara and B. Ramamurthy, “Opensec: Policy-based security using software-defined networking,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 1, pp. 30–42, 2016.

[15] Seungwon Shin and Guofei Gu, “Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?),” in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–6, 2012.

[16] N. Ahuja and G. Singal, “Ddos attack detection prevention in sdn using openflow statistics,” in *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pp. 147–152, 2019.

[17] A. Melis, D. Berardi, C. Contoli, F. Callegati, F. Esposito, and M. Prandini, “A policy checker approach for secure industrial sdn,” in *2018 2nd Cyber Security in Networking Conference (CSNet)*, pp. 1–7, 2018.

[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] J. Sonchack, O. Michel, A. J. Aviv, E. Keller, and J. M. Smith, “Scaling hardware accelerated network monitoring to concurrent and dynamic queries with* flow,” in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 823–835, 2018.

[20] T. Liu, F. Callegati, W. Cerroni, C. Contoli, M. Gabbrielli, and S. Giallorenzo, “Constraint programming for flexible service function chaining deployment,” *arXiv preprint arXiv:1812.05534*, 2018.

[21] F. Rossi, P. V. Beek, and T. Walsh, *Handbook of constraint programming*, vol. 1. UK: Elsevier, 2006.

[22] E. C. Freuder, “In pursuit of the holy grail,” *Constraints*, vol. 2, no. 1, pp. 57–61, 1997.

[23] R. Bartak, “Constraint programming: In pursuit of the holy grail,” in *In Proceedings of the Week of Doctoral Students (WDS99 -invited lecture)*, vol. Part IV, (Prague, Poland), pp. 555–564, MatFyzPress, 1999.

[24] T. Bridi, A. Bartolini, M. Lombardi, M. Milano, and L. Benini, “A constraint programming scheduler for heterogeneous high-performance computing machines,” *IEEE transactions on parallel and distributed systems*, vol. 27, no. 10, pp. 2781–2794, 2016.

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[26] B. Butler, “What p4 programming is and why it’s such a big deal for software defined networking.,” 2016.

[27] “Openflow switch specification, version 1.5.1 (protocol version 0x06),” March 2015.

[28] E. Kaljic, A. Maric, P. Njemcevic, and M. Hadzialic, “A survey on data plane flexibility and programmability in software-defined networking,” *IEEE Access*, vol. 7, pp. 47804–47840, 2019.

[29] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[30] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, “Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics,” in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 15–28, 2017.

[31] B. Butler, “What p4 programming is and why it’s such a big deal for software defined networking,” Jan 2017.

[32] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, “Minizinc: Towards a standard cp modelling language,” in *International Conference on Principles and Practice of Constraint Programming*, pp. 529–543, Springer, 2007.

[33] C. Schulte, G. Tack, and M. Z. Lagerkvist, “Modeling and programming with gecode,” *Schulte, Christian and Tack, Guido and Lagerkvist, Mikael*, vol. 1, 2010.

[34] K. Kuchcinski and R. Szymanek, “Jacop-java constraint programming solver,” in *CP Solvers: Modeling, Applications, Integration, and Standardization, co-located with the 19th International Conference on Principles and Practice of Constraint Programming*, 2013.

[35] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[36] S. Layeghy, F. Pakzad, M. Portmann, *et al.*, "A new qos routing northbound interface for sdn," *Journal of Telecommunications and the Digital Economy*, vol. 5, no. 1, p. 92, 2017.

[37] W. Queiroz, M. A. Capretz, and M. Dantas, "An approach for sdn traffic monitoring based on big data techniques," *Journal of Network and Computer Applications*, vol. 131, pp. 28 – 39, 2019.

[38] T. Alharbi, S. Layeghy, and M. Portmann, "Experimental evaluation of the impact of dos attacks in sdn," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, 2017.

[39] 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.

[40] B. S. Davie and Y. Rekhter, *MPLS: technology and applications*. Morgan Kaufmann Publishers Inc., 2000.

[41] V. Mohan, Y. J. Reddy, and K. Kalpana, "Active and passive network measurements: a survey," *International Journal of Computer Science and Information Technologies*, vol. 2, no. 4, pp. 1372–1385, 2011.

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

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] D. Antonioli and N. O. Tippenhauer, "Minicps: A toolkit for security research on cps networks," in *Proceedings of the First ACM workshop on cyber-physical systems-security and/or privacy*, pp. 91–100, 2015.

[50] I. Z. Bholebawa and U. D. Dalal, "Design and performance analysis of openflow-enabled network topologies using mininet," *International Journal of Computer and Communication Engineering*, vol. 5, no. 6, p. 419, 2016.

[51] N. Ahuja and G. Singal, "Ddos attack detection & prevention in sdn using openflow statistics," in *2019 IEEE 9th International Conference on Advanced Computing (IACC)*, pp. 147–152, IEEE, 2019.

Andrea Melis is an Adjunct Professor and a Post-Doc Researcher at Department of Computer Science and Engineering at University of Bologna. His research focuses on aspects of computer security related to innovative software architectures. In particular, his actual research activity aims to study, design and implement innovative solutions that allow to improve the safety and operational robustness of connected production industrial network and devices for cyber-security contexts.

Siamak Layeghy is a research fellow at the School of Information Technology & Electrical Engineering at the University of Queensland, Brisbane, Australia. He received his PhD in Software Defined Networking form the University of Queensland. His research interests include Software Defined Networks, Cybersecurity, AI and Machine Learning.

Davide Berardi is a Ph.D. Student of Computer Science and Engineering at Alma Mater Studiorum – Università degli Studi di Bologna, Bologna, Italy. He received a Master Degree in Computer Science at the same University in 2016. His research interests focus on Computer Security, Cyber Security Red Teaming and Network Virtualization.

Marius Portmann received his PhD in Electrical Engineering from the Swiss Federal Institute of Technology (ETH, Zurich) in 2002. He currently is an Associate Professor at The University of Queensland, Australia. His research interests include general networking, in particular SDN, wireless networks, pervasive computing and cyber security.

Marco Prandini received the Master and Ph.D. degrees in electronic and computer engineering from the University of Bologna, Italy in 1995 and 2000 respectively. He is currently a Research Associate at the Department of Computer Science and Engineering of the same university. His research activities started in the field of public-key infrastructures and later moved to subjects related to the security of microservice-based architectures, software-defined networks, IoT, and industrial control systems.

Franco Callegati is an associate professor at the University of Bologna, Italy. His research interests are in the field of teletraffic modeling and performance evaluation of telecommunication networks. He is currently working on performance evaluation and experimental validation of SDN/NFV–based networking solutions and 5G. He has been active in EU-funded research projects since FP4. He is Senior Member of the IEEE.