Journal of Network and Systems Management (2023) 31:52
https://doi.org/10.1007/s10922-023-09744-1

RESEARCH

4MIDable: Flexible Network Offloading For Security VNFs

Benjamin Lewis[1] · Matthew Broadbent[2] · Charalampos Rotsos[1] · Nicholas Race[1]

Received: 8 July 2022 / Accepted: 22 May 2023

Abstract

The ever-growing volume of network traffic and widening adoption of Internet protocols to underpin common communication processes augments the importance of network security. In order to enforce network security policies, network managers adopt a widening set of middleboxes and network appliances to improve traffic monitoring and processing capabilities. The resource requirements to support network security appliances are constantly increasing, making efficiency of these systems an essential aspect. The move toward Software-Defined Networking and programmable data planes offers a mean to offload traffic processing functionalities to within the network itself. To this end, we present the 4MIDable framework: a platform that facilitates the integration of existing middleboxes and monitoring appliances with an SDN (P4) network infrastructure. We also present P4Protect, a 4MIDable agent that protects the network from control plane DoS attacks with negligible impact on control plane latency, and P4ID (P4-Enhanced Intrusion Detection), a 4MIDable agent that offers stateful processing and feedback to unmodified Intrusion Detection System middleboxes and reduces traffic processing by over 80% without affecting threat detection rates.

Keywords P4 · VNF · IDS · SDN

✉ Benjamin Lewis
b.lewis@lancaster.ac.uk

Matthew Broadbent
m.broadbent@napier.ac.uk

Charalampos Rotsos
c.rotsos@lancaster.ac.uk

Nicholas Race
n.race@lancaster.ac.uk

[1] School of Computing and Communications, Lancaster University, Infolab 21, Bailrigg, Lancaster LA1 4WA, United Kingdom

[2] School of Computing, Edinburgh Napier University, 10 Colinton Road, Edinburgh EH10 5DT, Scotland

Published online: 15 June 2023

1 Introduction

The expanding adoption of network technologies in a widening set of everyday use cases have increased the need for network resilience. Middlebox devices have emerged as the de facto solution to improve security and network visibility. They typically process protocol header information across network layers and in non-standardized ways, allowing operators to monitor application level statistics, detect malicious activity and apply access control. Security is a key application domain for middlebox applications, allowing network administrator to drop access and monitoring enforcement points that is compatible with the existing network configuration.

Traditionally, middleboxes rely on hardware acceleration to ensure support for line-rate traffic processing, and thus incur a significant CAPEX. Unfortunately, the increasing size of network infrastructures makes hardware middlebox deployment economically unsustainable. Advancements in virtualization technologies have enabled the wider adoption of Virtual Network Functions (VNFs) as a financially efficient alternative to middleboxes.

VNF appliances provide a great benefit with respect to flexibility and elasticity over hardware-accelerated middleboxes, but software still cannot match the processing speeds of hardware. To bridge this performance gap, the research community have explored the applicability of Software-Defined Networking (SDN) mechanisms as a means to offload processing in the network fabric.

OpenFlow [25] was one of the first attempts to design an SDN mechanism for legacy forwarding devices. Although the research community and even vendors adopted the protocol, its design aimed primarily to match the existing capabilities of network ASICs and the control abstraction supports limited flexibility and scalability. This design can reveal limitations when trying to implement security applications in the data plane, notably in terms of packet-parsing and stateful processing without controller intervention.

More recently, focus in the SDN community shifted towards enabling programmable data planes, not least in the wake of research into programmable switch chips that have performance equal to or greater than fixed function data planes [4]. P4 [5], proposes a Domain-Specific Language (DSL) specification that emerged from these efforts. The language revisits the design of the SDN abstraction, in an effort to allow functional requirements of the control plane to be mapped in the data plane. Using an open-source, high-level language significantly reduces the barrier to entry for defining network behavior. P4 has since evolved, with support from a number of hardware [18] and software [29] platforms. The P4Runtime [32] has also been introduced, providing an alternative to the OpenFlow protocol for programming flexible behavior. Several research efforts [39, 42] have developed new middlebox Domain-Specific Languages (DSLs) that allow offloading onto a P4 data plane. Nonetheless, such frameworks limit opportunities to retrofit existing middlebox appliances, while the design of such DSLs have a strong packet-oriented philosophy that does not allow an easy fit for complex security applications, like stateful IDS [12].

However, security appliances have a varying set of requirements spanning from simple control of forwarding decision, all the way, to application-layer protocol processing. For example, IDS have often focused on a middlebox architecture, sited at network boundaries between Local and Wide Area Networks. With SDN, the transitions between networks may not be as clearly defined. Appropriate placement of network monitoring functions requires an understanding of physical and logical topologies, which are abstract concepts in SDN. Further to this, traditional appliances have finite capacity [14] for traffic processing. Larger, more diverse networks need a means to route traffic to the appropriate middlebox.

This work proposes 4MIDable; a control framework allowing existing security middleboxes, appliances and host-based applications to offload traffic processing and filtering to programmable control and data planes. This is achieved through the implementation of a translation layer and associated interfaces between the middleboxes and the SDN switch interfaces. This serves to give the controller visibility of a wider context and understanding of the network behavior, through the detection and mitigation of network threats. Our approach is designed around P4-programmable data planes and enables their capability to be exposed and customized to suit individual applications. We evaluate the framework in the context of two applications built using the framework: P4Protect, a DoS control plane protection appliance, and P4ID, a system that allows an off-the-self Suricata [38] instance to offload pre-filtering traffic processing in the data plane. Our evaluation utilizes both hardware and software switches and demonstrates that the platform has a minimal overhead on control plane latency and reduces up to 32% on average the CPU utilization of an IDS appliance in a common network deployment scenario. The contributions of this paper are the following:

- We define a minimal architecture and an API for traffic monitoring, filtering and processing, designed to support offloading for security VNFs.
- We present the integration of popular applications—including DDoS attack detectors and the Suricata IDS—with the 4MIDable framework.
- We demonstrate that 4MIDable can minimize service degradation for the control plane during a DDoS attack, and has the ability to reduce the CPU and traffic volumes required by a Suricata instance, while maintaining high detection rates.

For the rest of this paper, we elaborate on related efforts (Sect. 2) and present the design of our 4MIDable system(Sect. 3) and a series of integration scenarios with popular applications (Sect. 4). Finally, we evaluate the scalability and precision of our platform (Sect. 5) and conclude our work (Sect. 6).

2 Related Work

The 4MIDable framework facilitates the offloading of traffic-manipulation operations to programmable data planes, primarily targeting network security applications. In this section, we will explore existing work covering these two aspects.

2.1 Data Plane Offloading

Offloading network processing to programmable network dataplanes and the development of application-oriented network control APIs was a key research topic in the early days of SDN technologies. Relevant attempts extended the functionality of network applications in the fields of routing [27] intrusion detection [41] and policy enforcement [22]. Unfortunately, the fixed design of the OpenFlow protocol offered limited opportunities to scale functionality for real network workloads.

The development of P4 compilers for ASIC-based network devices has rejuvenated research efforts to design traffic-processing offloading mechanisms for middleboxes. OpenFunction [39] is one of the first attempts to develop middlebox systems using a DSL, and provides a compiler supporting a wide range of backend platforms, including P4 and DPDK. Gallium [42] is an *llvm* backend which allows users to compile *Click* programs and offload performance-critical code parts into P4 programs. Relevant efforts aim to develop a holistic framework for middlebox development and expects that middlebox functionality should be implemented from scratch, using the respective DSL language. Our approach remains complementary to these efforts, aiming to develop an offloading architecture supporting compatibility with unmodified network appliances by-design.

2.2 Network Security and SDN

Network programmability has enabled unprecedented opportunities for network security applications to monitor, filter, and process traffic at line rate. OpenFlow-enabled switches allowed a wide range of security middleboxes—like firewalls and address obfuscation—to apply security policies in the data plane with minimal effort. Nonetheless, security applications that depend on deep packet inspection and state require the design of hybrid architectures that couple network fabric with software processing.

A good example of a complex security middlebox that has motivated the development of novel control architecture is the IDS, which inspects packets down to the application layer, using Deep Packet Inspection (DPI). DPI is not a capability intrinsically present within any current version of the OpenFlow specification, as it is focused on packet headers rather than payloads.

CIPA [7], is one of the first attempts to develop a distributed IDS using SDN technologies. The platform uses OpenFlow rules to apply forwarding decisions

regarding malicious traffic in the data plane, and intrusion detection is underpinned by a distributed neural network implemented in the control plane.

SnortFlow [41] proposes a means to integrate Snort-based intrusion detection into a cloud environment by running it as part of the hypervisor, using OpenFlow to provide network re-configurability; though, when their system is under high load, the detection agent (Snort) will start dropping packets. Whilst our approach also uses an IDS, the choice of IDS is flexible, and does not need to run as part of the hypervisor, and can run on bare-metal switches.

TENNISON [12] implements a more pragmatic approach to IDS, using multi-stage monitoring that filters only traffic determined to be of interest to the network administrator to a software IDS in order to apply DPI processing. However, unlike our approach, TENNISON relies heavily on the control plane to determine which traffic should be forwarded to the IDS, due to OpenFlow constraints, whereas 4MIDable relies on the data plane as much as possible.

The HEX Switch [33] introduces an OpenFlow switch architecture, allowing users to develop protocol extensions—including IDS features—as part of the forwarding logic. However, this requires specific hardware platforms and support the additional OpenFlow security actions.

Emerging technologies, such as P4, have overcome packet inspection limitations with the introduction of further programmability with significantly richer capabilities [18] [17] [16]. Achieving DPI with P4 remains challenging, as the focus of the language is on fixed-length bit-fields. While the language [13] does support variable-length header fields, the support of hardware devices, especially ASICs to match based on these fields, is limited. Ndonda et al. [28] describe intrusion detection for Industrial Control Systems with white- or blacklisting functionality for a specific protocol using P4. However, the work relies heavily on the control plane for flow processing. By contrast, in our approach, the only control-plane based processing is to install rules into switches once the IDS has made a detection. This will then continue to forward malicious traffic to the IDS or block it.

Gray et al. [14] use P4 switches to extract features for ML processing, achieving traffic support up to 100Gbps and improving detection rates in comparison to a Suricata instance. In contrast to our work, where we focus on the existing IDS to perform the detection role, rather than using P4 to extract metadata features. They then compare their results to the performance of Suricata, whereas we focus on the integration of existing appliances.

Dao et al. [9] implement a neural network model into the P4 dataplane, using a BMV2-based switch. The authors use this model to explore the impact of different model parameters on applying classifications on incoming traffic.

P4DDPI [1] proposes a Intrusion Detection System for P4 to detect malicious domain names in DNS requests, and the authors show that it is capable of achieving line-rate on the Tofino platform. To achieve the full parsing of DNS headers, the authors rely on a process of recirculation, which can incur performance penalties.

P4-ONIDS [37] explores a similar approach with respect to IDS offloading as 4MIDable. The system allows installation of IDS rules into the data plane, by developing a compiler for IDS rulesets into 5-tuple matches, as well as, filtering the first

N packets for each flow. The system uses a count-min sketch algorithm, for efficient lookup memory use, though which incurs slow reconfiguration times.

P4-ONIDS shares the closest similarity with our work, however, they rely partly on a process of compiling existing Snort rules into 5-tuple rules to be installed into switches. However, the majority of traffic uses well-known ports, so the rule-based approach can be overly selective. The authors acknowledge this, and propose their count-min sketch based approach. Their approach uses less memory than P4ID, but, is not demonstrated in hardware switches. Ours is evaluated both on the software-based BMV2 and hardware Tofino platforms. In our approach, we also use the detection output of the IDS to steer ongoing forwarding decisions.

Table 1 compares a number of the discussed frameworks, and categorises them based on platform, whether they are designed to be extensible, whether they can accept feedback from existing security appliances, whether they are open source, and whether they can operate independently of the control plane.

3 4MIDable Framework

The 4MIDable architecture is depicted in Fig. 1, and consists of three layers: the *Data Plane Layer*, consisting of the P4 network fabric programmed using the 4MIDable P4 pipeline; the *Control Plane Layer*, which translates the middlebox offloading requirement into effective network configurations; and the *Application Layer*, consisting of the middlebox appliances and the adaptation agents. The 4MIDable framework is implemented in Golang [10]. This offers a number of benefits for parallel processing and flexibility for the creation and instantiation of monitoring plugins.

Through the next sections, we will describe each layer of the framework in depth, using one of our existing plugins to demonstrate how each component of the system is used.

Table 1 Framework comparison

System name	Platform	Extensible	Appliance feedback	Open source	Control-plane independence
4MIDable	P4 (BMV2/ Tofino)	Yes	Yes	Yes	Yes
P4-ONIDS	P4 (BMV2)	No	No	No	Yes
P4DDPI	P4 (Tofino)	No	No	No	Yes
CIPA	OpenFlow	No	No	No	No
SnortFlow	OpenFlow	No	Yes	No	No
TENNISON	OpenFlow	Yes	Yes	Yes	No
Gray et al	P4	No	No	No	No
Ndonda et al	P4	No	Yes	No	No

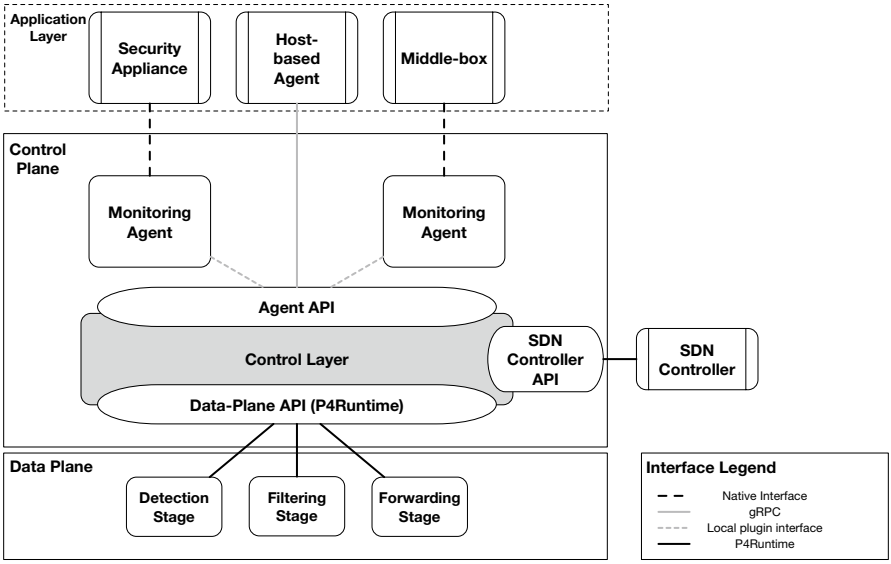


Fig. 1 4MIDable framework architecture

3.1 Data Plane Layer

One of the key benefits of the P4 language is the ability to push application requirements into the switch pipeline organization. 4MIDable exploits this functionality by designing a reference P4 pipeline supporting generic packet processing (akin to the approach taken by OpenFlow), and compatible with with the P4 driver of the ONOS controller. The pipeline supports by design extensions on several points in the pipeline that allows individual middlebox appliances to embed custom traffic-processing operations.

The reference 4MIDable P4 application implements a set of core parsing, lookup and forwarding functionalities, that allow compatibility with commodity SDN controllers. These core functionalities, outlined in Fig. 2, are exposed by providing the SDN controller the P4Info information associated with the unmodified reference pipeline. This allows the controller to be compatible with any pipelines that are a superset of the functionality offered in the reference pipeline. To this end, we have based our pipeline on that as used with an off-the-shelf SDN controller, ONOS [31]. By ensuring that our reference pipeline is compatible with ONOS, we can take advantage of an existing SDN controller for forwarding and routing.

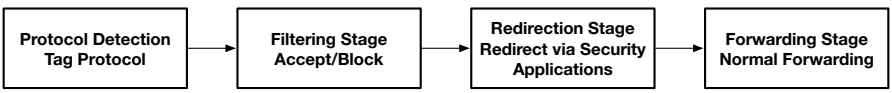


Fig. 2 4MIDable reference pipeline architecture

To enable this integration, we use P4 annotations to ensure that identifiers for tables and actions remain consistent across compiler runs. Using these identifiers enables the programmer to insert the required functionality at the right stage in their application.

The reference pipeline consists of four stages. The *parsing stage* extracts headers fields for common protocols: including Ethernet, IPv4, IPv6, TCP and UDP. Following this, the *detection stage* allows the expression of custom stateful monitoring and processing functionality, with the use of meters, registers and counters. Applications can attach actions to packets at this stage to drop or re-direct traffic, which will be applied at the forwarding and redirection stages. The *filtering and redirection stage* is designed to apply encapsulation, mirroring, redirection or drop traffic actions, assigned in the previous stage. This stage provides a set of fixed packet actions, designed to support basic traffic-manipulation actions for security middleboxes. The *forwarding stage* is inspired by the 'basic' ONOS [3] pipeline, and can be used by middleboxes to inject custom traffic-manipulation logic in the P4 pipeline; for example, adding traffic thresholds using registers or meters. Using this as a reference point offers us information on the minimum capability required for operation with the SDN controller. This can be extended further to suit network administrator requirements.

When adding functionality to the reference pipeline, programmers and network administrators are afforded the flexibility to determine how resources on the network should be allocated between different resources and stages. In the reference pipeline, we have described the stages at which programmers would be likely to be adding this functionality, primarily in the detection and filtering stages. Administrators can also optionally inhibit other stages of the pipeline from executing for a given packet, if certain criteria have been met. This however would reduce the SDN controller's visibility of particular traffic flows, which may impact upon traffic-management schemes that are in use on the wider network.

3.2 Control Layer

```
type Packet struct {
  Raw        *[]byte
  Reason       uint8
  Is_clone       bool
  Ingress_port uint16
  Process_done  bool
  Send          bool
  Egress_port uint16
  DeviceId    uint64
  MgId        uint16
}
```

Listing 1: Packet Metadata

```
Type PacketInHandler interface {
  HandlePacketIn(p *Packet) (*Packet, error)
}

type DigestHandler interface {
  HandleDigest(in *StreamMessageIn) error
}

type IdleTimeoutHandler interface {
  HandleIdleTimeout(in *StreamMessageIn) error
}

type Monitor interface {
  GetNextUpdate() (Update, error)
}
```

Listing 2: 4MIDable Agent Interface

The Control Layer virtualizes access to the P4 pipeline across multiple middlebox appliances. The controller connects to all network switches, using the P4Runtime API and exposes a single Northbound gRPC interface. This single interface encapsulates native P4 messages with additional metadata, notably plugin identifiers so that data can be passed to the appropriate monitoring plugin.

The Control Layer uses an internal state database to store its configuration, allowing the controller to gracefully restart and reconfigure. These configurations include running plugins installed flow rules, managed switches and their current pipeline configuration. This state database can also be shared between instances for horizontal scaling.

The *SDN Controller Interface* is a mechanism to allow 4MIDable developers to integrate control applications available in modern SDN controller distributions. It provides a means of integrating 4MIDable pipelines with forwarding behaviour managed by a commodity SDN controller. With this, we abstract the monitoring and pipeline management away from the SDN controller, instead presenting a pipeline equivalent to that described by our reference pipeline.

With the reference pipeline being based on, and compatible with, the ONOS 'basic' pipeline, this affords us compatibility with any ONOS application with support for the 'basic' pipeline. This lowers the barrier of entry for implementing P4ID in networks already using an SDN controller, while also greatly increasing the capability of networks using a combination of 4MIDable and ONOS.

To provide this interface, we implement a P4Runtime server, which is the same interface as used to communicate between a controller and P4 switches. 4MIDable emulates interfaces for each switch, allowing the controller to discover and manage network topology. When table entries are sent from the SDN controller, these are passed to the control plane to be installed into the appropriate section of the running pipeline. Statistics collection and events are handled in a very similar manner: the SDN controller will receive the appropriate P4 entities after they have been processed by our control layer. This processing is only to ensure that any identifiers or data are consistent with the reference pipeline. Note that the SDN controller has an

abstracted view of the underlying pipeline, and can only interact with it as though it is the reference pipeline. This means that the SDN controller does not have access to any plugin-specific stages, tables, or other entities, so any changes to these will not impact upon the SDN controller itself, maximising compatibility.

3.3 Application Layer

A key goal for 4MIDable is to support both unmodified middlebox appliances and to offer a offloading API that can be invoked by remote middleboxes. For the latter, 4MIDable offers a rich gRPC northbound API, that allows applications to inject forwarding decisions in the data plane (*e.g.* block traffic from a subnet), monitor and filter traffic (*e.g.* redirect traffic for port 80 to a middlebox) and process traffic (*e.g.* masquerade honeypot traffic). As the API is implemented using Google protobuf, it is compatible with all major programming languages and integration success essentially depends on source-code modification.

The offloading API provides low-level access to P4 *events*. These are messages including Packet-Ins, Digests, Counter and Meter updates: mechanisms essential to gain precise traffic monitoring information. Access to such APIs is exposed via interface classes and an application can gain access by registering a handler via the NBI. Messages are sent to handlers in a sequential way and handler ordering is achieved during registration using priorities. Listing 2 presents the structure of handler interfaces, while P4Runtime messages sent to the handler are coupled with a metadata struct—shown in Listing 1—containing switch information and flags to control further message processing by other handlers.

Nonetheless, for some appliances, code changes are not possible, either due to the complexity of the programming model or due to binary formats. In order to support such systems, 4MIDable uses *middlebox agents* to link middleboxes with the framework. Agents are middlebox-oriented and serve two main purposes: to translate middlebox configurations into a set of *static* offloading configurations during boot, and to *dynamically* change offloading configurations based on the middlebox logging information during run-time. This on-the-fly processing allows plugins to harness middlebox detections and respond to changes in network conditions. Monitoring plugins use the *Monitoring API* to communicate with the rest of the system.

Agents interact with the rest of the framework using an *Agent API*, which is used to manage the life cycle of agents as well as instantiating flow rules in the data plane and passing traffic and statistics up to 4MIDable agents. Depending on the appliance format, agents can operate either in local mode, where they run as part of the controller, or in remote mode. Middlebox agents have mandatory and optional interfaces. The mandatory interfaces are the *Start* and *Stop* interfaces. These are used to set up and tear down any forwarding rules that are required by a given monitoring plugin. Static configuration can be parsed and installed while this interface is running. These interfaces are intentionally minimal to offer the greatest flexibility to plugin authors. Plugin authors can find sample code and documentation on our GitHub repository [2].

Finally, middlebox agents can either be pipeline-agnostic or pipeline-dependent. Pipeline-agnostic agents are those which can be implemented with no modification to the reference pipeline. Pipeline-dependent agents require additional functionality from the data plane. This functionality includes additional packet header processing, data collection and stateful packet processing. Pipeline dependent applications have the option of either requiring a particular pipeline configuration, or they can specify 'tags' which they require. Pipeline configurations specify the tags they support. With this, the control layer can reconcile which pipeline configuration to install to support all applications currently requested.

4 4MIDable Applications

To better demonstrate the architecture of 4MIDable, this section presents the design of two agents, enabling integration with common network security appliances. Our agent implementations demonstrate that 4MIDable can support the offloading requirements of a wide range of security appliances and improve both resource utilization and network responsiveness.

4.1 P4Protect

P4Protect is a native 4MIDable agent, offering protection against control-plane denial-of-service (DoS) attacks to SDN networks. Such attacks can not only overload a switch's control-plane channel [20], but our evaluations show that they can render the controller unusable for the duration of an attack. P4Protect is designed to offer general-purpose DoS protection and can be used in conjunction with any number of middlebox agents and SDN applications.

P4Protect tracks the rate of new connections recorded in the control plane at any given time and applies threshold-based rate limiting, using stateful packet processing. This allows the control plane to remediate service delivery during attacks by black-holing malicious traffic. Communication between the switch and controller is not interrupted when the P4Protect protection engages, and other middleboxes or controllers can inspect and modify black-listed hosts by polling register-state to determine whether traffic is being blocked.

Operating P4Protect in an autonomous mode, where no modification is made to the controller, requires configuring maximum traffic rates at compile-time. This trades flexibility for ease of deployment.

4.2 P4ID

P4ID is one of the most advanced 4MIDable agent designs and allows network offloading for unmodified Suricata IDS instances. The plugin reduces the volume of traffic processed by the IDS, while retaining a level of detection equivalent to no pre-filtering being in place. P4ID improves middlebox efficiency by exploiting the

principle that an IDS will generate an alert based on a packet or sequence of packets received near the start of a flow.

The plugin designs extend the based P4 pipeline with additional stages to enable stateful packet filtering. The new stages track new flow arrivals and forward a programmable number of packets from the beginning of each flow to the IDS host. Once the threshold of packets to send has been exceeded, the switch allows traffic to be processed through the switch unimpeded. Furthermore, if the flow triggers an IDS alert, the pipeline allows the agent to carry on packet delivery to the IDS or even enable packet drops. Finally, the P4 program maintains a per-flow timeout value, which is used to repeat packet forwarding to the IDS, when the interval between two consecutive packets exceeds a programmable threshold.

Flow monitoring is expensive in hardware, and the P4ID pipeline extension implements an efficient per-flow state management mechanism. Trying to implement IDS signatures as table entries in the data plane can prove ineffective, as they typically specify application layer packet contents. P4 data planes can parse the application layer of a packet using variable-length packet headers, but this increases parsing and lookup complexity significantly, and is best done in software. Thus, the P4ID pipeline adopts a more coarse-grain approach to classifying packet flows using the 5-tuple, which is much more efficient compared to a signature searching for a specific sequence in a packet payload. If an alert is generated, a flow rule is installed into the monitoring table, ensuring that any matching packets continue to be sent to the IDS. This improves accuracy as the IDS continues to receive traffic associated with an alert.

The pipeline uses a fixed-width register array to maintain flow state, and a 5-tuple hash is used to associate array entries with flow state. Each flow is assigned two registers. The first register stores a receipt timestamp for the last flow packet, while the second register counts the number of flow packets. The packet counter is used to decide if a packet should be sent to the IDS, by comparing the register value against the number packet expected to be sent to the IDS. Furthermore, the timestamp register is updated on the receipt of every packet and is used to implement the reset mechanism of the packet filtering mechanisms. If the interval between two consecutive packets is above a threshold, then the program resets the packet counter to zero, which resets packet filtering.

In order to improve efficiency, a number of optimizations are adopted. Firstly, register entries are capped to 262144 entries, based on an 18-bit–wide hash. while this hashing approach is subject to collisions, we believe that this –evident also in our experiments using real data– has negligible impact on detection rates. Secondly, the hash uses 104 bits of data, or 2^{104} possible combinations (Source and Destination IP addresses, ports and protocol identifier), and generates an 18-bit identifier. This gives 2^{86} possible combinations where a given hash input would equate to the same hash output. This flow collision would also have to occur within the timeout period, otherwise the flow would be directed towards the IDS for monitoring anyway. Therefore, we put forth that the saving in switch memory is an acceptable trade-off compared to the potential for collisions. Thirdly, the timeout mechanism in the flow filtering mechanism ensures visibility of long-lived benign flows, as well as mitigating against flow collisions. Flow collisions can be mitigated against further

by parsing the TCP header, and always resetting the packet threshold if a packet has the SYN flag set, which signifies the start of a connection. However, this eviction strategy would only be effective for TCP traffic. Finally, threshold values can be programmed by the control plane, stored as metadata fields to be used for comparison. This design choice improves filtering flexibility and allows IDS systems to respond based on network demands for accurate detection vs traffic loads to the IDS.

In total, the registers used by P4ID occupy 262144 48-bit values at full utilization, used for capturing timestamps, and packet counts can use either 8 or 16 bit registers, depending on requirements. This therefore requires 1.5iMB of memory used for timestamps, along with a further 0.26–0.52MiB being used in the implementation for the BMV2 software switch. Note that this can be reduced further by reducing the accuracy of timestamps or the number of flows. In our implementation for the Tofino platform for example, we reduce timestamp memory occupancy by 50%, by truncating a 48-bit nanosecond timeout value to a 32-bit value instead. With this reduction, we can offer a high-resolution timeout, but with a maximum value of only 4295 milliseconds (the maximum value a 32-bit register in nanoseconds can hold). The alternative is to reduce accuracy by discarding the lower 16 bits of the value, equivalent to 0.06 milliseconds.

In order to integrate the Suricata IDS with 4MIDable, the agent uses a log parsing module, to translate Suricata alerts into a 5-tuple table entry into the switch, used to dynamically decide if further packets should be forwarded to the IDS. Specifically, the agent monitors the IDS' "Event", which produces alerts in a JSON format. The logging format gives a message with the alert ID, the message associated with the detected signature, and the 5-tuple of the flow that has been alerted on. This flow is parsed into the constituent source and destination ports and addresses, and a *Flow Request* is inserted in the flow table of the P4ID register table. The agent runs on the same host as the IDS, and operates proactively, relying solely on IDS alerts. Threshold values can be programmed from the control plane, and themselves read into metadata fields to be used for comparison. By using the control plane, rather than hard-coding the threshold values into memory, the data plane application can be made more flexible, and can respond based on network demands for accurate detection vs traffic loads to the IDS.

P4ID demonstrates the ability of the 4MIDable platform to integrate existing middleboxes without any code modifications. Furthermore, it takes advantage of Golang's native support for JSON parsing into objects, in conjunction with an external library to monitor file updates. Flow requests based on these alerts are simply built using helper methods and passed into the control layer.

5 Evaluation

This section evaluates the 4MIDable framework using two common middlebox use cases. We begin with P4Protect, demonstrating the ability to mitigate control-plane volume attacks, with minimal latency overheads. We then demonstrate the impact of hardware filtering offloading on detection precision and processing load using the P4ID agent. All experiments run on a Dell server (Intel Xeon 4114, 10C,

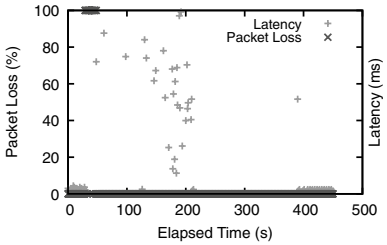
32 G RAM) and we use the Mininet [21] emulation platform to realize our topologies. In our experiments we utilize two P4 switches: The Behavioural Model V2 (BMV2) [29] reference P4 implementation, a tool "primarily designed for developing, testing and debugging P4 data planes" [30], and the Intel Tofino [18] switch, a commercially available P4 hardware switch implementation.

5.1 Latency Evaluation

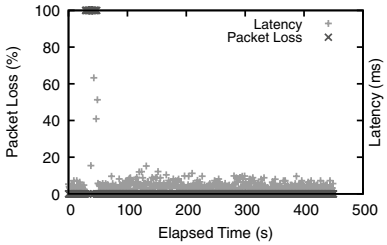
In order to evaluate the impact of our 4MIDable framework on the control plane, we measure the impact of the P4Protect agent on an a modified ONOS [3] learning switch application. For this experiment, we utilize a simple topology consisting of a BMv2 instance connecting two hosts. One of the hosts executes a control plane DoS attack, by injecting ARP packets with randomized MAC and IP addresses at a fixed rate. The attack can reduce control channel responsiveness, when using a reactive control application [20]. Macias *et al*., [24] demonstrate that 1218 packets/sec are sufficient to disrupt the normal behaviour of a controller. Our ARP attack generates 3000 packets/sec (672 kbps), a rate sufficient for a single host to saturate the control channel between the BMv2 switch and the ONOS controller. The second host generates an L2 RTT estimation probe with a rate of 1 packet/second and, in parallel, uses the PCAP library to capture traffic and record the estimated propagation delay. To measure RTT, each measurement packet contains a special EtherType value, a timestamp and a unique packet ID. The BMv2 pipeline is configured to forward all RTT packets to the ONOS controller as `PacketIn` messages. The controller appends a receipt timestamp to the packet and retransmits it back through the receiving switch port using a `PacketOut` message.

Each experiment runs for 500 sec, with the attack starting 30 s after the start of the RTT estimation probe and lasts for 300 s. For each experiment, we record the CPU utilization of the ONOS controller, as well as the latency and packet loss of the heartbeat probe. We utilize two experimental configurations: an *unprotected configuration*, with the switch connecting directly to the ONOS controller; and a *P4Protect configuration*, which uses a 4MIDable instance with an P4Protect agent to process all control plane communications between the controller and the switch.

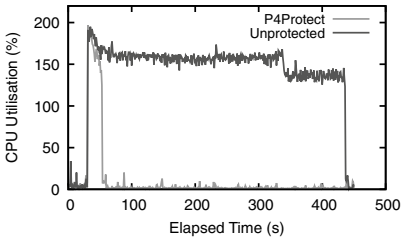
Figure 3 reports the `PacketIn` latency and loss, and the average controller CPU utilization for each experimental setup. Based on the latency and packet loss measurements we highlight that in the unprotected configuration (Fig. 3a), the control plane RTT rapidly climbs, until responses stop being received by the RTT measurement host due to control channel overload. In parallel, CPU utilisation (Fig. 3c) on one core remains at maximum, suggesting that the underlying implementation on the controller may be thread-limited. In contrast, the P4Protect configuration experiences a high utilization and latency only for the first seconds of the attack, but operation soon returns to normal levels. The initial spike is a result of the time that P4Protect requires to detect the attack. Finally, we highlight the impact of 4MIDable on control plane latency is minimal during normal operation. The RTT probe estimates on both configurations a mean RTT value of 50 msec during normal operation with low variance.



(a) Latency and packet loss (Unprotected configuration)



(b) Latency and packet loss (P4Protect configuration)



(c) CPU utilization (Unprotected and P4Protect configurations)

Fig. 3 Latency, packet loss and CPU utilization during a DoS attack to the control plane of a switch while using a learning switch ONOS application with and without the 4MIDable P4Protect agent

In the scenario presented, our filtering is based on individual switch ingress ports. If an attacker attempts to overload the controller from multiple ingress ports, they could have a more disruptive impact upon the controller. This is because, with port-based protection, the threshold has to be met on each port before the protection engages, which is designed to avoid disruption to hosts connected to other switch ports. To counter this, it is possible to add a second filtering stage and a set of additional registers, which ensure that no combination of ports is sending sufficient traffic to overload the controller through a particular switch.

5.2 Detection Precision Evaluation

In this section we evaluate the precision of the filtering mechanism in 4MIDable using the P4ID agent with the Suricata IDS. Our experiments evaluate the performance of 4MIDable using both hardware (Tofino) and software (BMV2) switch platforms. Specifically, we use the open-source P4 Behavioural Model, a software switch that serves as a reference implementation of a P4 switch.

Figure 4 shows the evaluation environment used with the P4ID agent, wherein we have traffic arriving to the switch which is then flagged or allowed to proceed as 'bypass' traffic. For the BMV2-based environment, the switch is connected to a series of virtual Ethernet (veth) pairs, while the Tofino switch is connected directly to a series of physical network interfaces connected to the evaluation host. In order

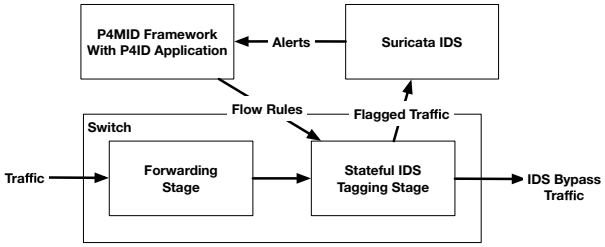


Fig. 4 P4ID experimental topology

to emulate network attacks, we utilize open-source traffic traces, replayed to the switch's input interface using the tcpreplay [40] tool. Traffic is captured from the IDS and Bypass points using the tcpdump tool. We capture all traffic to ensure that the switch has processed all packets successfully. In each case, the switch is managed using the P4ID agent and the framework uses the P4Runtime switch interface.

In our experiments, we use two popular IDS benchmarking datasets: the ID2T dataset, which uses the Intrusion Detection Dataset Toolkit (ID2T) tool [8] to inject artificial attack traffic in legitimate traffic network traces from an office network, and the CICIDS2017 dataset [35], a contemporary and publicly available dataset.

The ID2T tool offers 6 attack types. The **Botnet** attack emulates a peer to peer botnet. The **DDoS** attack replicates traffic associated with the slowloris [36] attack, which relies on holding a large number of open connections to a webserver for as long as possible. The **EternalBlue** attack emulates the network behaviour of a ransomware tool [26]. The **Port Scanning** attack recreates an NMAP [23] network scanning session. NMAP can create many new connections in a short period of time, as it attempts to connect to services on successive port numbers. The **Sality** attack recreates traffic that reflect the behaviour of a malware, which can also act as a peer-to-peer botnet [11]. Finally, the **SQL Injection** [15] attack, which aims to damage, gain access to or infiltrate databases. The baseline trace has a duration of 5-minutes, containing 37,000 packets. Our base capture was captured on an office network running a variety of hosts. The base capture is used to provide consistent benign background traffic between each class of attack. Table 2 lists the range of the attacks included within the dataset as well as the number of packets available in each attach trace.

The CICIDS2017 dataset [35] is a contemporary and publicly available dataset, comprising of a number of benign packet traces with the attack traffic from a range of attacks. Each packet trace is named by the day of the week and contains one or more attacks at specified times in the capture file. The attacks in this dataset include **Botnet**, **Brute-force**, **Denial-of-Service** and **Web-based** attacks. Table 2 lists the range of the attacks included within the dataset, which are the attacks we use to evaluate our pre-filtering approach. Table 2 lists the range of the attacks included within the dataset, which are the attacks we use to evaluate our pre-filtering approach.

For each dataset, we prepare appropriate intrusion detection rules, derived from the EmergingThreats [34] ruleset. We use publicly available rulesets to demonstrate that our approach is effective, even with off-the-shelf configurations. We also filter

Table 2 Attack type and size used from the ID2T and IDS2017 datasets

ID2T			
Attack	(10^3) Packets	Attack	(10^3) Packets
Benign	37	Port Scan	38
Botnet	37	Sality	37
DDoS	46	SQL Inject	43
EternalBlue	37		
IDS2017			
GoldenEye	361	Port Scan	1485
SlowHTTPTest	374014	SlowLoris	424
Web Brute-force	1345	SQL Inject	51
FTP Brute-force	6315	Web XSS	233
HeartBleed	427		

data-set traces that are undetectable by Suricata using published intrusion detection rulesets or are not suited to signature-based detection, *e.g.* in encrypted streams or when designed to evade signature-based detection [19]. Finally, we run Suricata against each testing trace and record the total number of raised alerts and signatures, which reflect the *baseline* set of alerts and signatures that should be raised by the P4ID agent.

It is worth highlighting that the two datasets provide unique features to analyse different aspects of the 4MIDable architecture. The ID2T dataset contains synthetically generated attack traffic that matches the characteristics of real attacks with small durations. As a result, some rule-sets may misclassify attacks. For example, the botnet attacks cannot be detected with our ruleset, while the Distributed DoS attacks are mislabelled as a port-scanning attack, due to a high connection rate. The CICIDS2017 dataset, on the other hand, contains a trace containing attacks spaced over the course of 8 h (equivalent to a working day). When replaying attacks, we have divided the dataset into a series of individual captures representing all traffic on the network at the time of the attack, and for a fixed period before and after the attack.

For this set of experiments, we monitor three IDS metrics: the number of attacks detected by the IDS configuration, the amount of packets processed during operation and the CPU utilization of the IDS during operation. Our aim is for the 4MIDable IDS system to raise the same number of Suricata signatures as the configuration that runs Suricata without any traffic filtering. Furthermore, in our analysis, we distinguish between IDS alerts and signatures. A signature describes an attack fingerprint and it can raise multiple alerts during an attack.

Figure 5 depicts the achieved detection rate when using the P4ID platform with the ID2T dataset, as well as the ratio of traffic that 4MIDable forwarded to the IDS. The best possible detection is the closest value to the baseline figure, where all traffic is sent via the Intrusion Detection System. In every case, the best possible detection is achieved with a reduction in traffic being processed by the IDS in excess of 50% on the BMV2 platform, and a traffic reduction of over 70%

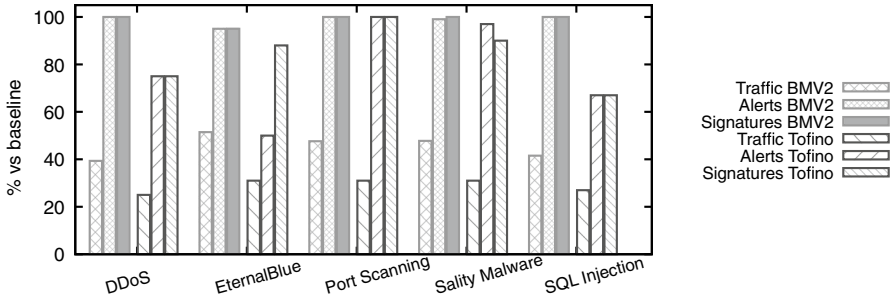


Fig. 5 Comparison of the number of alerts and signatures raised by a Suricata IDS when using the 4MIDable P4ID agent and the ID2T trace on the BMv2 and the Tofino P4 switch frameworks

on the Intel Tofino platform. The figure also highlights that the number of alerts generated by the IDS is reduced. As mentioned previously, an alert refers to the number of occurrences of a given signature. In each case where alerts are lower, we still detect all attack signatures, and each attacker is identified by the IDS successfully.

The EternalBlue attack [6] is a representative ID2T attack type (network-based exploit which installs backdoors on vulnerable hosts) with a reduced rate of alerts, which is still detected in multiple stages. These stages in chronological order are as follows: (1) the vulnerable protocol in use is identified as is the associated host; (2) the installation of the DoublePulsar backdoor is then detected, along with the source and target of that attack; (3) the external address of the attacker is also provided in the logs; (4) the EternalBlue attack and response are then both identified by the IDS, along with further traffic associated with the exploitation. The missing signature is a false positive describing a port-scan on the network.

In all other cases (other than the DDoS), 4MIDable achieves the same level of signature detection as the baseline ID2T experiment, and in the majority of cases, we achieve full detection of alerts. We suggest that when using the P4ID to filter traffic, network administrators should consider whether existing thresholds could be lowered to suit changes in traffic profiles.

Figure 6 depicts the IDS signature and alert detection rates when using the IDS2017 dataset in conjunction with the P4ID platform. In general, we see that the Intel Tofino platform offers the greatest reduction in traffic, with an average reduction of 67% across all tested datasets. With this reduction, we continue to detect 97% of attack signatures. BMV2 on the other hand offers a decrease in traffic of 70%, yet signature detection falls to only 91%.

We focus primarily on the number of signatures detected, as this is representative of the attack patterns and behaviours being correctly detected by the IDS (as used in conjunction with the P4ID platform).

Figure 6 also shows the number of alerts detected by each platform compared to the baseline, where all traffic is directed to the IDS. With alerts, we expect that the number of alerts being generated may be lower, as overall traffic reaching the IDS is reduced. This reduction is particularly prevalent in port scanning

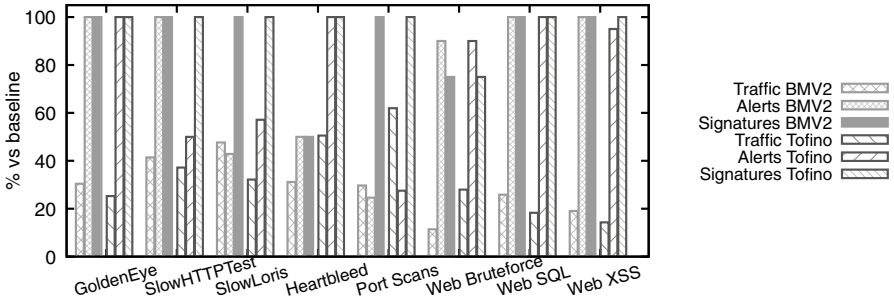


Fig. 6 Precision of the P4ID 4MIDable agent using the IDS2017 traces with the BMv2 and Tofino switches

and denial-of-service attacks, where low volumes of traffic are automatically whitelisted if individual flows fail to alert.

The Web Brute-force attacks are one case where not all signatures are detected. In this case, the signature that isn't detected is a false positive associated with an application that isn't running on the network. The HeartBleed attack is another case where not all signatures are detected when running on the BMV2 platform. The alert that fails to be detected pertains to a vulnerable client receiving malicious data from a server. The Tofino platform successfully detects this attack. However, this comes at a cost: the Tofino implementation re-directs 50% of traffic to the IDS for this dataset, compared to BMV2 redirecting only 31% of traffic.

When compared to P4-ONIDS, we see that our approach gives us significantly better alert detections in some cases, the authors do not provide results for signature detection. With P4-ONIDS, the authors give their headline figure as a traffic reduction of 60% for accuracy of over 90% in alert detection. In our case, we can achieve 100% detection of alerts and signatures for the GoldenEye attack, with only 25% of traffic directed towards the IDS on the Tofino platform, or 30% of traffic on the BMV2 platform. When comparing overall figures, their lowest detections are approximately 10% of alerts compared to when no filtering is in place. In the same situation, with a 50 packets per flow, we achieve an average of 70% of alerts and 87.5% of signatures, with an average of 48% of traffic steered towards the IDS.

5.3 Resource Efficiency Evaluation

The reduction of traffic volumes forwarded to the IDS by our P4ID agent offers a significant benefit in terms of the the IDS CPU load. To evaluate this platform aspect, we run the experimental configuration described in the previous section and increase the traffic rate of our testing host. The high-load scenario uses the benign dataset from the IDS2017 dataset, replayed at a high rate (450 Mbps), alongside the malicious attack dataset being replayed in real time. The malicious dataset is replayed in real-time, to allow IDS rules which rely on timed thresholds to function correctly. The malicious datasets have a peak rate of 16 Mbps. The increase in traffic

volume is essential in order to match a more realistic traffic found in production networks.

We execute both our baseline and P4ID IDS configurations and record in each run the average CPU utilisation for the duration of the experiments. Table 3 shows the difference in average CPU utilisation for each attack dataset between the two experiments. From the results, we highlight that the CPU utilization of the P4ID configuration reduces the IDS load by an average of 31.9%, and in all cases, the IDS continues to detect all attack signatures correctly. It is worth highlighting that any load reduction depends significantly on traffic characteristics and traffic rates. For example, a network carrying long-running high-volume flows, can achieve better results. The employed dataset consists of mixed workloads, reflecting a mix of applications typically found in a production enterprise environment.

6 Conclusion

Middleboxes are increasingly becoming the de facto mechanism to apply security polices in networks. Recent advancements in network programmability enable unprecedented opportunities to improve the efficiency of middleboxes by offloading traffic processing and filtering to the network fabric. P4 is the latest advancement in the field of SDN, allowing applications to map processing requirements in the network fabric, using a dedicated DSL.

4MIDable is a middlebox framework supporting traffic processing and filtering offloading on programmable data planes from unmodified security middleboxes. The platforms offers a versatile offloading API, that can be used with both native network appliances, as well as, unmodified applications with the agent of 4MIDable agents. In parallel, the 4MIDable platform virtualized access to the P4Runtime, thus allowing middlebox appliances to co-exist with existing control applications running on a P4 controller. To demonstrate the capabilities of the platform, we present two 4MIDable agent implementations: P4Protect offers a DoS protection service, and P4ID allows traffic filtering offloading from unmodified Suricata IDS instances. Our evaluations show that 4MIDable has minimal impact on data plane latency, while PAID can reduce traffic being processed by an IDS by over 70%, while maintaining high detection rate of attacks and network threats.

Our future work includes developing further novel applications, including Denial-of-Service mitigation, as well as furthering the development of the 4MIDable

Table 3 CPU reduction for Suricata with offloading

Dataset	CPU Average Delta	Dataset	CPU Average Delta
GoldenEye (DoS)	−26.4%	Web - Bruteforce	−25.8%
SlowHTTPTest (DoS)	−33.4%	Web - SQL	−28.4%
Slowloris (DoS)	−25.3%	Web - XSS	−25.3%
Heartbleed	−28.6%	Port Scans	−34.5%

framework itself. At present, network administrators must assemble pipelines to suit the applications they are using; our next step will be to automate pipeline assembly from a series of artifacts.

Author Contributions BL is the primary author of this paper. NR, MB and CR contributed to the study conception and design. The first draft was written by BL, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Data Availability Not applicable.

Code Availability The code for 4MIDable is available on GitHub.

Declarations

Competing interests The authors have no competing interests to declare that are relevant to the content of this article.

Ethical Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication Not applicable.

References

1. AlSabeh A, Kfoury E, Crichigno J, et al (2022) P4ddpi: Securing p4-programmable data plane networks via dns deep packet inspection. In: Proceedings of the 2022 Network and Distributed System Security (NDSS) Symposium, pp 1–7
2. Benjamin Lewis (2017) 4MIDable agent GitHub Repository. https://github.com/p4lang/PI
3. Berde P, Gerola M, Hart J, et al (2014) Onos: towards an open, distributed sdn os. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, pp 1–6, 10.1145/2620728.2620744
4. Bosshart, P., Gibb, G., Kim, H.S., et al.: Forwarding metamorphosis: fast programmable match-action processing in hardware for sdn. ACM SIGCOMM Comput. Commun. Rev. **43**(4), 99–110 (2013). https://doi.org/10.1145/2534169.2486011
5. Bosshart, P., Daly, D., Gibb, G., et al.: P4: programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev. **44**(3), 87–95 (2014). https://doi.org/10.1145/2656877.2656890
6. Boyanov, P.: Educational exploiting the information resources and invading the security mechanisms of the operating system windows 7 with the exploit eternalblue and backdoor doublepulsar. Assoc. Sci. Appl. Res. **14**, 34 (2018)
7. Chen, X.F., Yu, S.Z.: Cipa: a collaborative intrusion prevention architecture for programmable network and sdn. Comput. Secur. **58**, 1–19 (2016). https://doi.org/10.1016/j.cose.2015.11.008

8. Cordero CG, Vasilomanolakis E, Milanov N, et al (2015) Id2t: A diy dataset creation toolkit for intrusion detection systems. In: 2015 IEEE conference on communications and network security (CNS), IEEE, pp 739–740, 10.1109/CNS.2015.7346912
9. Dao TN, Hoang VP, Ta CH, et al (2021) Development of lightweight and accurate intrusion detection on programmable data plane. In: 2021 international conference on advanced technologies for communications (ATC), IEEE, pp 99–103
10. Donovan, A.A., Kernighan, B.W.: The Go programming language. Addison-Wesley Professional (2015)
11. Falliere, N.: Sality: Story of a Peer-to-peer Viral Network, p. 32. Symantec Corporation, Rapport technique (2011)
12. Fawcett, L., Scott-Hayward, S., Broadbent, M., et al.: Tennison: a distributed sdn framework for scalable network security. IEEE J. Sel. Areas Commun. **36**(12), 2805–2818 (2018). https://doi.org/10.1109/JSAC.2018.2871313
13. Foundation ON (2020) P416 language specification. https://p4.org/p4-spec/docs/P4-16-v1.2.1.html
14. Gray N, Dietz K, Seufert M, et al (2021) High performance network metadata extraction using p4 for ml-based intrusion detection systems. In: 2021 IEEE 22nd international conference on high performance switching and routing (HPSR), IEEE, pp 1–7, 10.1109/HPSR52026.2021.9481849
15. Halfond WG, Viegas J, Orso A, et al (2006) A classification of sql-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, IEEE, pp 13–15
16. Harkous H, Jarschel M, He M, et al (2019) Towards understanding the performance of p4 programmable hardware. In: 2019 ACM/IEEE symposium on architectures for networking and communications systems (ANCS), IEEE, pp 1–6
17. Ibanez S, Brebner G, McKeown N, et al (2019) The p4 netfpga workflow for line-rate packet processing. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp 1–9, 10.1145/3289602.3293924
18. Intel Corporation (2021) Intel tofino series programmable ethernet switch asic. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html
19. Jazi, H.H., Gonzalez, H., Stakhanova, N., et al.: Detecting http-based application layer dos attacks on web servers in the presence of sampling. Comput. Netw. **121**, 25–36 (2017). https://doi.org/10.1016/j.comnet.2017.03.018
20. Klöti R, Kotronis V, Smith P (2013) Openflow: a security analysis. In: 2013 21st IEEE international conference on network protocols (ICNP), IEEE, pp 1–6, 10.1109/ICNP.2013.6733671
21. Lantz B, Heller B, McKeown N (2010) A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. Association for Computing Machinery, New York, NY, USA, Hotnets-IX, 10.1145/1868447.1868466
22. Lara, A., Ramamurthy, B.: Opensec: policy-based security using software-defined networking. IEEE Trans. Netw. serv. Manag. **13**(1), 30–42 (2016). https://doi.org/10.1109/GLOCOM.2014.7036903
23. Lyon, G.F.: Nmap network scanning: the official Nmap project guide to network discovery and security scanning. Insecure. Com LLC (US) (2008). https://doi.org/10.5555/1538595
24. Macıas SG, Botero JF (2019) Performance evaluation of the onos controller under an ddos attack. In: 9th Latin American network operations and management symposium (LANOMS 2019)
25. McKeown, N., Anderson, T., Balakrishnan, H., et al.: OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Comput. Commun. Rev. **38**(2), 69–74 (2008). https://doi.org/10.1145/1355734.1355746
26. Mohurle, S., Patil, M.: A brief study of wannacry threat: ransomware attack 2017. Int. J. Adv. Res. Comput. Sci. **8**(5), 1938–1940 (2017)
27. Nascimento MR, Rothenberg CE, Salvador MR, et al (2011) Virtual routers as a service: the routeflow approach leveraging software-defined networks. In: Proceedings of the 6th International Conference on Future Internet Technologies. Association for Computing Machinery, New York, NY, USA, CFI '11, p 34-37, 10.1145/2002396.2002405
28. Ndonda GK, Sadre R (2018) A two-level intrusion detection system for industrial control system networks using p4. In: 5th International Symposium for ICS & SCADA Cyber Security Research 2018 5, pp 31–40, 10.14236/ewic/ICS2018.4
29. Open Networking Foundation (2016a) Behavioural model version 2. https://github.com/p4lang/behavioral-model
30. Open Networking Foundation (2016b) Performance of BMV2. https://github.com/p4lang/behavioral-model/blob/main/docs/performance.md

31. Open Networking Foundation (2022) Basic.p4. https://github.com/opennetworkinglab/onos/blob/master/pipelines/basic/src/main/resources/basic.p4
32. P4 Language Consortium (2017) P4Runtime GitHub Repository. https://github.com/p4lang/PI
33. Park T, Xu Z, Shin S (2018) Hex switch: Hardware-assisted security extensions of openflow. In: Proceedings of the 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges. Association for Computing Machinery, New York, NY, USA, SecSoN '18, p 33-39, 10.1145/3229616.3229622
34. Proofpoint Inc (2021) Emerging threats documentation. https://doc.emergingthreats.net/
35. Sharafaldin I, Lashkari AH, Ghorbani AA (2018) Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSp, pp 108–116, 10.5220/0006639801080116
36. Shorey T, Subbaiah D, Goyal A, et al (2018) Performance comparison and analysis of slowloris, goldeneye and xerxes ddos attack tools. In: 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, pp 318–322, 10.1109/ICACCI.2018.8554590
37. Tavares K, Ferreto T (2021) P4-onids: A p4-based nids optimized for constrained programmable data planes in sdn. In: Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuidos, SBC, pp 434–447, 10.5753/sbrc.2021.16738
38. The Open Information Security Foundation (2022) https://suricata.io/
39. Tian, C., Munir, A., Liu, A.X., et al.: Openfunction: an extensible data plane abstraction protocol for platform-independent software-defined middleboxes. IEEE/ACM Trans. Netw. **26**(3), 1488–1501 (2018). https://doi.org/10.1109/TNET.2018.2829882
40. Turner A (2005) Tcpreplay: Pcap editing and replay tools for * nix. http://tcpreplay.synfin.net/
41. Xing T, Huang D, Xu L, et al (2013) Snortflow: a openflow-based intrusion prevention system in cloud environment. In: 2013 Second GENI Research and Educational Experiment Workshop, IEEE, pp 89–92, 10.1109/GREE.2013.25
42. Zhang K, Zhuo D, Krishnamurthy A (2020) Gallium: automated software middlebox offloading to programmable switches. In: Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, pp 283–295,10.1145/3387514.3405869

Benjamin Lewis is a PhD Student within the School of Computing and Communications at Lancaster University. His research interests revolve around network programmability and dataplane offloading and their application to network security.

Matthew Broadbent's research interests are broadly focused around the intersections of computer networking, security and multimedia. The research he has conducted in these areas has focused on the application of software-defined networks and flexible infrastructures; an emerging research topic that has garnered much attention from both academia and industry.

Charalampos Rotsos is a senior lecturer at Lancaster University. His research interests include service management and orchestration, network programmability, and cloud operating systems.

Nicholas Race is Professor of Networked Systems at Lancaster University. His research focuses on developing future networking services built upon Software Defined Networks and Network Functions Virtualisation. This includes new techniques to enhance the Quality of Experience of media streaming and support for the detection and remediation of network anomalies. He leads the EPSRC Prosperity Partnership "Next-Generation Converged Digital Infrastructure" (NG-CDI) with BT, developing a future network that is "autonomic", with the capability to react and reconfigure infrastructure accordingly with minimal human intervention. He is also the lead at Lancaster of the EPSRC Prosperity Partnership "Future Personalised Object-Based Media Experiences Delivered at Scale Anywhere" with the BBC, which is building an intelligent network compute platform enabling the efficient utilisation of network compute and delivery resources at scale.