P4 Edge Node enabling Stateful Traffic Engineering and Cyber Security [Invited]

F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti , F. Cugini and P. Castoldi

***Abstract*—Next-generation edge nodes interfacing innovative IT clusters, 5G fronthaul and IoT gateways to the optical metro/core network will require advanced and dynamic online Quality of Service (QoS) per-flow traffic treatment, assuring for example ultra-low latency requirements. However, current Software Defined Networking (SDN) implementations (e.g., OpenFlow) do not support forwarding procedures based on network state, profile variations and the history of flow statistics at the node level. Currently, such procedures require the intervention of the SDN controller, leading to scalability issues and additional latency in the data plane forwarding. Moreover, severe security challenges are expected to affect such nodes threatening IT resources. Thus, increasing bandwidths will require direct deep packet inspection avoiding the involvement of the SDN controller, as performed currently, or dedicated and costly security systems.**

This paper leverages on the potential of the P4 open source language, recently introduced by the inventors of OpenFlow, to program the data plane structure and behavior of an SDN switch. P4 is able to instantiate custom pipelines and stateful objects, enabling complex workflows, user-defined protocols/headers and finite state machines enforcement. Moreover, P4 allows portable implementations over different hardware targets, thus opening the way to open source fully-programmable devices.

Special effort is dedicated to motivate and apply P4 within a multi-layer edge scenario, proposing the architecture and the applicability of an SDN P4-enabled packet-over-optical node. Moreover, three specific multi-layer use cases covering dynamic TE (e.g., traffic offload and optical bypass) and cyber security (e.g., DDoS port scan) are discussed and addressed through P4-based solutions. Experimental evaluations have been conducted over a multi-layer SDN network exploiting reference P4 software switches (i.e., BMV2) and Field Programmable Gate Array (FPGA) at 10 Gigabit Ethernet optical interfaces. Extensive results report effective dynamic TE and cyber security mitigation enforcement at P4 switches without any controller intervention, showing excellent scalability performance and overall latencies practically in line with current commercial OpenFlow switches.

***Index Terms*—Edge Node, Traffic Engineering, Flowlet, P4, SDN, Multi-Layer, Optical Bypass, Token Bucket, Cyber Security, SYN Flood, BMV2, NetFPGA.**

I. Introduction

The future convergence of next-generation wireless, wired and IT infrastructure is paving the way to the deployment of enhanced edge nodes [2] [3]. In particular, edge packet-over-optical aggregation nodes are emerging to interface heterogenous local segments such as 5G Radio Access Networks (RAN) fronthauling, Internet of Things (IoT) gateways and IT Cloud/Fog clusters to the Metro-core transport network. These innovative technologies and platforms are expected to provide extreme Quality of Service (QoS) differentiation and high throughput at the same time, exploiting the Software Defined Networking (SDN) control plane and the orchestration of services and network resources across different segments, from the access to the optical backbone [4] [5]. However, several issues may affect the utilization of current SDN solutions at the edge where a plethora of traffic flows with high throughput and dynamic profiles will need advanced Traffic Engineering (TE)-based treatment, due to stringent QoS constraints. Moreover, besides control and QoS, edge resources require to be secured against online cyber security attacks.

To address TE, packet-layer substantial innovations such as Segment Routing [6] [7] also enabling SDN/NFV Service Chaining [8] and Network Slicing [9] have been proposed to automatically steer per-service traffic slices onto Elastic Optical Networks (EON) lightpaths in the context of a multi-layer network employing efficient cross-stratum reoptimization and maintenance [10] [11]. However, such procedures rely on flows with bandwidth reserved in the control plane only. In the data plane, bursty and unpredictable behavior of traffic forwarded by the edge node may be subject to different time-dependant profiles and statistical variations, that may induce bottlenecks, congestion, queue delay, thus affecting QoS (e.g., ultra-low latency requirements). Indeed, current SDN implementations do not support such required stateful-driven forwarding at wire speed directly at the nodes, whereas the controller is typically involved to react upon critical events. However, this may pose serious scalability issues at the controller and may noticeably delay reaction enforcement at the data plane, thus leading to serious forwarding inefficiencies (e.g., unexpected latency and jitter increase). In addition, at the data plane level, dedicated fixed-function hardware is not the best solution to address SDN flexibility and configurability requirements.

To address cyber security, Network/Security Operation Centers (NOC/SOC) defend information systems at edge nodes using dedicated tools and systems like Intrusion Detection Systems, firewalls, Security Information and Event Management tools, enhanced with blockchain-based trustworthiness mechanisms as recently proposed for next generation 5G fronthaul [12]. Usually, detection of attacks or resource bottlenecks is performed without automated tools and is based on systems logs, statistics and experience of operators. In addition, mitigating the attack in decentralized IT (e.g., fog resources) and edge nodes is not trivial and may require the reconfiguration of many network devices at once. Currently, traditional switching solutions are not able to deal with Distributed Denial of Service (DDoS) attacks and implement simple access/black list solutions, such as BGP flow spec traffic redirection [13]. However, they did not prove to be adequate against cyber attacks at network edges due to lack of effective context-based security ana-

Manuscript received September 6, 2018.

F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti and P. Castoldi are with Scuola Superiore Sant'Anna, Pisa, Italy. F. Cugini is with CNIT, Pisa, Italy.

This paper is an extended version of the work presented in [1].

This work has been partially supported by the PROMAS project (Tuscany Region POR FESR 2014-2020) and the EU H2020 METRO-HAUL Project (contract number 761727).

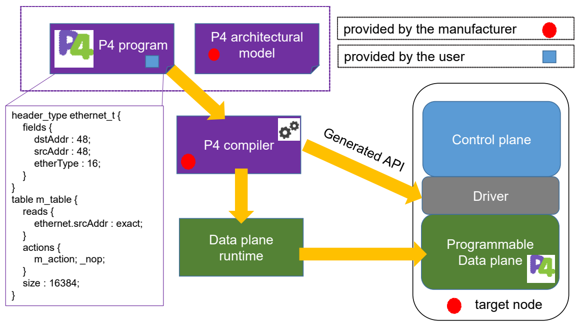
lytics. In the standard SDN context, such attacks may be handled by the centralized controller. However, the default SDN behavior to re-direct unknown packets to the controller may be extremely dangerous for the controller itself, being potentially exposed to attack floodings, out-of-service events and information update inconsistencies, especially in the case of stateful applications [14] [15].

The P4 technology has been recently introduced to enable advanced and configurable packet processing functionalities of network devices, supporting protocol and target platform independence [16] [17]. P4 is a high-level, platform-agnostic language for programming the data plane of SDN network devices. P4 allows to define customized and sophisticated switch pipelines, packet forwarding policies and actions, producing portable implementations over different hardware targets (e.g., network interface cards, FPGAs, software switches and hardware ASICs). Specifically, the availability of stateful programming objects, such as counters, meters and registers enables finite state machines and conditional behavior implementation directly in the hardware. The novelty and the potentials introduced by P4 has gained significant attention by many system vendors and the P4 consortium already involves more than 50 industrial partners. So far, the P4 community has been extremely active in developing and improving the P4 compiler itself and the P4 Behavioral Model software switch (i.e., BMV2). However, limited effort has been reported in the scientific literature to show potentially disruptive innovations and advantages of the P4 technology, especially in the context of multi-layer packet over optical networks.

This paper proposes the adoption of the P4 technology in an SDN multi-layer packet over optical network to enable advanced data plane programmability. In particular, the work proposes an innovative edge node architecture including a P4 switch with native support of deep packet inspection. The P4-enabled node exploits direct stateful processing at wire speed, not demanded - as in OpenFlow systems - to the SDN Controller. Then, it proposes dynamic P4-based TE solutions for multi-layer scenario, such as traffic offloading and dynamic optical bypass. In addition, augmented firewalling capabilities are envisioned proposing a P4 DDoS mitigation proof-of-concept to protect internal edge resources without the need of dedicated firewall hardware. Finally, the evaluation include the P4 code proposals, along with their enforcement in a multi-layer edge node over two different platforms: the reference P4 software switch, namely BMV2 [18], and the Field Programmable Gate Array (FPGA) technology employing 10 GE optical interfaces at full rate [19]. Experimental results report the P4 impact in terms of latency, its scalability in terms of number of sustainable flow entries and its effectiveness showing online TE enforcement and fast detection against cyber-attacks.

With respect to our preliminary study [1], this work introduces the following novel contribution:

1) A presentation of the P4 language, along with related works, and its potentials in the context of multi-layer networks;
2) A detailed architecture proposal of the P4-enabled packet over optical edge node;
3) Extended experimental results including P4 over FPGA framework exploiting 10 Gigabit Ethernet optical interfaces at full line rate.



```
header_type ethernet_t {
   fields {
      dstAddr : 48;
      srcAddr : 48;
      etherType : 16;
   }
}
table m_table {
   reads {
      ethernet.srcAddr : exact;
   }
   actions {
      m_action; _nop;
   }
   size : 16384;
}
```

Fig. 1. Workflow of P4 language compiler and API over programmable devices.

II. The P4 language

The P4 (Programming Protocol-independent Packet Processors) is a high-level programming language explicitly devoted to design the SDN data plane of packet processors [17]. P4 has been conceived in the SDN paradigm, since some P4 proponents are the inventors of the OpenFlow protocol [20]. According to the P4 proponents, such language aims at becoming the abstract programming language of a general purpose networking chip performing dedicated packet forwarding scheme based on the SDN paradigm. Following the concept of network disaggregation, P4 introduces open source programmability of network data plane, enabling own-made development of new proprietary protocols or headers, advanced forwarding and congestion control strategies, ad-hoc monitoring and telemetry functions without the need of costly dedicated proprietary devices or time consuming hardware firmware upgrades.

Compared to the state-of-the-art of packet processing systems, based on micro-code, P4 provides the configurable building blocks of an abstract network node, ranging from Layer0 up to Layer7 functions: parsers (including non-standard headers), metadata (i.e., data that can be internally associated to a packet for processing, for example its input port), conditional controls, tables, along with a primitive set of actions. In addition, following the abstract forwarding model imposed by the language, packet-forwarding policies, algorithms and per-packet custom actions can be implemented producing portable implementations over different hardware targets (e.g., network interface cards, FPGAs, software switches, bare metal switches and hardware ASICs).

The P4 language operates within a high-level view of the general macro-blocks of the switch, called *abstract forwarding model*. The Protocol Independent Switch Architecture (PISA) is the current model and represents one of the most significant results of P4 research team driving the development of the P4 language itself. A comparison evaluation between the P4 implementation and the state-of-the-art of fixed-function switch hardware has shown that packet processing speeds are the same with almost no additional cost or power [16].

The P4 abstract forwarding model is composed by the following blocks:

1) a programmable Parser block, responsible of identifying the stack of allowed protocols and fields defined by the program;
2) a programmable Ingress pipeline, made of a set of match+actions tables, responsible of conditional packet

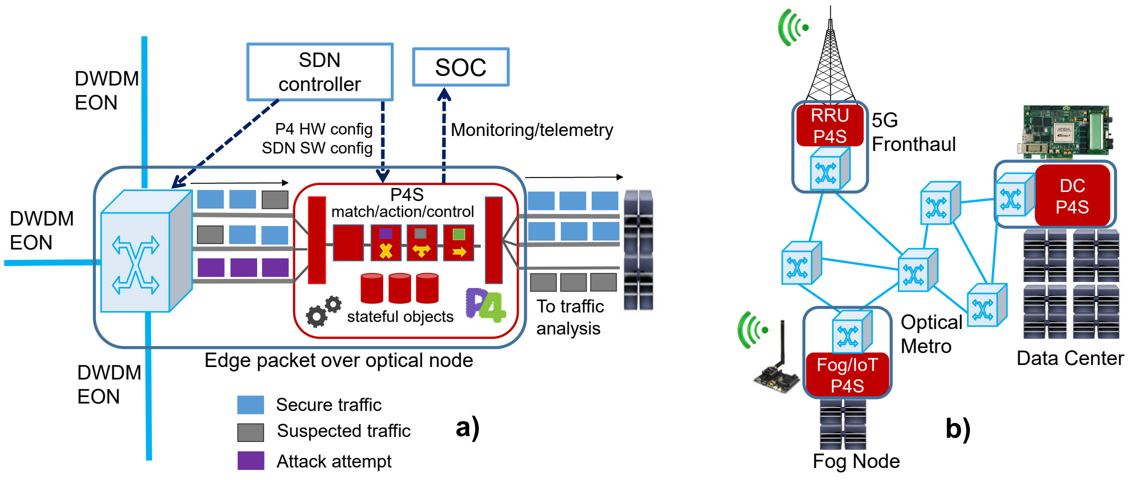


Fig. 2. Packet-over-optical P4-based edge node: internal functional architecture and SDN control/monitoring (a); deployment and applicability scenario of P4 nodes acting as advanced forwarding devices and online distributed security barrier at the edge of the optical metro network (b).

processing and field update, egress port and queue selection;

3) a programmable Egress pipeline, used for per-instance header modifications after egress port selection.

Figure 1 shows a simple example excerpt of a P4 program defining the Ethernet header and a table matching the Ethernet source address. Moreover, it shows the workflow of P4 programs compilation and hardware enforcement. The P4 program, written by the user utilizing the architectural model of the physical target, is compiled providing:

1) a front-end representation (typically a JSON file) used to drive a back-end target-specific compiler for runtime data plane enforcement;
2) an auto-generated runtime API to control the driver between control and data plane (i.e., to directly populate tables with flow entries following the P4 namespaces).

The P4 language defines a set of stateful objects that can be used to implement finite state machines and complex state-based decisions. Stateful elements store variables beyond the processing lifetime of a single packet, that may be read or updated depending on specified control conditions. In particular, two stateful constructs are available: *tables* and *extern objects*. Tables are read-only for the data plane, but their entries can be modified by the control plane. Extern objects have state that can be read and written by both the control and the data plane. In particular, among extern objects, registers (storing values), counters (storing incremental occurrences) and meters (storing rate values) may be instantiated. These stateful elements and their size are explicitly declared in the P4 code and allocated during the instantiation phase. This way, P4 can be used to dedicate pre-planned and dynamic countable hardware resources to specific functions and processes. Moreover, P4 enables to instruct stateful switches data plane with advanced functionalities with respect to standard OpenFlow switches, for instance implementing user-defined protocols or finite state machines.

Finally, a P4 program may be designed in a modular fashion with a baseline code structure and a set of extendible code pieces. As an example, P4 codes implementing basic router and switch functionalities have been proposed allowing incremental functions and support of protocols. In general, parsers, actions and even tables may be re-used in a P4 code (e.g., merging two P4 programs implementing different TE techniques), thus enabling a number of alternative or parallel functionalities within the same switch. However, stateful objects require per-service instantiation, otherwise the monitored state may become inconsistent.

A. Related Work on P4

A number of recent work rely on the P4 language to deploy advanced services such as telemetry, protocol implementation and stateless firewalling.

In-band telemetry applications of P4 resort to the collection of online network state parameters to be processed by the management plane [21]. The idea behind in-band telemetry is to collect flow-based direct measurements in the data plane (e.g., latency, queue transit time) using packet manipulation of non-standard headers (e.g., storing a timestamp value) that an external knowledge plan is able to process, also resorting to Artificial Intelligence and Machine Learning techniques. This allows to derive finer statistics for feedback-based automatic SDN intervention procedures.

An example of protocol implementation at run-time using P4 has been presented by the work in [22]. The Bit Index Explicit Replication (BIER) is a novel SDN-oriented protocol proposed for multicast routing that requires a dedicated bit-indexed header encoding the multicast tree links selection [23]. P4 allows a switch pipeline description implementing the header building and its encoding/decoding procedure.

Applications of P4 have been presented in the context of SDN security, however mainly limited to stateless firewall configurations including port/protocol filtering, blacklist and rate limiter [24]. Stateless P4-based header and packet header manipulation achieving mitigation techniques such as anti-spoofing mechanisms have been explored and analyzed [25]. All such approches deal with active stateless processing of the header, without introducing simple finite state machines and history-based processing.

The most important work on FPGA and P4 is presented in [19], in which authors developed an open source compiler and runtime for P4 over FPGA and evaluated in deep detail the performances of some P4 programs in terms of hardware resource occupancy and latency compared with fixed function ASIC. Proposed P4 programs rely on L2/L3 forwarding and complex protocol implementations in the context of market data and distributed computation agreement.

Finally, stateful SDN data plane programmability by means of alternative strategies and tools besides P4 have been proposed and discussed concerning efficiency and security [26], among which one of the most interesting and significant proposes the extension of the OpenFlow protocol supporting finite state machine abstraction compatible with table-oriented API [27] [28].

This work, differently from previous studies, focuses on the application of P4 stateful capabilities exploiting advanced TE and cyber security in multi-layer edge nodes.

III. P4 IN MULTI-LAYER EDGE NODES

This section introduces the application of the P4 language to a multi-layer packet-switched (e.g., IP/MPLS or Carrier Ethernet) over optical-switched (e.g., DWDM or EON based) edge node with advanced and programmable SDN forwarding plane. While requested data is becoming closer to the user (i.e. fog nodes), attached edge nodes connected to the metro or metro/core network need a more refined treatment of selected class of traffic requiring QoS and TE (e.g., strict latency requirements), subject to profile statistical modifications or high burstiness behavior. The architecture of the edge packet-over-optical node encompassing P4 programmable data plane is depicted in Fig. 2-a. The optical part comprises a SDN-controlled ROADM (e.g., disaggregated whitebox) [29] [30] with its tributary cards attached to a P4 switch (P4S), representing the key packet-switching element of the edge node. A number of P4S optical interfaces are connected to the ROADM cards, while the remaining interfaces connect local or internal resources. The P4S is hardware-programmed by the P4 language and is handled by a SDN controller/orchestrator, responsible for table entries population and service deployment. Multiple functions may be programmed at the same device, including TE/QoS features (e.g., latency-aware forwarding, dynamic offloading or bypass) and security applications (e.g., block, mitigation, telemetry and anomalies reports to SOC, suspected traffic deviation). Reporting data, statistics, alarms, telemetry functions to a Monitoring Handler/SOC are also programmable inside the P4S, enable possible integrated multi-layer proactive monitoring infrastructure. For example, P4 in-band telemetry [21] combined with optical layer advanced monitoring realized in the context of disaggregated networks [31] may be integrated in a joint multi-layer telemetry system.

Different potential P4-based edge node deployment scenarios are envisioned, as illustrated in Fig. 2-b. In particular, such extended node may be placed as Data Center (DC) gateway, edge node device (e.g., fog node or IoT gateway), 5G fronthaul node (e.g., extended Remote Radio Unit - RRU supporting Fiber to the Antenna technology). For example, in the 5G Fronthaul scenario, specific P4 switches may be employed to perform online traffic telemetry and implement precise latency-assured traffic forwarding (e.g., dynamically manipulating the IP/MPLS QoS service flag thus driving stateful scheduling with priority). In addition, such node may be employed even at intermediate network nodes (e.g, aggregation metro node). In this case, traffic engineering solutions may be enabled not only at the edge but also in the metro network. In the case of cyber security solutions, this choice may avoid that security threats reach the edge and frees resources to better focus on more sophisticated attacks. The P4 solution allows a unique SDN edge switch deployment with advanced TE and security functions, detailed in the next sections, avoiding processing burden at the controller and additional specialized TE and firewall hardware inside cloud/fog node.

In multi-layer optical networks, TE techniques such as optical bypass are possible through instantiation of optical paths and steering of traffic based on policies that can be configured by a SDN controller using specific flow entries. However, such TE techniques and traffic steering configurations are typically enforced with static or stateless match conditions. When traffic conditions change such policies are typically modified by the controller. However, this requires the deployment of complex monitoring and telemetry techniques at the controller, possibly incurring in severe scalability issues in the case of high volume traffic, e.g., as in the case of DC gateway. The availability of a programmable data plane enables the deployment of advanced traffic engineering solutions (e.g., stateful TE) at SDN devices without requiring the intervention of a controller. In addition, the presence of a stateful SDN device capable of providing complex monitoring/telemetry information and alerts may drastically help the SDN controller for both scalability (i.e., reduced number of monitoring polling messages) and operation performance (i.e., accurate and fast detection of anomalies) and new type of networking statistics such as min/average/max latency spent in queue). Telemetry data may be elaborated by an external telemetry collector interacting with the SDN controller, allowing the latter to react immediately without being overwhelmed by excessive monitoring messages.

Several use cases may strongly motivate the adoption of a stateful P4 switch in the architecture of an edge node:

- Advanced Traffic Engineering (e.g., dynamic traffic-based routing, massive load balancing in segment routing applications);
- Cyber Security mitigation and intrusion detection;
- QoS precise forwarding (e.g., ultra-low latency for 5G, lossless Ethernet, automatic packet reordering);
- Advanced monitoring solutions (e.g., active probe generation for fast failure detection or forecasting, in-band telemetry);
- Packet header customization and manipulation for online service differentiation (e.g., mice and elephant flows header differentiation in DC scenarios).

In the next sections, advanced traffic engineering and cyber security mitigation use cases in the context of a multi-layer edge node will be presented targeting the P4 technology as candidate solution.

IV. STATEFUL TRAFFIC ENGINEERING WITH P4

Fig. 3 shows an inter-data center connectivity use case, where DC1 and DC2 are connected by a packet-switched path and by an additional packet-over-optical path exploiting optical bypass. Traffic originated by DC1 and destined to DC2 follows the flow rules installed in the S1 switch. Stateless flow rules may be enforced to a standard SDN switch (e.g., OpenFlow switch), based on specific packet attributes.

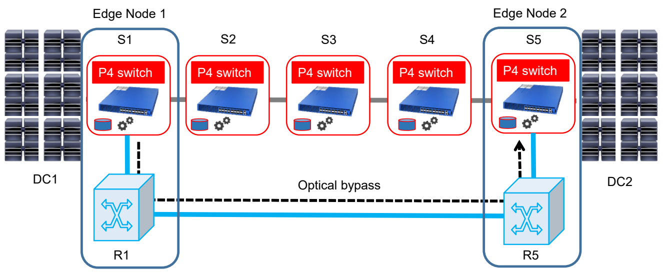


Fig. 3. Advanced traffic engineering use case: data center gateways equipped with P4-based edge node performing dynamic TE.

As an example, latency-sensitive traffic (e.g., matched on protocol/application type) may be steered to the optical bypass. However, forwarding dictated by dynamic traffic conditions and profiles (i.e., stateful TE) is not feasible using a standard OpenFlow switch, without the active involvement of the controller. Programmable data plane enables stateful TE. Two examples are provided in the following subsections: dynamic traffic offloading and dynamic optical bypass based on stateful traffic conditions.

A. *TE: Traffic offloading*

In the first TE use case, traffic offloading is implemented. An incoming traffic rate threshold *TH* is considered. The objective is to dynamically reroute just the portion of traffic beyond the rate threshold along an alternative path, i.e. implementing traffic shaping and limiter and avoiding congestion. To enforce such use case, a P4 program is exploited to build the forwarding plane of the switch. The program first defines the required packet headers (i.e., Ethernet and IP headers), then relies on a pipeline of two Ingress flow tables and on a stateful structure provided by P4, the *Meter*. As defined in [32], Meters are three state markers (i.e., output states are red, yellow and green) based on the definition of two rates, the Committed Information Rate (CIR) and the Peak Information Rate (PIR). Marker is set to red if rate exceeds the PIR, to green if rate is below the CIR, to yellow if rate is between CIR and PIR.

The traffic offloading P4 code key sections are reported in Fig. 4. First, packets are parsed to check the protocol stack. In this case Ethernet frames are checked and Ethernet fields saved by the parser (not shown in the figure). Then packets enter the ingress section, designed by the *control* tag, where the tables pipeline is defined. Two tables are defined: the *m_table* and the *m_filter*. In the *m_table*, packets are inspected by their source MAC address (i.e., *ethernet.srcAddr*) and, in case of match, the *m_action* is executed. In the P4 program, an array of meters is declared and instantiated (*my_meter*). The *m_action* triggers the related meter identified by the *meter_idx* index and saves the result in a packet metadata field. Metadata fields are defined to carry out specific information related to the packet. Standard metadata are already defined in P4, such as the packet output interface (i.e., *standard_metadata.egress_spec*). The program defines an additional metadata providing the meter execution result (i.e., *meter_tag*). Then, the packet is passed to a second flow table (*m_filter*) which applies a token-bucket behavior according to its current Meter value, selecting between either the default or the alternative output port (i.e., through the *steer_port* action). In both cases, forwarding

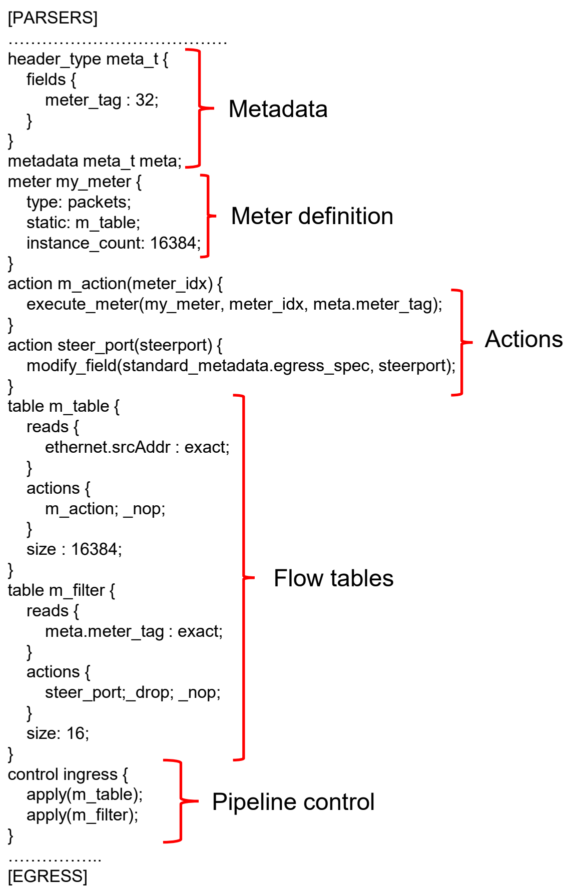

```
[PARSERS]
.......................................
header_type meta_t {
    fields {
        meter_tag : 32;
    }
}
metadata meta_t meta;
meter my_meter {
    type: packets;
    static: m_table;
    instance_count: 16384;
}
action m_action(meter_idx) {
    execute_meter(my_meter, meter_idx, meta.meter_tag);
}
action steer_port(steerport) {
    modify_field(standard_metadata.egress_spec, steerport);
}
table m_table {
    reads {
        ethernet.srcAddr : exact;
    }
    actions {
        m_action; _nop;
    }
    size : 16384;
}
table m_filter {
    reads {
        meta.meter_tag : exact;
    }
    actions {
        steer_port;_drop; _nop;
    }
    size: 16;
}
control ingress {
    apply(m_table);
    apply(m_filter);
}
.................
[EGRESS]
```

Fig. 4. TE: dynamic traffic offloading P4 code based on meters and token-bucket.

rules are dynamically applied according to actual traffic conditions, with no Controller intervention. Indeed, P4 dictates the general switch behavior abstracting from the entries of the flow tables. This means that the behavior of the switch can be applied to different matching conditions just updating the flow entries of the P4 switch, without the need of reprogramming it. For example, the *steer_port* action identifies the output port of the packet using the *steerport* parameter. Thus, in the case of topology changes (e.g., an additional ROADM is added with the possibility of implementing two rates meter), the update of the output port for yellow and red traffic may be simply re-adapted by modifying at runtime the flow entry of table *m_filter*. In addition, the meter *TH* threshold is configured as configuration entry of the meter at runtime, therefore it can be tuned by network administrator or by the controller during the switch functioning. It is worth to note that the whole P4 program, including parsers, meters, actions, tables and pipeline control sections, is less than 100 lines of P4 code.

B. *TE: Optical bypass*

In the second TE use case, the edge node implements dynamic optical bypass. An incoming traffic rate threshold *TH* is considered. When DC1-DC2 traffic rate remains below threshold *TH*, traffic is forwarded along the P4 switches chain in the packet-switched layer. Conversely, when traffic exceeds the threshold, all the matching packets are automatically steered to the optical node R1 which injects traffic along the pre-established optical path between R1 and R2. To implement such use case, a P4 program builds the internal structure and describes the forwarding workflow of the edge node.

The key sections of the P4 program are illustrated in Fig. 5. The program defines a metadata *flowlet_meta_t* with two fields to store the timestamps of the input interface of the switch related to the packet itself and the previous one. In addition, an array of registers *flowlet_reg* are defined and instantiated. The control relies on a pipeline of two flow tables of type Ingress (*m_flowlet* and *m_bypass*), however, in this case, the pipeline is not static but subject to conditions. The first table is always executed to match the considered traffic flow according to configured parsing conditions (e.g., source/dest MAC/IP etc, in this case source MAC address). In addition, when matching conditions apply, the *flowlet_action* action is performed. Four commands are executed in the action: the timestamp of the previous matched packet is read from the register and saved in the packet metadata (*previous_ts* field); the current packet timestamp is saved in the metadata (*current_ts* field); the current timestamp is written and stored in the register; the default output interface is selected (e.g., port towards S2). Therefore, the P4 stateful register is used to store in a vector previously collected timestamps. This way, frame rate and traffic profile (e.g., flowlet) can be assessed driving to specific forwarding conditions. This is obtained by defining a pipeline control behavior subject to internal matching conditions. In particular, a P4 Control condition is set to either exploit the second ingress flow table *m_bypass* in the case of flowlet match (i.e., the interarrival time between matching packets is below *TH*, the constant *FLOWLET_INTERVAL* defined in the code), which applies the *steer_port* action, changing the output port value. This activates optical bypass forwarding of matched frames towards R1, or maintaining the default output port towards S2. Also in this case the whole P4 program is less than 100 lines of code.

Note that the optical bypass outgoing port is selected by the parameter of the *steer_port* action. Such value is stored as flow entry of the *m_bypass* table and can be modified at any time (e.g., by the P4 switch command line interface or by the SDN controller). This means that optical bypass selection itself may be adapted to traffic conditions or network status. For example, the SDN controller may decide to steer traffic to another available optical bypass, thus just requiring a single entry update at the P4 switch. Moreover, thanks to P4 stateful objects and internal telemetry, based on steered traffic statistics, the P4 switch might trigger, through the SDN controller, a lightpath adaptation request to the optical control plane (e.g., elastic operations to an active stateful Path Computation Element in the case of additional bandwidth requirement [33] or predictive analytics on traffic flows [34], physical parameter adaptation in the case of poor QoS [35]).

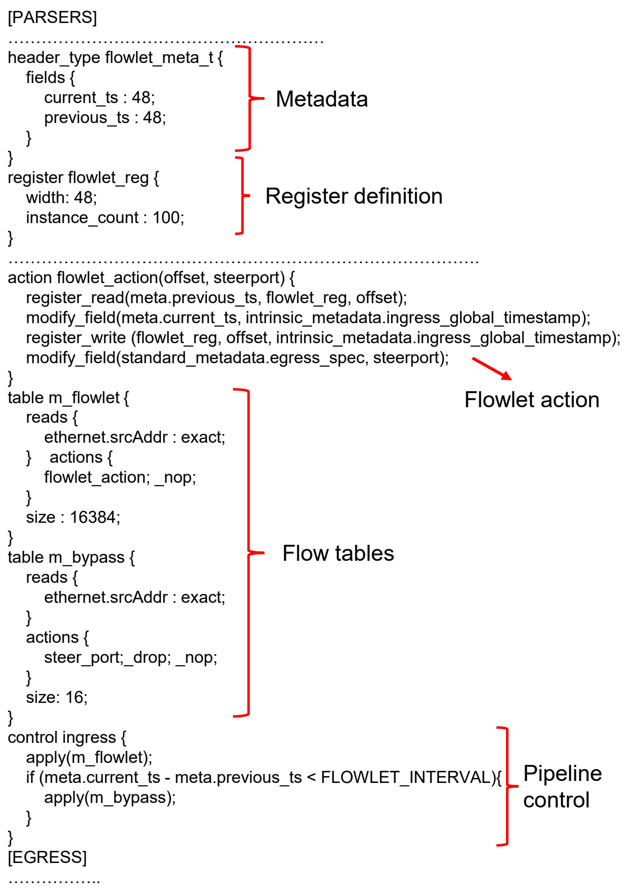

```
[PARSERS]
.........................................................
header_type flowlet_meta_t {
   fields {
      current_ts : 48;
      previous_ts : 48;
   }
}
register flowlet_reg {
   width: 48;
   instance_count : 100;
}
..............................................................................
action flowlet_action(offset, steerport) {
   register_read(meta.previous_ts, flowlet_reg, offset);
   modify_field(meta.current_ts, intrinsic_metadata.ingress_global_timestamp);
   register_write (flowlet_reg, offset, intrinsic_metadata.ingress_global_timestamp);
   modify_field(standard_metadata.egress_spec, steerport);
}
table m_flowlet {
   reads {
      ethernet.srcAddr : exact;
   }   actions {
      flowlet_action; _nop;
   }
   size : 16384;
}
table m_bypass {
   reads {
      ethernet.srcAddr : exact;
   }
   actions {
      steer_port;_drop; _nop;
   }
   size: 16;
}
control ingress {
   apply(m_flowlet);
   if (meta.current_ts - meta.previous_ts < FLOWLET_INTERVAL){
      apply(m_bypass);
   }
}
[EGRESS]
................
```

Fig. 5. TE: dynamic optical bypass P4 code based on registers and flowlet switching.

V. Cyber security mitigation with P4

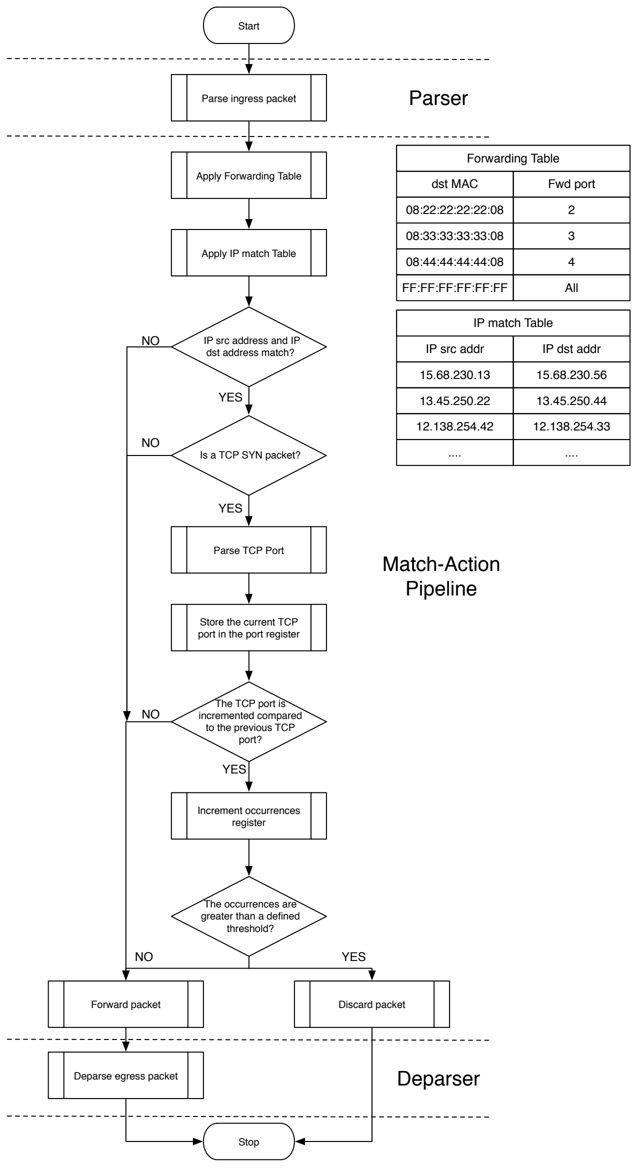
The same network scenario shown in Fig. 3 and the node architecture of Fig. 2-a are also exploited to show how the P4 technology can be efficiently used to react against cyber-attacks. As an example of cyber threat, a distributed denial-of-service (DDoS) attack exploiting address/port scan is considered. All possible TCP/UDP ports of a target IP destinations are attacked from multiple infected IP source nodes. Such type of attack can not be simply blocked by access/black lists, since this could affect also legitimate remote connections. That is, such type of attacks can not be blocked by using traditional OpenFlow switches where just basic stateless permit/deny flow matches are practically available.

Conversely, the stateful nature of P4 provides innovative solutions to address such critical threats directly within the network nodes, by detecting attacks by means of deep packet inspection and packet sequence correlation at runtime. The designed P4 program, besides including TE solutions, also relies on P4 Registers to store header information (e.g., IP dest and TCP/UDP port along with the related timestamp) for a number N of previously received packets. Then, P4 Control conditions can be configured to analyze the retrieved data and identify possible ongoing port scan attacks. This way, packets normally directed to the default output port can be temporarily blocked, successfully dropping such attempts for a configurable amount of time, or re-directing suspected traffic to dedicated stateful firewalls thus implementing attack mitigation. Such functions are implemented directly at the switch, as before, without involving the SDN controller with excessive amount of packets, which typically happens in DDoS attacks impacting controller's stability and functioning.

A. P4 application: DDoS

The proposed P4 switch edge node is implemented as proof of concept against the TCP SYN flood attacks [36], enforced on both software switches and programmable hardware functionalities of FPGA. Fig. 6 depicts the related P4 workflow. The parser section of the program defines the rules to parse incoming packet. All ingress frames received at a given interface, coming from external hosts are first parsed to detect the protocol stack. In particular, Ethernet framing, (optional) IP parsing and (optional) TCP parsing are performed in cascade. After this step, forwarding is applied by checking a Forwarding Table (defined in the P4 program) and the physical egress port is assigned, based on the destination address of the current packet. Based on these fields, the switch detects whether a packet comes from a suspicious host or it belongs to a suspected traffic profile (IP match table may be populated by a centralized security controller). Then, the program enters the control section and checks if the packet is a valid TCP SYN and, if not, forwards it. Otherwise, it parses the TCP port to detect anomalous behaviour. To check anomaly, stateful evaluation of the session is performed. In particular, if TCP destination port is incremented compared to the previous TCP port of the same session (i.e., the basic TCP SYN flood mechanism) a stateful object is updated accordingly to store and keep updated the session state.

In this case, two register variables are allocated per IP Match Table entry. The first variable stores the TCP port received by the last packet belonging to the session, whereas the second variable stores the number of attempts matching the TCP SYN Flood basic behavior. The two registers are continuously updated upon each new packet processing. If the number of detected attempts (i.e., the second register) becomes greater than a pre-defined threshold (e.g., 10 attempts), the packet is discarded. Otherwise, registers are updated and the packet is forwarded following the standard P4 egress section.


Start
Parse ingress packet
Parser
Apply Forwarding Table
Apply IP match Table
NO
IP src address and IP dst address match?
YES
NO
Is a TCP SYN packet?
YES
Parse TCP Port
Match-Action Pipeline
Store the current TCP port in the port register
NO
The TCP port is incremented compared to the previous TCP port?
YES
Increment occurrences register
The occurrences are greater than a defined threshold?
NO
YES
Forward packet
Discard packet
Deparse egress packet
Deparser
Stop

Forwarding Table	
dst MAC	Fwd port
08:22:22:22:22:08	2
08:33:33:33:33:08	3
08:44:44:44:44:08	4
FF:FF:FF:FF:FF:FF	All

IP match Table	
IP src addr	IP dst addr
15.68.230.13	15.68.230.56
13.45.250.22	13.45.250.44
12.138.254.42	12.138.254.33
....	....

Fig. 6. Cyber security: P4 program workflow targeting mitigation on TCP SYN flood attacks.

VI. Experimental Evaluation

The proposed P4 programs enforcing TE and cyber security have been implemented and experimentally evaluated in a multi-layer network testbed reproducing the network depicted in Fig. 3. The P4 switch has been evaluated in two different hardware versions, showing the P4 capability to be target-independent:

1) the reference software switch used in the P4 framework, namely the Behavioral Model version2 (BMV2) implemented in a Linux PC;
2) the NetFPGA-Sume board, a FPGA dedicated to networking applications.

A. Behavioral Model Version 2 soft switch

In this experimental evaluation section, the edge multilayer node includes a P4 switch realized with a BMV2 software switch connected to a Reconfigurable Add-Drop Multiplexer (ROADM). In particular the optical bypass is implemented through 100G commercial muxponder, handled by a local agent connected to a optical-layer SDN controller running NETCONF as southbound API [30]. Five servers (CPU 3.40GHz, 4 GB RAM, Ubuntu 14.04 kernel 4.4.0-31-generic), equipped with multiple 1 and 10 Gigabit Ethernet interfaces, are operated with the BMV2 software switch and configured with the three P4 version 14 programs presented in the previous sections. Traffic is generated and received by two Linux PC servers running Python-based traffic generators and receivers based on the *scapy* library.

Fig. 7 shows the BMV2 P4 S1 switch behavior (see Fig. 3) when the P4 program of Sec. IV-A is applied for traffic offload. Traffic is received by the P4 switch interface connected to the generator. In this case the *m_table* flow-entry matching the traffic subject to meter measurement is the source MAC address of the packet generator, thus the meter is applied to all the generated traffic. Note that with the same program it would be possible to meter and tune forwarding for specific traffic flows. When aggregated traffic rate exceeds *TH* (set to a value of 300 packets/s), the second *m_filter* flow table applies the configured P4 rule, correctly identifying for forwarding towards the alternative port only the portion of traffic exceeding *TH*. In particular, the flow entries configured steering traffic to the *IP iface*

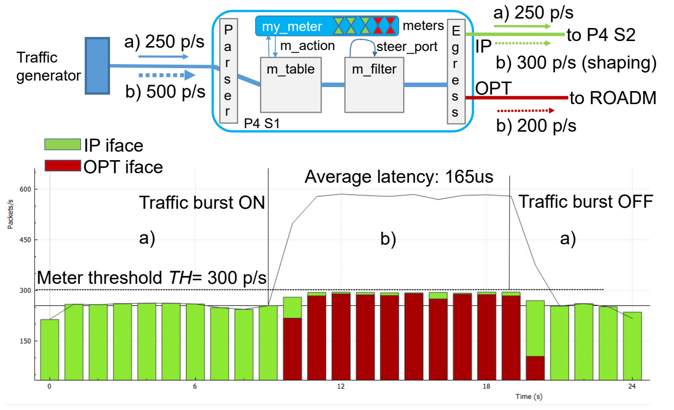


Fig. 7. BMV2 results: TE traffic offload behavior (packets/s versus experiment time [s]).

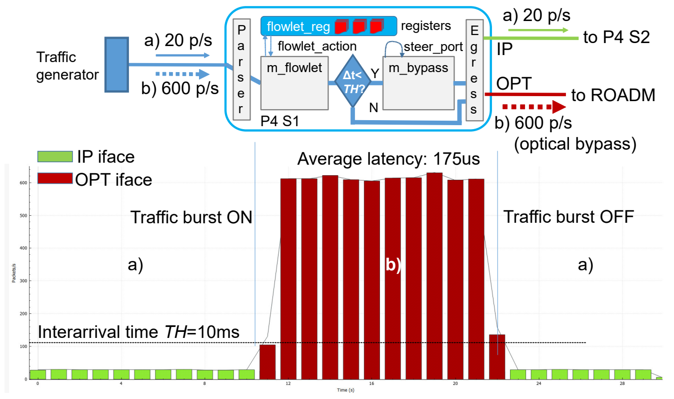
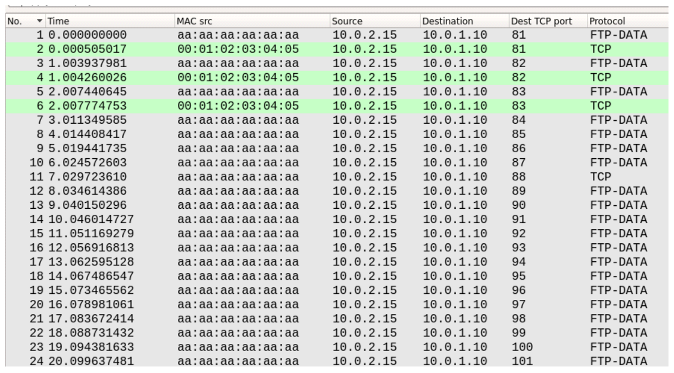


Fig. 8. BMV2 results: TE optical bypass behavior (packets/s versus experiment time [s]).



No.	Time	MAC src	Source	Destination	Dest TCP port	Protocol
1	0.000000000	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	81	FTP-DATA
2	0.000505017	00:01:02:03:04:05	10.0.2.15	10.0.1.10	81	TCP
3	1.003937981	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	82	FTP-DATA
4	1.004260026	00:01:02:03:04:05	10.0.2.15	10.0.1.10	82	TCP
5	2.007440645	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	83	FTP-DATA
6	2.007774753	00:01:02:03:04:05	10.0.2.15	10.0.1.10	83	TCP
7	3.011349585	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	84	FTP-DATA
8	4.014408417	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	85	FTP-DATA
9	5.019441735	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	86	FTP-DATA
10	6.024572603	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	87	FTP-DATA
11	7.029723610	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	88	TCP
12	8.034614386	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	89	FTP-DATA
13	9.040150296	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	90	FTP-DATA
14	10.046014727	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	91	FTP-DATA
15	11.051169279	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	92	FTP-DATA
16	12.056916813	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	93	FTP-DATA
17	13.062595128	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	94	FTP-DATA
18	14.067486547	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	95	FTP-DATA
19	15.073465562	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	96	FTP-DATA
20	16.078981061	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	97	FTP-DATA
21	17.083672414	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	98	FTP-DATA
22	18.088731432	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	99	FTP-DATA
23	19.094381633	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	100	FTP-DATA
24	20.099637481	aa:aa:aa:aa:aa:aa	10.0.2.15	10.0.1.10	101	FTP-DATA

Fig. 9. BMV2 results: Wireshark capture of TCP SYN Flood port scan received and blocked after three attempts by the cyber security P4 program.

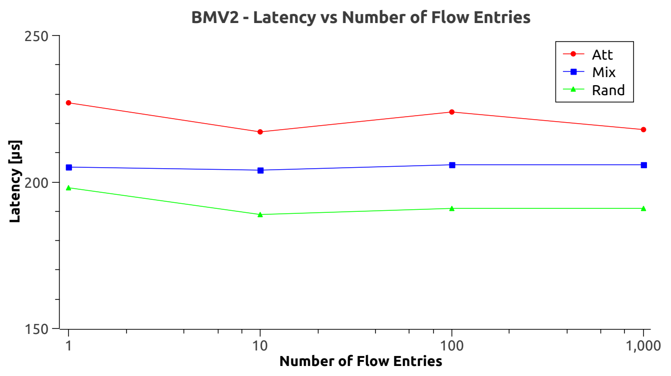


Fig. 10. BMV2 results: scalability performance of the cyber security P4 program in different attack rate scenarios.

port (i.e., the port connected to switch S2, in the packet-switched layer) if meter result is set to 0 (i.e., traffic is lower the threshold, see Fig. 7 case *a*), or steering traffic to the *OPT iface* optical port (i.e., the port connected to the muxponder) if meter is not 0 (see Fig. 7 case *b*, during the traffic burst injection). Results show that upon a traffic burst of 600 packets/s, the switch applies a shaper on the IP port limiting the output to the threshold, while exceeding traffic is automatically re-directed to the optical domain. Such solution allows to keep controlled the profile and the burstiness of the packet switched layer, avoiding possible congestion. With this program, the measured latency of the P4 BMV2 switch is 165us. The meter rate TH does not influence latency results, since the number of operations inside the switch and the amount of instantiated resources is the same for any constant value.

Fig. 8 shows the same P4 switch S1 behavior when the P4 program of Sec. IV-B is applied for optical bypass. First, a traffic flow at rate below *TH* is considered (see Fig. 8 case *a*). The first flow table used for matching purposes (i.e., table *m_flowlet*) identifies the metadata timestamp to be stored in the P4 stateful register. The P4 Control condition is not met and traffic is forwarded along the default output port (to S2) towards B. When incoming traffic increases with a traffic burst (see Fig. 8 case *b*), exceeding *TH*, (i.e., the inter-arrival time decreases below the `FLOWLET_INTERVAL` constant, set to 10ms) the control condition imposes an additional flow table transit (i.e., *m_bypass*), successfully enforcing for packet forwarding the optical bypass, i.e. selecting the optical output port. This means the whole matching traffic is re-directed to the optical pipe. As shown in the figure, when traffic burst terminates and rate decreases below *TH*, matching traffic is rerouted again to S2 at the packet layer. Note that, hysteresis-based conditions relying on two thresholds can also be easily implemented to improve network stability. In this case the measured average switching latency is 175us, again with no dependance on the selected TH values.

Cyber security P4 program of Sec. V-A has been implemented over the BMV2 (at switch S5) and evaluated. Fig. 9 shows the capture collected at switch S5 related to a simple cyber security attack use case. A DDoS block of port scan with incremental Dest TCP Port is implemented (1 packet/s rate). In particular (see P4 program workflow in Fig. 6), for each matching flow, a *port* register stores the TCP port of the last matched frame, while an additional *occurrences* register stores the number of consecutive scan condition occurrences. If scan is detected, a threshold of N=3 packets is allowed to be forwarded by the switch while the subsequent ones (Port >83) are dropped, successfully blocking the considered cyber-attack.

The cyber security program has been evaluated with more complex attack scenarios. Fig. 10 shows the scalability analysis as a function of the P4 program size in terms of configured matching and forwarding conditions. Up to 1000 flow entries of table *IP_Match* (matching IP source and destination) have been configured on switch S5. Then, three types of traffic flows are considered. In the *rand* case, the switch is loaded

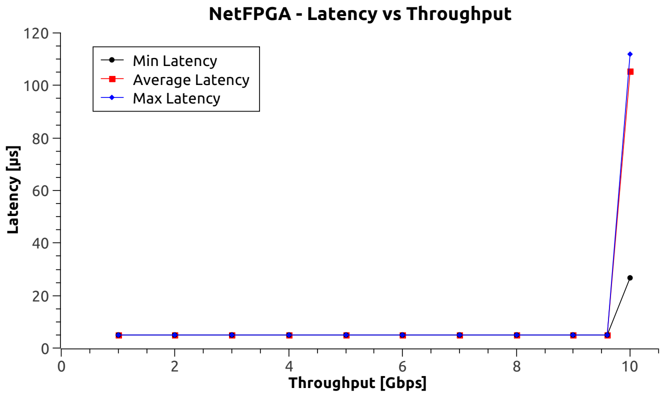


Fig. 11. NetFPGA results: latency as a function of the traffic throughput.

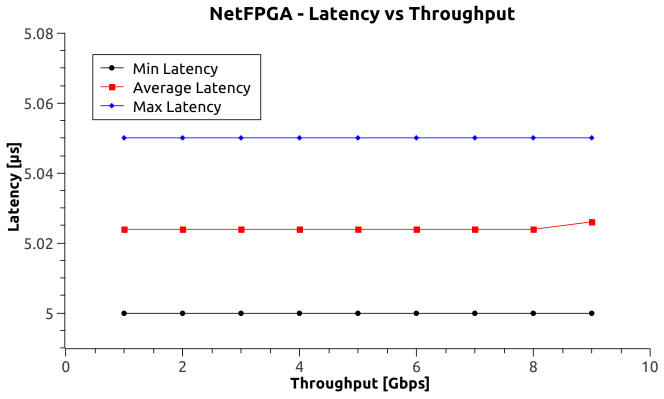


Fig. 12. NetFPGA results: zoomed version of Fig. 11 in the 1-9Gbps range.

TABLE II
P4-NetFPGA hardware resource utilization

NetFPGA Resource	Reserved by P4 program (%)
LUTs used	23.18
Flip-Flop used	16.76
MUX used	0.43
DSP used	0
RAM blocks used	37.04

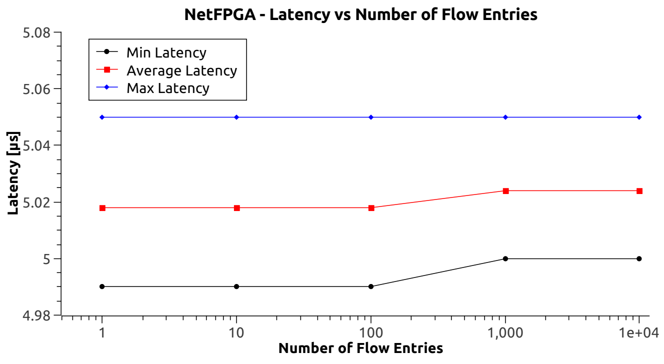


Fig. 13. NetFPGA results: latency as a function of installed flow entries.

with random traffic with no attacks; the *att* case includes only packets referring to the attack; the *mix* case includes a 50:50 combination of both. Results show that very constant latency performance of around 200us is achieved at the increase of P4 flow table entries, while latency variations (up to 40us) are experienced as a function of the actual internal P4 operations according to traffic conditions. In particular, the attack case requires a longer workflow execution with respect to non-attack scenario, impacting the BMV2 total processing time of a packet in the two different scenarios.

B. P4-based NetFPGA

To evaluate the impact of P4 over real programmable hardware devices, the P4 switch implementing cyber security program has been also implemented on a NetFPGA-Sume board [19] [37]. The board is based on the Xilinx Virtex 7 FPGA capable to support 4x10 Optical 10 Gigabit Ethernet interfaces SFP+ ports. The board is equipped with 8 GB of DDR3-SODIMM RAM and a x8 Gen3 PCIe that allows to control the NetFPGA from an external host. In this case, the board can be plugged as a standard Network Interface Card (NIC) to the PC, with the possibility of reconfiguring the hardware. The NetFPGA hardware is re-configured by means of the Vivado, SDK Toolkit and Xilinx SDNet software toolkits. P4-based hardware enforcement resorts to proprietary drivers interpreting the JSON files produced by the P4 compiler (p4c version 16). Two NetFPGA SFP+ 10G optical interfaces are connected by means of optical fibers to the Spirent SPT N4U traffic generator and analyzer. The generator is equipped with the MX-10G-S8 card providing up to 8 SFP+-based 10G optical Ethernet interfaces, with traffic profiles obtained by setting different values of the total transmitted throughput. Besides generic TCP traffic profile, specific attack profiles were created emulating TCP SYN Flood attack sequences with configurable percentage of the total throughput.

Fig. 11 reports the P4-NetFPGA latency in the worst case scenario (i.e., 10 K entries in the IP Match table and 100% attack scenario, in this specific case the discard action has been disabled to measure the latency) as a function of the optical 10 Gigabit Ethernet traffic throughput. Results show an average latency of 5us, practically constant, with constant and very low variance (i.e., min and max latencies differ of around, 50ns) and independent on the traffic throughput in the range 0-9.6Gbps. Latency increases up to 110us in the range 9.6-10Gbps upon quasi full-rate condition when NICs typically introduce significant packet loss. Similar latency (i.e., 4.8us) has been measured on the commercial HP3800 SDN switch at 10Gpbs (exploiting dedicated ASIC), where traffic was simply filtered in a static and stateless configuration, using a standard OpenFlow 1.3 rule in the hardware table matching the TCP port. Thus, this comparison proves that the P4 processing is enforced without introducing significant latency degradation with respect to dedicated ASIC.

Fig. 13 shows the latency as a function of the monitored IP sessions in the worst case (9Gbps, 100% attack). Excellent scalability performance is provided, achieving a quasi-constant profile of average (5.01us at 10k entries), min and max values. This result, from the point of view of the SDN framework, is particularly noticeable. The reason behind the quasi-constant latency resides in the usage of fully-associative Content Addressable Memory (CAM) within the NetFPGA [38]. In fact, tables with flow entries are instantiated as CAM tables. Unlike Random Access Memory (RAM) that receives an input address and returns data word associated to that address, a CAM memory receives a data

TABLE I
P4-NETFPGA LATENCIES IN ATTACK AND NON-ATTACK SCENARIOS

TCP SYN attack rate (f/s)	TCP non-attack rate (f/s)	Throughput (Gbps)	Min latency (us)	Average latency (us)	Max latency (us)
100	751202	9.134	5.01	5.023	5.05
752188	100	9.146	5	5.024	5.05

word in input and searches the entire memory in a parallel fashion to detect at which memory offset the input data word is stored. The NetFPGA implements CAM memory by means of its on-chip Block RAM (BRAM), enabling massive paralleling search [19]. This means that, given the maximum size of flow entries that may be stored in a table, the time needed to perform a memory look up is kept constant and therefore, the performance of a P4 SDN device over the NetFPGA is practically independent on the number of installed flow entries, thus enabling processing over a large amount of flow sessions.

The impact of attack events on latency is shown in Tab. I, in which the first (second) row reports results at 9Gbps almost all regular (attack) traffic, respectively. The additional impact of attack attempts results in around 1ns average, blocking all attack profiles. This means that the full P4 workflow processing takes few additional ns with respect to the only parser and deparser sections.

Finally, Tab. II reports the NetFPGA hardware resources utilized by P4, consuming around 23% of programmable logic, expressed as LookUpTable (LUT) rate and 37% of memory (RAM). Such results show that a single NetFPGA can support more complex P4 programs, e.g. multi-profile or parallel workflows, thus validating the P4 effectiveness for NetFPGA in the cyber security framework.

VII. CONCLUSION

The P4 language has the potential to become a disruptive instrument to program and customize the data plane of SDN network devices. Multi-layer networks with special focus on the edge segment will be required to assure high level of QoS connecting heterogeneous platforms, therefore they will be a candidate target over which P4 implementations may gain consensus.

A P4-based architecture of a edge packet-over optical node was presented, along with P4 solutions suitable multi-layer networks, designed and implemented to provide dynamic TE enforcement of optical bypass and traffic offloading. In addition, P4 was also exploited to effectively react against DDoS cyber-attacks requiring stateful capabilities, acting as an active cyber security barrier.

The P4 solutions have been experimentally validated on BMV2 P4 switches and NetFPGA boards, showing impressive scalability performance with the size of the P4 program and in terms of switch latency to perform P4 operations, especially in the NetFPGA implementation. For example, only 5us overall switch latency were experienced running the cyber security P4 code, with no performance degradation using even 10^4 flow entries, thanks to the NetFPGA parallel architecture. Noticeably, results show no significant performance degradation with respect to fixed function commercial switches while gaining a remarkable degree of flexibility and open source programmability. All dynamic TE and cyber security solutions have been successfully implemented within the P4 switch without involving the SDN controller for modifying flow rules during networking operations.

This work demonstrated P4 scalability and flexibility in key multi-layer edge node use cases, thus opening the road to innovative and disruptive open-source traffic forwarding and manipulation procedures to be programmed in the data plane of next-generation converged networks.

REFERENCES

[1] F. Paolucci, F. Cugini, and P. Castoldi, "P4-based Multi-Layer Traffic Engineering Encompassing Cyber Security," in *Optical Fiber Communication Conference (OFC)*. Optical Society of America, 2018, p. paper M4A.5.

[2] F. van Lingen, M. Yannuzzi, A. Jain, R. Irons-Mclean, O. Lluch, D. Carrera, J. L. Perez, A. Gutierrez, D. Montero, J. Marti, R. Maso, and a. J. P. Rodriguez, "The unavoidable convergence of NFV, 5G, and fog: A model-driven approach to bridge cloud and edge," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 28–35, 2017.

[3] Y. J. Ku, D. Y. Lin, C. F. Lee, P. J. Hsieh, H. Y. Wei, C. T. Chou, and A. C. Pang, "5g radio access network design with the fog paradigm: Confluence of communications and computing," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 46–52, April 2017.

[4] V. Lopez, J. M. G. Josa, V. Uceda, F. Slyne, M. Ruffini, R. Vilalta, A. Mayoral, R. Munoz, R. Casellas, and R. Martinez, "End-to-end service orchestration from access to backbone," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 6, pp. B137–B147, June 2017.

[5] A. Sgambelluri, F. Paolucci, F. Cugini, L. Valcarenghi, and P. Castoldi, "Generalized SDN control for access/metro/core integration in the framework of the interface to the routing system (I2RS)," in *2013 IEEE Globecom Workshops (GC Wkshps)*, Dec 2013, pp. 1216–1220.

[6] A. Sgambelluri, F. Paolucci, A. Giorgetti, F. Cugini, and P. Castoldi, "Experimental demonstration of segment routing," *Lightwave Technology, Journal of*, vol. 34, no. 1, pp. 205–212, 2016.

[7] A. Giorgetti, A. Sgambelluri, F. Paolucci, F. Cugini, and P. Castoldi, "Segment routing for effective recovery and multi-domain traffic engineering," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 2, pp. A223–A232, Feb 2017.

[8] F. Paolucci, "Network service chaining using segment routing in multi-layer networks," *J. Opt. Commun. Netw.*, vol. 10, no. 6, pp. 582–592, Jun 2018.

[9] L. Velasco, L. Gifre, J. L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, and F. Cugini, "An architecture to support autonomic slice networking," *Journal of Lightwave Technology*, vol. 36, no. 1, pp. 135–141, Jan 2018.

[10] H. Yang, J. Zhang, Y. Zhao, Y. Ji, J. Han, Y. Lin, and Y. Lee, "CSO: cross stratum optimization for optical as a service," *IEEE Communications Magazine*, vol. 53, no. 8, pp. 130–139, August 2015.

[11] F. Paolucci, F. Cugini, G. Cecchetti, and P. Castoldi, "Open network database for application-based control in multilayer networks," *Journal of Lightwave Technology*, vol. 35, no. 9, pp. 1469–1476, May 2017.

[12] H. Yang, Y. Wu, J. Zhang, H. Zheng, Y. Ji, and Y. Lee, "BlockONet: Blockchain-based trusted cloud radio over optical fiber network for 5G fronthaul," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, March 2018, pp. 1–3.

[13] P. Marques, N. Sheth, R. Raszuk, B. Greene, J. Mauch, and D. McPherson, "Dissemination of Flow Specification Rules," *IETF, RFC 5575, Network Working Group*, Aug. 2009.

[14] S. Fichera, L. Galluccio, S. C. Grancagnolo, G. Morabito, and S. Palazzo, "OPERETTA: An openflow-based remedy to mitigate TCP synflood attacks against web servers," *Comput. Netw.*, vol. 92, no. P1, pp. 89–100, Dec. 2015.

[15] W. J. A. Silva, "Avoiding inconsistency in OpenFlow stateful applications caused by multiple flow requests," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, March 2018, pp. 548–553.

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

[17] "P4 Language Consortium," https://p4.org.

[18] "Behavioral Model version 2 (BMV2)," https://github.com/p4lang/behavioral-model, 2017.

[19] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4fpga: A rapid prototyping framework for p4," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17, 2017, pp. 122–135.

[20] "OpenFlow Switch Specification version 1.5.1," https://www.opennetworking.org/software-defined-standards/specifications, 2017.

[21] J. Hyun and J. W. K. Hong, "Knowledge-defined networking using in-band network telemetry," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Sept 2017, pp. 54–57.

[22] W. Braun, J. Hartmann, and M. Menth, "Demo: Scalable and reliable software-defined multicast with bier and p4," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, May 2017, pp. 905–906.

[23] A. Giorgetti, A. Sgambelluri, F. Paolucci, P. Castoldi, and F. Cugini, "First demonstration of SDN-based bit index explicit replication (BIER) multicasting," in *2017 European Conference on Networks and Communications (EuCNC)*, June 2017, pp. 1–6.

[24] P. Voros and A. Kiss, "Security middleware programming using P4," *HAS 2016, Human Aspects of Information Security Privacy, Lecture Notes in Computer Science*, vol. 9750, pp. 277–287, 2016.

[25] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with sdn data plane," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[26] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A survey on the security of stateful sdn data planes," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1701–1725, thirdquarter 2017.

[27] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful openflow applications inside the switch," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014.

[28] C. Cascone, L. Pollini, D. Sanvito, and A. Capone, "Traffic management applications for stateful sdn data plane," in *2015 Fourth European Workshop on Software Defined Networks*, Sept 2015, pp. 85–90.

[29] L. Velasco, A. Sgambelluri, R. Casellas, L. Gifre, J. L. Izquierdo-Zaragoza, F. Fresi, F. Paolucci, R. Martinez, and E. Riccardi, "Building autonomic optical whitebox-based networks," *Journal of Lightwave Technology*, pp. 1–1, 2018.

[30] A. Sgambelluri, J.-L. Izquierdo-Zaragoza, A. Giorgetti, L. Gifre, L. Velasco, F. Paolucci, N. Sambo, F. Fresi, P. Castoldi, A. C. Piat, R. Morro, E. Riccardi, A. D'Errico, and F. Cugini, "Fully disaggregated ROADM white box with NETCONF/YANG control, telemetry, and machine learning-based monitoring," in *Optical Fiber Communication Conference*. Optical Society of America, 2018, p. Tu3D.12.

[31] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network telemetry streaming services in SDN-based disaggregated optical networks," *Journal of Lightwave Technology*, vol. 36, no. 15, pp. 3142–3149, 2018.

[32] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker," *IETF, RFC 2698*, Sept. 1999.

[33] F. Paolucci, A. Castro, F. Fresi, M. Imran, A. Giorgetti, B. Bhownik, G. Berrettini, G. Meloni, F. Cugini, L. Velasco, L. Potì, and P. Castoldi, "Active PCE demonstration performing elastic operations and hitless defragmentation in flexible grid optical networks," *Photonic Network Communications*, vol. 29, no. 1, pp. 57–66, Feb. 2015.

[34] F. Morales, L. Gifre, F. Paolucci, M. Ruiz, F. Cugini, P. Castoldi, and L. Velasco, "Dynamic core VNT adaptability based on predictive metro-flow traffic models," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, no. 12, pp. 1202–1211, Dec 2017.

[35] G. Meloni, F. Paolucci, N. Sambo, F. Cugini, M. Secondini, L. Gerardi, L. Poti, and P. Castoldi, "PCE architecture for flexible WSON enabling dynamic rerouting with modulation format adaptation," in *European Conference on Optical Communication (ECOC)*, Sept. 2011.

[36] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[37] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sept 2014.

[38] A. M. S. Abdelhadi and G. G. F. Lemieux, "Modular SRAM-based binary content-addressable memories," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2015, pp. 207–214.