NATIONAL TECHNICAL UNIVERSITY OF ATHENS

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMMUNICATION, ELECTRONIC AND INFORMATION ENGINEERING

NETWORK MANAGEMENT AND OPTIMAL DESIGN LABORATORY

Automated Monitoring and Security Services in Federated Software-defined Network Infrastructures

Doctoral Dissertation

of

Adam I. Pavlidis

Supervisor: Vasilis Maglaris, Professor Emeritus, NTUA

Athens, June 2020

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Επικοινωνιων, Ηλεκτρονικης και Συστηματων Πληροφορικης

Εργαστηριο Διαχειρισης και Βελτιστου Σχεδιασμου Δικτυων Τηλεματικης

Αυτοματοποιημένες Διαδικτυακές Υπηρεσίες Ασφάλειας και Συλλογής Δεδομένων σε Προγραμματιζόμενες Ομόσπονδες Υποδομές

Διδακτορική Διατριβή

του

Αδάμ Ι. Παυλίδη

Συμβουλευτική Επιτροπή: Βασίλειος Μάγκλαρης, Ομότιμος Καθηγητής ΕΜΠ (επιβλέπων)

Συμεών Παπαβασιλείου, Καθηγητής ΕΜΠ

Ευστάθιος Συκάς, Καθηγητής ΕΜΠ

………………………..	………………………..	………………………..
Ευστάθιος Συκάς Καθηγητής, ΕΜΠ	Συμεών Παπαβασιλείου Καθηγητής, ΕΜΠ	Νεκτάριος Κοζύρης Καθηγητής, ΕΜΠ
………………………..	………………………..	………………………..
Δημήτριος Σούντρης Καθηγητής, ΕΜΠ	John Baras Professor, UMD	Στυλιανός Σαρτζετάκης Ερευνητής ‘Α, ΙΠΣΥ
	………………………..	
	Δημήτριος Καλογεράς Ερευνητής ‘Β, ΕΠΙΣΕΥ	

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 25^{η} Ιουνίου 2020.

Αθήνα, Ιούνιος 2020

....

Αδάμ Ι. Παυλίδης
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Abstract

This dissertation explores technological advances for network programmability and softwarization to implement automated services for network monitoring and security. Its main focus are software-defined schemas pertaining to data collection, anomaly detection and (collaborative) mitigation of large-scale cyber-attacks.

Initially, we introduce a monitoring architecture for the collection and processing of network monitoring data exported from dispersed vantage points, i.e. agents within devices. These measurements are used to create centralized and localized monitoring views that enhance visibility into anomalous events. Typically, such processing techniques perform well, but rely on traditional protocols for data extraction. In contrast, data plane programmability presents a promising alternative for rapid data processing and anomaly detection. To that end, the P4 Domain Specific Language is investigated to offload related workloads directly within network hardware. Specifically, we propose an in-network DDoS anomaly detection schema that combines important metrics (flows, packet symmetry) typically associated with malicious traffic. These metrics are maintained per protected subnets and evaluated within time-based epochs to generate alarms for external mitigation systems.

In addition to anomaly detection, this dissertation also explores solutions for attack mitigation. As a first step, we propose a framework that distributes filtering rules for multi-vector anomalies to devices across an attack path, enhancing their mitigation potential. Specifically, this is modeled as a combinatorial optimization problem that assigns source-based mitigation actions to devices, considering operator policies for specific attacks and hardware constraints. An important aspect of this work is the automated distribution of rules to heterogeneous multi-vendor environments. To that end, popular techniques for network automation are investigated to seamlessly translate and distribute generic directives to device-specific instructions.

Subsequently, the proposed approach is extended to multi-domain scenarios by establishing trusted federations among network providers for collaborative DDoS mitigation. This approach attempts to preserve on-premise resources and prevent saturation of important links by mitigating malicious traffic earlier in the attack path. Our mitigation schema incorporates blockchain-based smart contracts for signaling, coordination and orchestration purposes. Similarly to our earlier efforts, filtering rules for malicious sources are appropriately assigned to federated partners, factoring in the importance of each flow and the reliability of a potential mitigator.

Source-based approaches may raise issues primarily in terms of scalability and effectiveness. As an alternative, recent technological advances may be used to create customized and agile solutions that employ IP-agnostic traffic characteristics for DDoS defense. To that end, this dissertation considers a two-level schema for anomaly detection and mitigation. The first level incorporates our P4 approach as a coarse-grained DDoS detection mechanism; triggered alarms are used to identify the suspected attack vector (protocol, port). Accordingly, a second protection level is instantiated tailored to the identified attack vector. Data related to the attack are collected in a fine-grained manner via high performance programmable XDP middleboxes. The collected data are fed to a supervised Machine Learning algorithm, that classifies packets as malicious or benign. Features corresponding to malicious packets are used to create unique signatures, employed for filtering purposes. This approach relies on distinct packet characteristics of malicious traffic and not frequently spoofed source IPs.

The proposed mechanisms are evaluated under realistic scenarios in modern experimental setups comprised of P4/XDP-capable hardware, SDN switches, virtual machines and physical servers, using real network data and synthesized traffic traces.

Keywords

DDoS attacks, Anomaly Detection, Attack Mitigation, Software-Defined Networking (SDN), Network Automation, Data Plane Programmability, P4, XDP, Interdomain Federation, Blockchain, Smart Contracts

Περίληψη

Η παρούσα διδακτορική διατριβή μελετά μεθόδους συλλογής δεδομένων, ανίχνευσης και (συνεργατικής) αντιμετώπισης κυβερνοεπιθέσεων μεγάλης κλίμακας και ιδίως κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσίας. Ιδιαίτερη έμφαση δίνεται σε τεχνολογίες για τον προγραμματισμό των δικτυακών συσκευών καθώς και τον έλεγχο τους μέσω λογισμικού, με στόχο τη δημιουργία αυτοματοποιημένων υπηρεσιών ασφάλειας και συλλογής δεδομένων.

Αρχικά, προτείνεται μια αρχιτεκτονική για τη συλλογή δεδομένων από κατανεμημένα σημεία εποπτείας – δικτυακές συσκευές – και την μετέπειτα επεξεργασίας τους. Η ανάλυση αυτών γίνεται με απώτερο σκοπό την αυξημένη ικανότητα ανίχνευσης κεντρικών ή τοπικά εστιασμένων δικτυακών ανωμαλιών. Σε δεύτερο χρόνο μελετάται το P4, μια γλώσσα ειδικού σκοπού για το ριζικό προγραμματισμό των δικτυακών συσκευών. Η μελέτη εστιάζει στη μεταφορά μηχανισμών ανίχνευσης επιθέσεων απευθείας στο υλικό όπου και θα εκτελούνται κατανεμημένα, με στόχο την ταχύτερη επεξεργασία δεδομένων. Κατ' επέκταση, προτείνεται ένας αλγόριθμος ανίχνευσης που συνδυάζει συνήθεις μετρικές (μοναδικές ροές – flows, συμμετρία κίνησης) για ανίχνευση επιθέσεων DDoS. Αυτές οι μετρικές διατηρούνται ανά προστατευόμενο πόρο (κόμβο, υποδίκτυο) σε διάφορα επίπεδα ευκρίνειας και αξιολογούνται ανά τακτά διαστήματα για την έγκαιρη αποστολή μηνυμάτων σε εξωτερικά συστήματα αντιμετώπισης επιθέσεων.

Μετά την ανίχνευση των επιθέσεων είναι αναγκαία η αποτελεσματική αντιμετώπιση τους. Αρχικά, ο επόμενος θεματικός άξονας της διατριβής εισάγει έναν μηχανισμό ανάθεσης κανόνων για την αποκοπή ετερογενών επιθέσεων (multi-vector attacks). Βασικός στόχος είναι η ευέλικτη και συνολικά αποδοτικότερη αντιμετώπιση της κακόβουλης κίνησης σε διάφορες συσκευές κατά μήκος του ίχνους της επίθεσης. Συγκεκριμένα, η ανάθεση μοντελοποιείται σαν ένα συνδυαστικό πρόβλημα βελτιστοποίησης ακέραιου προγραμματισμού, με γνώμονα διαχειριστικές πολιτικές και δυνατότητες των δικτυακών συσκευών. Μια σημαντική πτυχή αυτής της προσέγγισης είναι η αυτοματοποιημένη κατανομή κανόνων σε ετερογενή περιβάλλοντα που απαρτίζονται από συσκευές πολλαπλών κατασκευαστών. Συνεπώς, μελετήθηκαν σύγχρονες τεχνικές δικτυακού αυτοματισμού για τη μετάφραση γενικών κανόνων σε ειδικού τύπου οδηγίες και τη διανομή τους στις αντίστοιχες συσκευές.

Έπειτα, η προσέγγιση αυτή επεκτείνεται σε πολλαπλές διαχειριστικές περιοχές – Αυτόνομα Συστήματα – με τη μορφή ενός ομόσπονδου περιβάλλοντος για παρόχους δικτυακών υπηρεσιών. Βασικός στόχος είναι η προστασία εξωτερικών ζεύξεων καθώς και οικείων, εντός ΑΣ, αμυντικών μηχανισμών. Η αρχιτεκτονική ενσωματώνει έξυπνα ψηφιακά συμβόλαια (smart contracts) αποτυπωμένα σε αλυσιδωτές δομές συναλλαγών (Blockchain) για την σηματοδοσία, τον συντονισμό και την ενορχήστρωση του συνεργατικού μηχανισμού άμυνας. Η ανάθεση κανόνων αποκοπής στους ομόσπονδους εταίρους γίνεται με γνώμονα τη σημασία της κάθε κακόβουλης ροής καθώς και την αξιοπιστία του πιθανού συνεργάτη-εταίρου.

Οι προσεγγίσεις για ανίχνευση και αντιμετώπιση επιθέσεων πιθανώς να αντιμετωπίσουν προβλήματα κλίμακας και επίδοσης, κυρίως λόγω ψευδεπίγραφων διευθύνσεων IP. Με αφορμή αυτό το πρόβλημα, το τελευταίο κομμάτι της διατριβής εστιάζει σε ένα μηχανισμό δυο επιπέδων ο οποίος προσφέρει εξειδικευμένους και κλιμακώσιμους μηχανισμούς αντιμετώπισης επιθέσεων. Το πρώτο επίπεδο βασίζεται στην προγενέστερη προσπάθεια στην γλώσσα P4 και χρησιμοποιείται για την πρωτογενή αναγνώριση του τύπου της επίθεσης και του αμυνόμενου-θύματος. Στην συνέχεια, το δεύτερο επίπεδο προσαρμόζεται στην περίσταση με χρήση προγραμματιζόμενων ενδιάμεσων συσκευών βασισμένες στο περιβάλλον XDP (eXpress Data Path). Δεδομένα που σχετίζονται με την επίθεση συλλέγονται με υψηλή ευκρίνεια και εισάγονται σε ένα μηχανισμό επιβλεπόμενης μηχανικής μάθησης ο οποίος και τα κατηγοριοποιεί ως καλόβουλα ή κακόβουλα. Οι κακόβουλοι συνδυασμοί αποτελούν συνοπτική περιγραφή της επίθεσης και χρησιμοποιούνται για την αποκοπή της. Σημαντικό σημείο διαφοροποίησης είναι πως οι περιγραφές βασίζονται σε εγγενή χαρακτηριστικά της εκάστοτε επίθεσης και όχι σε διευθύνσεις IP.

Οι προτεινόμενοι μηχανισμοί αξιολογήθηκαν κάτω από ρεαλιστικές συνθήκες με χρήση πραγματικών δεδομένων καθώς και συνθετικής κίνησης. Η αξιολόγηση τους έγινε σε σύγχρονες πειραματικές υποδομές βασισμένες σε υλικό συμβατό με το P4 και το XDP, μεταγωγείς SDN, καθώς και υπολογιστικούς κόμβους.

Λέξεις Κλειδιά

Δίκτυα Οριζόμενα από Λογισμικό (SDN), Κατανεμημένες Επιθέσεις Άρνησης Παροχής Υπηρεσίας (DDoS), Αυτοματοποίηση Δικτυακών Λειτουργιών, Προγραμματισμός Επιπέδου Δεδομένων, Ανίχνευση Δικτυακών Συμβάντων και Επιθέσεων, Αντιμετώπιση Επιθέσεων, Συνεργασία Ομόσπονδων Δικτυακών περιοχών, XDP, P4

Acknowledgements

This dissertation is the outcome of a long, continuous and laborious undertaking at the National Technical University of Athens. As this journeys draws to an end, I reflect on the most defining moments; it is needless to say that I would not be where I am, without the help of a great many people. I cannot properly measure their individual contribution on my professional and personal development, however I will do my best to thank most of them.

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Vasilis Maglaris who has been the greatest and most important influence to me, from my undergraduate years to the conclusion of my doctoral studies. As the advisor of my dissertation, he has been an invaluable source of knowledge and experience. I sincerely thank him for believing in me and giving me the opportunity to pursue a PhD, as well as for the time he has generously invested in me.

I would like to thank Dr. Dimitris Kalogeras for his continuous contribution to our academic and professional efforts, as well as all the important lessons that helped me evolve as an individual; the members of my advisory committee, Professor Symeon Papavassiliou and Professor Efstathios Sykas, for their important counsel and directions through this journey; and the members of the examination committee for providing valuable feedback and continuously supporting our research efforts.

In my case, doctoral research was not conducted in seclusion but among colleagues at the NETMODE laboratory that created a pleasant working environment. I wish to all of them good fortune in their current and future endeavors. Among them, my deepest thanks to Dr. Kostas Giotis and Marinos Dimolianis. The former, for his considerable help and guidance in my undergraduate years, as well as for his contribution to our recent work. The latter, for sharing with me countless hours on joint research, professional responsibilities, as well as numerous stimulating discussions and brainstorming sessions. I also thank Dr. Mary Grammatikou for her important support and collaboration during these years; Dimitris Pantazatos and Nikos Kostopoulos for their valuable collaboration in our common obligations; Giannis Sotiropoulos, Kostantinos Mitropoulos and all undergraduate students with whom we have worked together as part of their undergraduate diploma thesis. A sincere thank you to all

NETMODE alumni that paved the way for us and especially Dr. Christos Argyropoulos for encouraging me onwards on various occasions during this journey.

I would like to thank Loukas Anagnostou, Dr. Theocharis Tsigkritis and Dr. Ilias Kotinas for their collaboration in our joint research. Also, my deepest thanks to all the members of the NTUA Network Operations Center, especially Spiros Papageorgiou and Thanasis Douitsis, for freely sharing their considerable experience and knowledge in various contexts, as well as providing important feedback.

A heartfelt thank you is due to all family members and friends, old & new, for all the moments of fun we shared and much needed support that helped me mentally. Among them, I owe a special thank you to my partner Olga, who has provided great comfort and a calming clarity in the most difficult times.

Last but not least, I thank my parents for their unwavering support and selfless love. Their lessons were, and still are, a moral compass for the mountainous terrain that is, life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Symbol	Description
ACL	Access Control List
API	Application Programming Interface
BGP	Border Gateway Protocol
CAPEX	Capital Expenditures
CDN	Content Delivery Networks
CnC	Command and Control
COTS	Commercial Off The Shelf
DDoS	Distributed Denial of Service attacks
DPDK	Data Plane Development Kit
DSL	Domain Specific Language
EMS	Element Management System
FPGA	Field Programmable Gate Array
gNMI	gRPC Network Management Interface
gRPC	Google Remote Procedure Call
HTTP	Hyper Text Transfer Protocol
HW	Hardware
ICT	Information and Communication Technology
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
INT	Inband Network Telemetry
IPS	Intrusion Prevention System
ISP	Internet Service Providers
KPI	Key Performance Indicators
MANO	Management and Orchestration

NAPALM	Network Automation and Programmability Abstraction Layer with Multivendor Support
NETCONF	Network Configuration Protocol
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
OF	OpenFlow Protocol
ONAP	Open Network Automation Project
OPEX	Operational Expenditures
OPNFV	Open Platform for Network Functions Virtualization
OSM	Open Source MANO
OSS	Operations Support System
OVS	Open vSwitch
OVSDB	Open vSwitch Database
P2P	Peer to Peer
P4	Programming Protocol-independent Packet Processors
REST	Representational state transfer
RPC	Remote Procedure Calls
SDN	Software-Defined Networking
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SW	Software
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VIM	Virtualized Infrastructure Manager
VNF	Virtual Network Function
XDP	Express Data Path
YAML	YAML Ain't Markup Language – Yet Another Markup Language
YANG	Yet Another Next Generation

1 Introduction

Rapidly evolving business needs are continuously reshaping ICT environments and related services in terms of elasticity, mobility, programmability and automation. Simultaneously, operators place considerable emphasis on high-speed data collection, processing and analytics especially considering devastating cyber-attacks. These requirements raise new challenges for large-scale distributed (and/or federated) environments such as cloud facilities, data centers, internet exchanges and service provider networks.

The Software-Defined Networking (SDN) and Network Function Virtualization (NFV) architectural paradigms attempt to address modern needs and requirements, offering deep network programmability, disassociation of software (SW) from hardware (HW) and full-fledged automation. This paradigm shift has extensively influenced researchers, device manufacturers and network professionals, establishing a cutting-edge ecosystem mainly centered on open and standardized solutions. Related efforts are spearheaded by technology powerhouses [1]–[4], (ii) innovative start-ups [5]–[7] and (iii) academic/industrial R&D consortiums [8], [9].

OpenFlow (OF) [9], one of the first SDN implementations, advocated for the separation of data and control plane. Devices direct traffic based on forwarding rules, inserted in tables via the OF unified control plane. Perhaps the most radical approach for network programmability is Programming Protocol-independent Packet Processors (P4) [5], a Domain-Specific Language (DSL) that allows developers and operators to flexibly define the processing pipeline of a device. Another approach, *whiteboxing* enables network gear to load various Network Operating Systems (NOS) (e.g. [3]), typically via a standardized boot-loader. However, potential for programmability might vary depending on the capabilities and APIs of each NOS.

Advances in network softwarization enabled NFV solutions that migrate functionality to virtualized resources. These Virtualized Network Functions (VNFs) may be combined with selected physical elements to deliver network services emphasizing on portability, reusability and slicing (separate a resource in distinct isolated chunks). Virtualized appliances are mostly implemented on Linux-based systems that often underperform in terms of packet processing. To that end, various solutions have been considered to accelerate packet processing such as the recently introduced eXpress Data Path framework (XDP) [4]. XDP is a high performance softwarized data plane based on Linux that can be seamlessly ported between machines as a special-purpose (e.g. data collection) middlebox.

SDN and NFV technologies may be used to address cyber-attacks, one of the most prominent threats for modern environments. Cyber-attacks continuously evolve in terms of sophistication and impact, pervasively affecting internetworked infrastructures. Notably, network domains are constantly plagued by massive Distributed Denial of Service (DDoS) attacks launched via infected hosts under the control of malicious actors. DDoS attacks are being offered as a paid commodity service referred to as *Booters* or *Stressers* under the guise of legitimate benchmark solutions. In response, most commercial service providers (e.g. ISPs, CDNs) offer security solutions ranging from monitoring/alerting to DDoS mitigation and full-fledged traffic scrubbing [10].

Important challenges and considerations for cyber-attacks are:

- Efficient Monitoring and data processing: Anomaly detection algorithms frequently rely on exported packet samples and flow records, sent to external systems for further processing. Related monitoring mechanisms and processing frameworks have to keep up with high-speed traffic rates without compromising on visibility and accuracy.
- Utilization of mitigation resources: Effective mitigation of massive heterogeneous attacks requires appropriate utilization of all available systems (e.g. routers, switches, servers). Massive attacks often exceed the capacity of a network organization threatening intermediate systems and network links. The most common solution to this problem is Remotely-Triggered Blackhole Routing (or Blackholing for brevity) that essentially renders the destination unreachable. Alternatively, neighboring or disjoint domains in the attack path may collectively assist in the mitigation process. However, multi-domain collaboration requires among others, appropriate incentives and communication mechanisms between partners.
- Shortcomings of existing solutions: On-premise appliances are based on costly and proprietary solutions that allow moderate, if any, flexibility. Cloud-based alternatives may additionally introduce significant latency and raise privacy concerns.

Inspired by these challenges and enabling technologies mentioned above, this dissertation emphasizes on programmable, automated and performant mechanisms to monitor network traffic, detect and (collaboratively) mitigate cyber-attacks.

Initially, a monitoring architecture is proposed in **section 3** that offers on-demand network monitoring data and related analytics to users (tenants and administrators) within shared network infrastructures. This schema considers different devices as advantageous observation (vantage) points to increase network visibility, an approach well-suited for anomaly detection schemas. Specifically, measurements are collected from scattered monitoring agents and directed to a data pipeline for processing and enrichment.

Typically, such processing techniques perform well and enable powerful analytics but rely on packet samples and flow records, exported to separate systems (i.e. collectors) for further processing. In contrast, data plane programmability is a promising technology that enables rapid control loops for the detection and mitigation of cyber-attacks. This approach aims to be one step ahead of long tested legacy approaches that rely on monitoring data exported from network devices and similar SDN solutions that piggyback on control plane messages [11], [12].

Thus, as a next step we designed and implemented a DDoS detection schema entirely in P4-enabled devices (**Section 4**). This in-network approach offers rapid attack detection, while enabling control plane triggers (i.e. alarms) to external mitigation systems. We employ important traffic features typically associated with anomalous events to increase accuracy while conforming to processing requirements. Specifically, we: (i) inspect network traffic and compute important traffic metrics per network subnet (i.e. network flows and packet symmetry), (ii) evaluate feature values to identify potential threats and (iii) convey alarms to external systems. This workflow can be deployed at various vantage points within a network architecture where each P4 device operates independently in a distributed fashion, considering different levels of granularity (e.g. subnets / hosts).

Monitoring and anomaly detection mechanisms are the first step towards defending against DDoS attacks. Subsequently, this dissertation also investigates automated techniques to mitigate network attacks (**Section 5**). Most SDN techniques and legacy solutions rely on formulating and distributing Access Control Rules (e.g. OpenFlow rules and ACL entries). Consequentially, a common issue is the shortage of mitigation resources and the appropriate placement of rules depending on the attack vector.

To that end, a framework is proposed for mitigating detected anomalies across a network topology. Generic mitigation actions are assigned to devices along an attack path, depending on their capabilities. These devices are organized into distinct stages and network operators express their defense preferences (i.e. mitigation policies) for specific attack types. The assignment of rules to defense stages is formulated as a Generalized Assignment Problem. *Items* (generic mitigation actions) are assigned to *bins* (defense stages) based on capacity *constraints* and *reward* values derived by operator policies. Due to the complexity of GAP, we consider to reduce the input size in order to achieve assignment in a timely manner. This may be accomplished by organizing actions into groups and aggregating malicious IP sources into prefixes [13]. After assignment, generic actions are translated to device-specific access control rules and seamlessly distributed using various southbound interfaces.

An important aspect of our work is DDoS mitigation in a device-agnostic manner. This was initially implemented by the *Ryu* SDN controller employing BGP, OpenFlow and SSH. In addition, a popular open source automation framework, *Salt* [14], was investigated. *Salt* adheres to the fundamental principles of SDN and provides a mature event-driven platform to manage network elements in a programmatic manner. The desired network state (i.e. mitigation actions to be deployed) can be enforced either with specific control plane protocols (e.g. BGP) or via configuration files each appropriately rendered for a specific device (i.e. vendor and operating system).

Massive DDoS attacks consist of many different sources and may overwhelm on-premise resources and saturate important external links. In both cases it is sensible to counter such attacks in their infancy via provider collaborations deploying distributed security mechanisms across multiple domains in an attack path. Thus, in **section 6**, we investigated the establishment of trusted federations among adjacent and disjoint network domains, i.e. autonomous systems (ASes), that collectively mitigate malicious traffic. Our approach focuses on signaling, coordination, and orchestration of a collaborative mitigation schema, facilitated by appropriate blockchain-based smart contracts. Reputation scores are used to rank ASes based on their mitigation track record. Similarly to our previous efforts, we model the allocation of defense resources across multiple collaborators as a combinatorial optimization problem (GAP) incorporating reputation scores and network flow weights. Each collaborator is able to employ any available mitigation mechanism; we employed source-based filtering appliances within the XDP framework.

Maintaining network data and filtering traffic for each malicious source may raise issues primarily in terms of scalability (i.e. volume of source IPs) and effectiveness (e.g. spoofed IPs). As an alternative, recent advances in data plane programmability enable customized solutions tailored to specific attack vectors.

Thus, in **section 7** a two-level schema for traffic classification and attack mitigation is investigated. The first level incorporates our P4-based approach as a rapid yet coarse-grained mechanism to identify network anomalies and the attack vector used. Subsequently, a refined second level schema is instantiated on-demand, tailored to the identified attack. This level is based on high performance XDP middleboxes that extract monitoring data and filter malicious traffic. In contrast to typically used source-based approaches, we consider an alternative IP-agnostic solution that uses combinations of packet characteristics (signatures) to identify malicious traffic. The underlying assumption is that traffic related to DDoS attacks, especially UDP-based, is characterized by a small number of salient characteristics. As a proof-of-concept, we focus on volumetric DNS attacks and assess: (i) packet processing performance, (ii) classification capabilities (ability to identify benign and malicious traffic) and (iii) number of unique rules generated.

The remainder of this dissertation is structured as follows:

Section 2 initially provides a brief overview with regards to network management, covering: softwarization/virtualization trends, high-performance software data planes, monitoring mechanisms, anomaly detection algorithms and mitigation frameworks.

Sections 3 and 4 address the collection and processing of monitoring data, primarily for anomaly detection. **Section 3** focuses on collecting monitoring data exported from dispersed vantage points to enhance visibility. **Section 4** further extends this approach by distributing monitoring and anomaly detection workloads to programmable data plane devices. **Section 5** introduces our mitigation framework that appropriately assigns and automatically translates generic actions to existing on-premise devices. **Section 6** extends this approach to offer DDoS mitigation via network provider collaborations. **Section 7** presents a modular two-level schema employing P4 and XDP that offers traffic classification and attack mitigation tailored to an attack vector. In addition, an IP-agnostic approach is presented for DNS-based volumetric attacks to account for common restrictions introduced by massive numbers of (spoofed) source IPs.

Section 8 summarizes important conclusions and discusses potential future directions. **Sections 9, 10** and **11** contain accordingly related publications, an extended abstract in Greek and references/bibliography.

2 State-of-the-Art: Network Management and Security

2.1 Software-defined Networking and Data Plane Programmability

Network architectures are organized into three distinct planes, namely: (i) **data** or forwarding, (ii) **control** and (iii) **management**. Data plane is related to packet forwarding operations and is often implemented in hardware for increased performance. Control plane refers to signaling operations (e.g. protocols) that affect data plane behavior. Management plane refers to all management operations (e.g. FCAPS – Fault Accounting Configuration Performance Security) that may directly or indirectly influence control and data plane activities.

Network devices, especially legacy ones, may operate across all three planes performing data, control and management plane actions. Internal interactions between data and control plane were, and sometimes still are, implemented in a black-box manner lacking flexibility/programmability. Furthermore, network management operations heavily relied on vendor-specific APIs and proprietary software tools. Large network deployments exhibited vendor lock-in, provided limited options for programmability and flexibility while also requiring considerable effort in configuration and maintenance.

These issues and considerations are addressed by SDN solutions that offer on-demand programmatic reconfiguration of network devices across all three planes (i.e. data, control and management). This is usually accomplished via the adoption of standardized vendor-agnostic APIs and the functional separation of the data plane from the control and management plane. The data plane implements rapid forwarding decisions assisted by hardware specific implementations (e.g. ASICs and TCAM memory banks) and communicates with an external controller that exposes northbound APIs to applications and operators. Perhaps the two most well-known approaches with regards to SDN are the OF (OpenFlow) protocol [9] and the P4 (Programming Protocol-independent Packet Processors) framework [5]. The former defines a unified, vendor-agnostic control plane protocol to communicate with devices and populate appropriate tables with forwarding rules. The latter is a Domain-Specific Language (DSL) that allows developers to define in-depth the processing pipeline of network devices (i.e. data plane). Past efforts, similar to OpenFlow are:

- ForCES [15]: Forwarding and Control Element Separation (ForCES) an approach that disassociate of Forwarding and Control elements. This is achieved using an information model and appropriately defined Logical Function Blocks (LFBs) that describe device capabilities and related network events.
- Ethane [16]: may be viewed as the predecessor of OpenFlow. The proposed architecture considers an omniscient, centralized controller that dictates access control and routing policies to plain forwarding boxes.

Other SDN-related efforts are based on Abstraction Layers and Unified Data Models:

- *Abstraction Layers*: Multi-protocol SDN controllers such as OpenDaylight [17], ONOS [18], Ryu [19] integrate multiple southbound protocols to interface with managed devices. A closely related work, NAPALM [20] maps generic operations to (i) device-specific capabilities (e.g. NETCONF, HTTP API) and (ii) configuration commands generated from appropriate device-specific templates (e.g. Jinja2).
- *Unified Data Models*: the IETF and the OpenConfig [21] informal working group attempt to standardize operations for data retrieval and configuration management employing YANG models.

This dissertation primarily (i) employs OpenFlow and other southbound protocols to disseminate Access Control Rules and (ii) deploys anomaly detection algorithms directly in P4-enabled network elements. Important frameworks used in (i) and (ii) are analyzed in subsections 2.1.1, 2.1.3 (i) and 2.1.2 (ii) accordingly.

2.1.1 OpenFlow Protocol

Depicted in Figure 2.1 is a high level overview of the OF protocol [9] and related key elements, i.e. *Switch* and the *Controller*. The OpenFlow protocol provides a secure control channel via which, a controller interfaces with a switch to retrieve information and deploy forwarding *rules* in a programmatic manner.

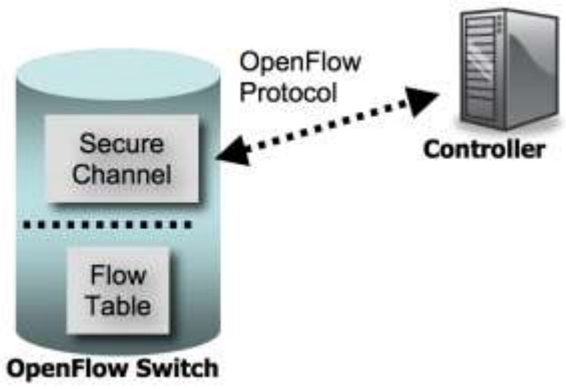


Figure 2.1 OpenFlow protocol: Controller-Switch communication [9]

By design, the OF protocol separates data plane functionality from control/management operations. Forwarding devices operate based on OF rules stored in special purpose data structures (*flow tables*). OF rules consist of Match Fields, Counters and Actions (see Table 2.1):

- **Match Fields (Headers)** to evaluate and match against packets. These include: Ingress port, MAC Addresses, VLAN ID and PCP, IP Addresses, L4 Ports. Later version of OpenFlow also include additional match fields such as MPLS headers.
- **Counters** (bytes, packets) that are updated for each matching packet.
- **Actions** to be executed on matching packets, e.g. forward via a specific port, drop, send to controller, and normal switch processing.

Match Fields (Headers)	Counters	Actions

Table 2.1 OF Rules

These rules are often deployed in hardware-based TCAM memories that allow line rate processing but are expensive and consume a lot of power. Software-based alternatives might suffer in terms of performance. In cases multiple rules match for a specific packet, ties are resolved using *priority*, a metadata that characterizes an OpenFlow rule.

Since its initial inception and first prototype implementations (OF 1.0), OpenFlow has changed considerably as new features were incorporated i.e. advanced pipeline, additional matching fields and actions. The current version of the OpenFlow specification is 1.5.1 [22]. A high level overview of the processing pipeline is depicted in Figure 2.2.

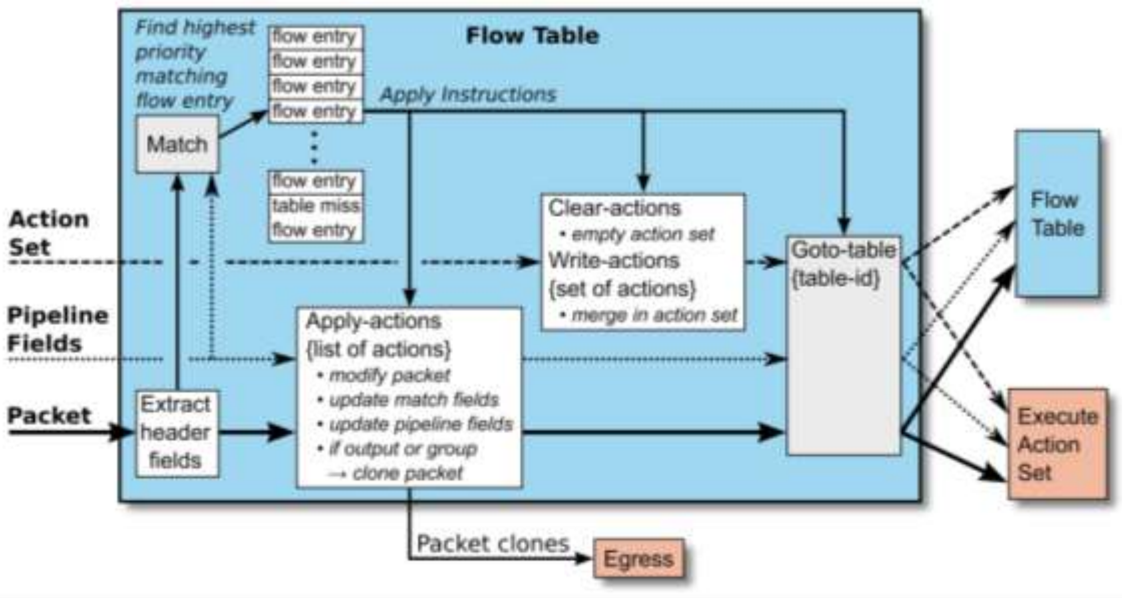


Figure 2.2 OpenFlow Pipeline for version 1.5.1 [22]

The pipeline supports multiple flow tables, that are sequentially numbered. These numbers are used to direct packets to the corresponding table for processing. The processing always starts with the first flow table. Each packet is associated not with one but multiple actions (action set) to be executed on final processing. In case a matching flow entry is found, the action set is updated. Flow rules may direct a packet to a subsequent table for further processing or execute the entire action set. Packets can only be directed to a flow table with a higher number than the current one. The pipeline supports ingress and egress processing; these are logically separated by the first egress table. Tables indexed with a lower number than the first *egress* table are considered *ingress* tables.

Notable features added in recent OpenFlow versions are mentioned below:

- *Multi Controller support*: additional controllers are able to connect and manage the switch via the OF protocol.
- *Groups*: Multiple actions may be grouped together into distinct buckets. Each group is associated with one or more buckets of rules. This approach may be used to implemented load balancing or fast-failover.
- *Meters*: An OF rule may be associated with a meter, comprised of multiple *bands*. A band is characterized by a specific threshold; whenever the traffic rate matching the OF rule exceeds the threshold, the band actions are executed. Typically, these include *dscp marking* and *drop*. Meters are well suited for rate limiting use cases.

Once the most popular implementation of SDN, the OF protocol provides a vendor-agnostic interface to insert rules and directly alter forwarding. OF was employed by Google to perform advanced traffic engineering across its datacenter backbone [23]; Internet2 deployed OF-enabled boxes to create L2 circuits between various points of presence. On a similar use case GÉANT, the pan European Research and Education Network, used OF switches [24] to implement L2 circuits, both point-to-point with reserved bandwidth as well as point-to-multipoint in a VPLS manner. Note that, in an effort to expose additional functionality to the controller, OF implementations grew increasingly more complicated over the years. Currently, interest in OF seems to diminish due to limited adoption and long development cycles, especially after the advent of the more flexible and radically programmable P4 framework [5].

2.1.2 Programming Protocol-independent Packet Processors - P4

A shortcoming of the OpenFlow architecture is its limited flexibility despite exposing various programmable capabilities. In a nutshell, OpenFlow devices are based on "fixed-function" hardware. Operators and experimenters cannot alter the switching pipeline but only utilize the capabilities implemented by the device manufacturer. As such, requested features require a revision of the specification and subsequent implementation by vendors. The need for rapid and radical data plane programmability and reconfiguration has led to the creation of the P4 framework.

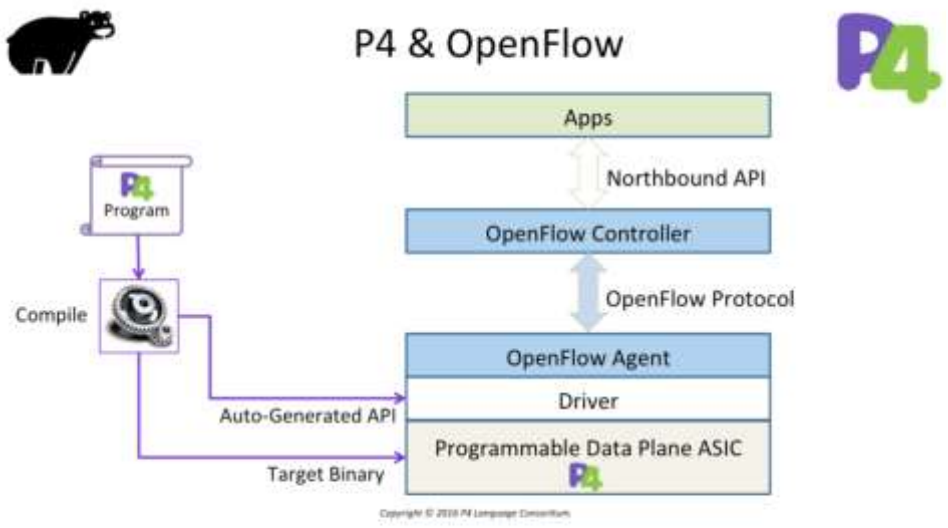


Figure 2.3 SDN: P4 vs OpenFlow, source: P4 Language Consortium[1]

[1] https://p4.org/p4/clarifying-the-differences-between-p4-and-openflow.html

The acronym P4 [5] stands for "Programming Protocol-Independent Packet Processors" and in essence is a high-level Domain Specific Language that defines packet processing functionalities of forwarding devices (*targets* in P4 terminology). P4 is usually perceived as the successor of OpenFlow; although similar in some aspects, these two approaches differ in an important way. P4 defines how a programmable device should process packets (i.e. data plane), whereas OF provides a unified interface (i.e. control plane) to populate forwarding rules in a fixed-function pipeline. These differences are also depicted in Figure 2.3 above.

Important design principles for the P4 language are:

- *Reconfigurability*: Packet parsing and processing may be altered on-demand based on operational requirements.
- *Protocol Independence*: P4 targets are not directly coupled with a specific protocol or pipeline processing. Operations are performed via (a) parsers that extract specific header fields and (b) multiple Match-Action tables containing rules that match against parsed headers.
- *Target-Agnostic*: P4 is independent of the underlying device (target) implementations. Appropriate compilers are used to translate P4 programs to vendor-specific code that is ultimately executed on the device.

Advances in hardware (chip design) have demonstrated that P4 is feasible on super-high speed switching designs. Perhaps the most well-known P4 targets are switches that operate on Tofino chipsets [25] from Barefoot. However, P4 is not limited to a single chip design or vendor; additional targets for P4 are smartNICs [26] and FPGAs [27].

The advent of such advanced data plane programmability coupled with moderate entry barrier in terms of Capital Expenditures (CAPEX) has introduced the concept of in-network computing. This promising paradigm takes advantage of the processing power available in programmable network devices to offload processing and computation tasks. Indicative examples range from monitoring e.g. In-Band Network Telemetry (INT) [28], to consensus building for distributed systems [29] and Map-Reduce algorithms for Big Data processing [30]. Additionally, prominent network use cases are Heavy Hitter [31], [32] and DDoS [33] detection. Although appealing, implementations need to account for memory and processing limitations, as network devices should first and foremost forward traffic. Thus, sacrificing forwarding power to offload non-network computations requires considerable planning and weighing the benefits against the costs.

Currently there are two different releases of the P4 language, P4$_{16}$ [34] and P4$_{14}$ [35]; for each release there are various versions. Within the scope of this thesis we focus on P4$_{16}$, version 1.2.0, and provide a brief overview of workflows, data types, and other important building blocks.

2.1.2.1 P4 Language Architectural Overview

Depicted in Figure 2.4 is a typical workflow for P4 programming. As mentioned, P4 is target independent; this is made possible via compilers provided by device manufacturers and reference (architectural) models. The architecture model is essentially a P4 program that defines (i) which parts of the target are programmable and (ii) additional capabilities offered to developers by the manufacturer.

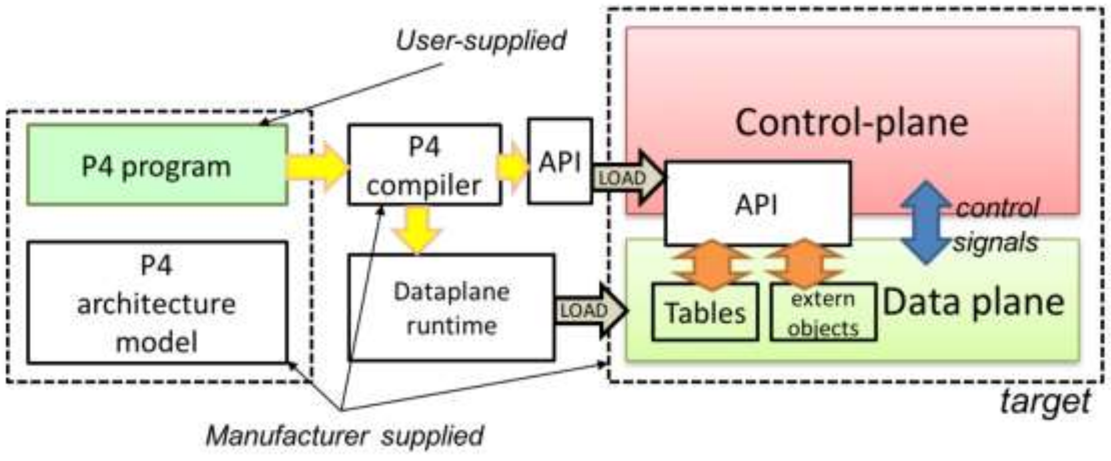


Figure 2.4 P4 Components and Workflow [34]

A popular P4 architecture model, the “v1 model”, is depicted in Figure 2.5. From a high-level standpoint the architecture model is comprised of the following blocks: (1) parser, (2) ingress processing, (3) traffic manager, (4) egress processing and (5) deparser. The (3) traffic manager is not considered programmable and refers to device specific internals. The rest are analyzed in subsection 2.1.2.3 below.

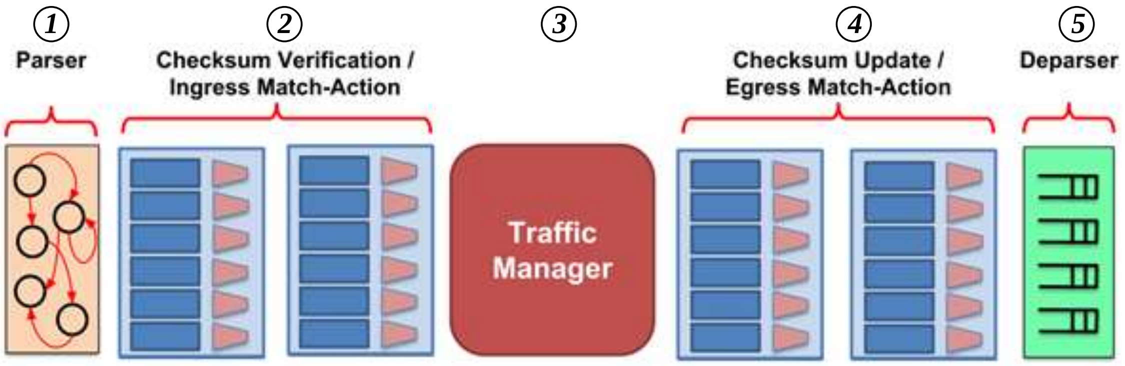
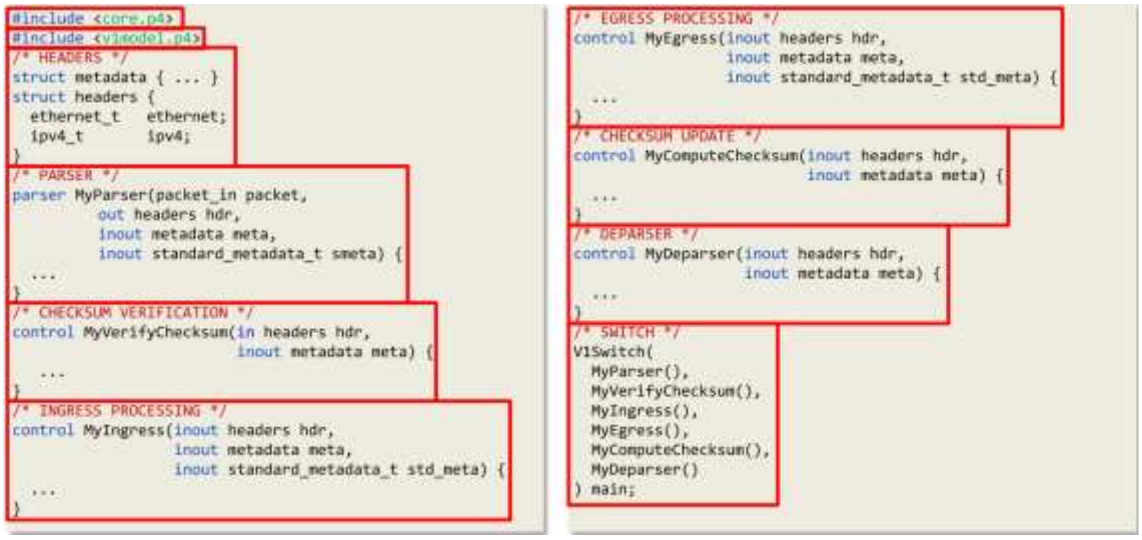

1
Parser
2
Checksum Verification / Ingress Match-Action
3
Traffic Manager
4
Checksum Update / Egress Match-Action
5
Deparser

Figure 2.5 P4 v1 Model architecture [36]

Figure 2.6 depicts a template for P4 program based on the v1 model. Additional libraries can be loaded into a P4 program via the "#include" operator. Note that apart from *v1model.p4*, the standard library of P4 is also included (*core.p4*). This contains error codes and packet_in/packet_out definitions used respectively for incoming and outgoing packets.



```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
  ethernet_t   ethernet;
  ipv4_t       ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
         out headers hdr,
         inout metadata meta,
         inout standard_metadata_t smeta) {
  ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
   ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
  ...
}
```

```
/* EGRESS PROCESSING */
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
  ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
  ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
  ...
}
/* SWITCH */
V1Switch(
  MyParser(),
  MyVerifyChecksum(),
  MyIngress(),
  MyEgress(),
  MyComputeChecksum(),
  MyDeparser()
) main;
```

Figure 2.6 P4 template program for v1 model [36]

2.1.2.2 Standard Types and Metadata

The P4 framework provides standard data types. Notable examples are:

- *int<N>*: signed integer, N bits
- *bit<N>*: unsigned integer, N bits
- *bool*: True, False
- *match_kind*: used in tables to specify the match method for parsed headers:
  - *exact*: parsed header matches exactly
  - *ternary*: parsed header matches using an arbitrary bitmask that supports wildcard matching

- o *lpm*: parsed header matches using the longest most specific compatible match, similar to the "longest prefix match" concept of IP routing.

Types *bit* and *int* support standard arithmetic (apart from division) and logical operations as well as comparisons. Furthermore, these basic types are used to implement a wide variety of use cases ranging from packet header definitions to probabilistic data structures. Finally, we elaborate on two important types *Header* and *Struct*:

- *Header*: Defines the exact representation of packet headers and associated object types for each fields. Note that a given *Header* representation may not be comprised of other *Header* types, i.e. nested header definitions are not supported. This is attributed to complications during the validation process of nested representations. A header is characterized as *valid* or *invalid* employing a hidden "validity" bit. An example is presented in Table 2.2 below.

typedef bit<48> EthernetAddress;
typedef bit<32> IPv4Address;
 	
// Standard Ethernet header
header Ethernet_h {
    EthernetAddress dstAddr;
    EthernetAddress srcAddr;
    bit<16> etherType
}
	// IPv4 header (without options)
	header IPv4_h {
	  bit<4> version;
	  bit<4> ihl;
	  bit<8> diffserv;
	  bit<16> totalLen;
	  bit<16> identification;
	  bit<3> flags;
	  bit<13> fragOffset;
	  bit<8> ttl;
	  bit<8> protocol;
	  bit<16> hdrChecksum;
	  IPv4Address srcAddr;
	  IPv4Address dstAddr;
	}

Table 2.2 P4 Header example [34]

- *Struct*: These type declarations define various schemas, containing even other structs (see Table 2.3). Also *Header* definitions may also contain structs. Apart from custom structs defined by the programmer, reference architectures usually define well-known structures, e.g. *standard_metadata*. Within this dissertation, structs have been used to define metadata headers that accompany packets across a processing pipeline.

struct ipv6_addr {
  bit<32> Addr0;
  bit<32> Addr1;
  bit<32> Addr2;
  bit<32> Addr3;
}
header ipv6_t {
  bit<4> version;
	header Tcp_h { … }
	header Udp_h { … }
	struct Parsed_headers {
	  Ethernet_h ethernet;
	  IPv4_h ipv4;
	  Tcp_h tcp;
	  Udp_h udp;
	}

```
  bit<8>        trafficClass;
  bit<20>       flowLabel;
  bit<16>       payloadLen;
  bit<8>        nextHdr;
  bit<8>        hopLimit;
  ipv6_addr     src;
  ipv6_addr     dst;
}
```

Table 2.3 P4 Struct example, source [34]

2.1.2.3 Programmable control blocks

The first programmable block is typically the *Parser*, essentially a Finite State Machine declaration that defines transitions between distinct states based on packet headers. The initial state is labeled "start". Each state extracts packet header values to be stored in the corresponding P4 header object. Transitions may be based on conditionals (e.g. IPv4, IPv6). The valid final stages are either *accept* or *reject*; accordingly these denote a successful or unsuccessful packet parsing.

Parser definitions contain as parameters (i) the packet, (ii) the struct used to store extracted headers and (iii) custom (meta)data structures. A typical example is presented in Table 2.4 below.

```
parser MyParser(
                packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {
    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

Table 2.4 P4 Parser example

Extracted values from packet headers are used in subsequent *control blocks* that define all operations related to packet processing and forwarding. Typically, the first control block after the *Parser* is the *Ingress*. Furthermore, Match-Action structures, i.e. *tables*, are used within control blocks (e.g. *Ingress*) to assign *actions* (P4 functions) to a packet. A control block declaration also defines all the parameters to be used within its context, e.g. metadata and parsed headers. The exact functionality and number of control blocks may differ depending on the architecture. The final control block is the *Deparser*, whose purpose is to reassemble valid headers for outgoing packets.

As mentioned, hardware (*target*) manufacturers use an architecture reference model to define the processing pipeline supported by their hardware. The building blocks mentioned above are ordered and glued together using the *package* definition. A P4 programmer uses the *package* keyword to map custom control blocks to the placeholders provided by the manufacturer.

2.1.2.4 Actions and Tables

A P4 *action* may be viewed as a function that groups together multiple operations. Match-Action *tables* (see Table 2.5 below) are employed to match packet headers and perform the defined *action*. Important *table* properties may include:

- *key*: comprised of a list of matching scopes, e.g. extracted headers and the *match_kind*, i.e. lpm, ternary, exact.
- *actions*: every available action to be associated with a specific rule.
- *size* (optional): the maximum number of entries for this table
- *default_action* (optional): specifies a default action in case no other entry is matched. This was commonly referred to as "table-miss" entry in OpenFlow.

```
table subnet_src
   key = {
      hdr.ipv4.srcAddr : ternary;
   }
   actions = {
      get_src_subnet;
      NoAction;
   }
   size = 258;
   default_action = NoAction();
}
```

Table 2.5 P4 Table example

2.1.3 Data Modeling and Abstraction Layers

SDN-related efforts are not strictly limited to OF and P4. Network softwarization and heterogeneous management can be also achieved via (i) vendor-agnostic interfaces based on unified data models and (ii) abstraction layers that translate generic operations to device-specific southbound APIs (e.g. BGP, OpenFlow, OVSDB, NETCONF, SSH). The former is mainly represented by the efforts of IETF and OpenConfig [21] to provide widely adopted data models. The latter typically employs multi-protocol SDN controllers [17]–[19], automation frameworks [14], [20], [37] in order to create layers of abstraction.

2.1.3.1 YANG and OpenConfig

Network management relied (and perhaps still does) on the Simple Network Management Protocol (SNMP) [38]. SNMP provides a mechanism to retrieve device data structured into Management Information Bases (MIBs). MIBs are defined using the Structure of Management Information (SMI) [39] language. Similarly, Yet Another Next Generation (YANG) [40] is a data modeling language that describes information (configuration, state data) and related operations (Remote Procedure Calls - RPCs) for managed objects. As in SMI-based MIBs, device vendors may opt to support generic YANG models or define their own. Currently the IETF is heavily involved into standardizing YANG models. Similar efforts are spearheaded by the OpenConfig [21] working group, a consortium comprised primarily from network operators such as Google, Facebook, Cloudflare, AT&T, Deutsche Telekom, Microsoft, Netflix and Comcast. This group focuses on vendor-neutral, widely-adopted models for configuration management. Related efforts are illustrated in Figure 2.7 below.

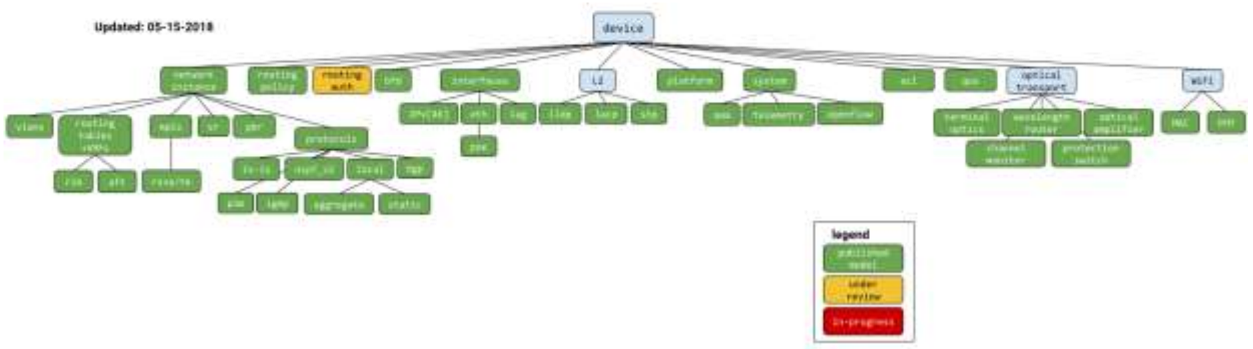


Figure 2.7 OpenConfig Standardization Efforts [21]

Initially, YANG models were conceptually coupled with the NETCONF protocol. However, management schemas that employ IETF and OpenConfig YANG models may use a variety of protocols. Typical examples are:

- *NETCONF*: The Network Configuration Protocol (NETCONF) interfaces with network devices to retrieve information and applies configuration changes. NETCONF operations are implemented as Remote Procedure Calls (RPCs), conveyed typically over SSH, that contain XML encoded data.

- *RESTCONF*: This approach is similar to NETCONF but is based on the Representation State Transfer (REST) paradigm. In short, RESTCONF supports CRUD operations on a hierarchy of YANG-modeled resources.
- *gNMI*: The gRPC Network Management Interface is a mechanism developed by Google to interface with network devices. Similar to NETCONF, gNMI allows operators to retrieve and manipulate the device configuration via YANG models. It is based on gRPC a modern, high performance framework for Remote Procedure Calls, initially developed by Google.

2.1.3.2 Multi-protocol SDN Controllers and Automation Frameworks

The wide adoption of a unified API is a slow and cumbersome process, heavily reliant on collaboration from vendors. Abstraction layers are a more flexible approach that exposes northbound APIs to applications, while seamlessly translating and conveying device-specific instructions.

SDN controllers, though initially supported only OpenFlow, now typically provide such layers of abstraction. New popular implementations such as OpenDaylight [17], ONOS [18] and Ryu [19] follow an architecture that leverages multiple southbound protocols to interface with managed elements. High level functionality is exposed to applications via northbound APIs. The core platform maintains important state, orchestrates data retrieval and polling intervals while "drivers" are used to interface with devices via the appropriate southbound protocol. A typical example for such an architecture is presented in Figure 2.8.

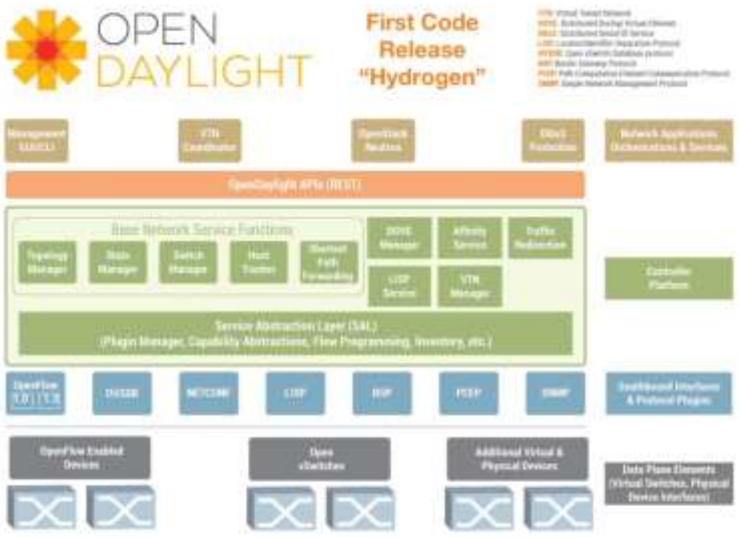

OPEN DAYLIGHT
First Code Release "Hydrogen"
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
Base Network Service Functions
Southbound Interfaces & Protocol Plugins
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 2.8 OpenDaylight architecture [17]

This architecture enables interoperability with other systems, a key feature for modern environments. Note that OpenDaylight is integrated with Openstack [41] a popular cloud management platform.

Automation frameworks provide similar abstractions as SDN controllers, typically employing a specific syntax to declare the desired state and/or orchestrate task executions. Puppet [42], arguably the most popular approach for managing systems, employs a Domain-Specific Language to declare the desired state. Other approaches, Ansible [37] and Salt (Saltstack) [14] have developed significant traction within the network community primarily due to substantial endorsement from vendors and positive feedback from early adopters (e.g. Cloudflare) .

Ansible adopts YAML (YAML Ain't Markup Language), an easy to understand serialization format to define *plays* (i.e. tasks). These are organized into *playbooks* and executed in desired targets (e.g. network devices, servers). Ansible follows a smooth learning curve and has significant vendor support thus making it a good fit for network automation. Typically, Ansible heavily relies on Jinja2 templates to create different configurations in a programmatic manner; templates are in essence predefined configuration stanzas with appropriate placeholders. These placeholders are filled using data from various sources e.g. Databases, Network Management Systems. Ansible playbooks are executed from a centralized host that has access (e.g. SSH) to managed devices.

Salt or *Saltstack* is an automation and configuration management framework built around an event bus. Similarly to Puppet, Salt is based on master-minion architecture. Minions may be perceived as software agents (daemons) deployed within managed systems (e.g. Linux servers). Communication with the master is achieved over a publish-subscribe messaging system. Contrary to Puppet, Salt is primarily Push-based; at the master's behest, minions execute Python modules or enforce a desired state as defined in SaLt State (*SLS*) files. The master bears a close resemblance to the all-seeing, omniscient SDN controllers. However, network devices are usually limited by vendor restrictions and cannot host a Salt minion; thus support is commonly offered via specialized *proxy* minions deployed in general purpose systems (e.g. Linux servers). Proxy minions operate as regular Salt minions and interface with network elements via a device-specific driver.

Perhaps the most well-known example is *NAPALM* [20], an open-source Python library providing high-layer abstractions for device programmability. These abstractions include data retrieval APIs (“getters”) and merge configuration operations (“setters”), based on Jinja2 device–specific templates. Abstract *get* actions and configuration stanzas are conveyed via the appropriate library for each vendor and operating systems. A functional overview of NAPALM is presented in Figure 2.9 below.

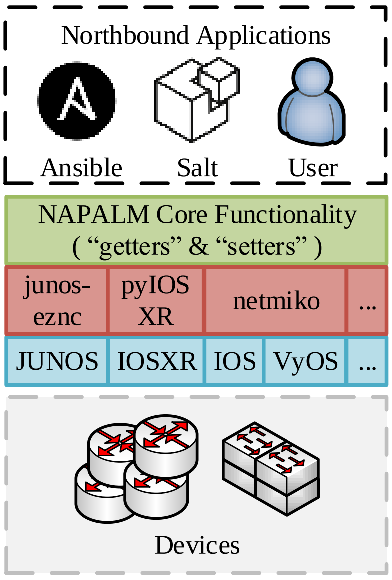


Figure 2.9 NAPALM Architecture

Note that both Ansible and Salt, make heavy use of NAPALM and Jinja2 templates in an effort to seamlessly integrate with heterogeneous network environments. These approaches slightly veer from typical SDN architectures that advocate for cutting-edge programmability and instead translate commands to configuration changes. However, they are extremely popular enabling operators to make the most of their current infrastructure and automate time consuming tasks.

2.2 Network Function Virtualization

Legacy network environments usually implement network functionality via dedicated hardware. Emerging requirements for programmability and agility brought forth a new architectural paradigm shift in the form of Network Function Virtualization (NFV) as defined by ETSI [43]. In a nutshell, functionalities traditionally implemented in hardware appliances are “softwarized” and migrated to Commercial Off-The-Shelf (COTS) hardware, as Virtual Network Functions (VNFs). Important principles of NFV architectures are:

- *Separation*: Network Functionalities are offered as a software-based service disassociated from the underlying hardware. This decoupled approach allows software and hardware to follow separate evolution paths and release cycles.
- *Efficiency*: Hardware can be dynamically repurposed for various purposes amortizing Capital Expenditures (e.g. procurement costs) and Operational Expenditures (e.g. power consumption and cooling).
- *Elasticity*: operators may scale the VNFs as elastic needs manifest. Moreover, VNF deployment can be streamlined and automated via software tools. These benefits are of considerable importance in the current dynamic landscape.

The NFV reference architecture proposed by ETSI is presented in Figure 2.10 below:

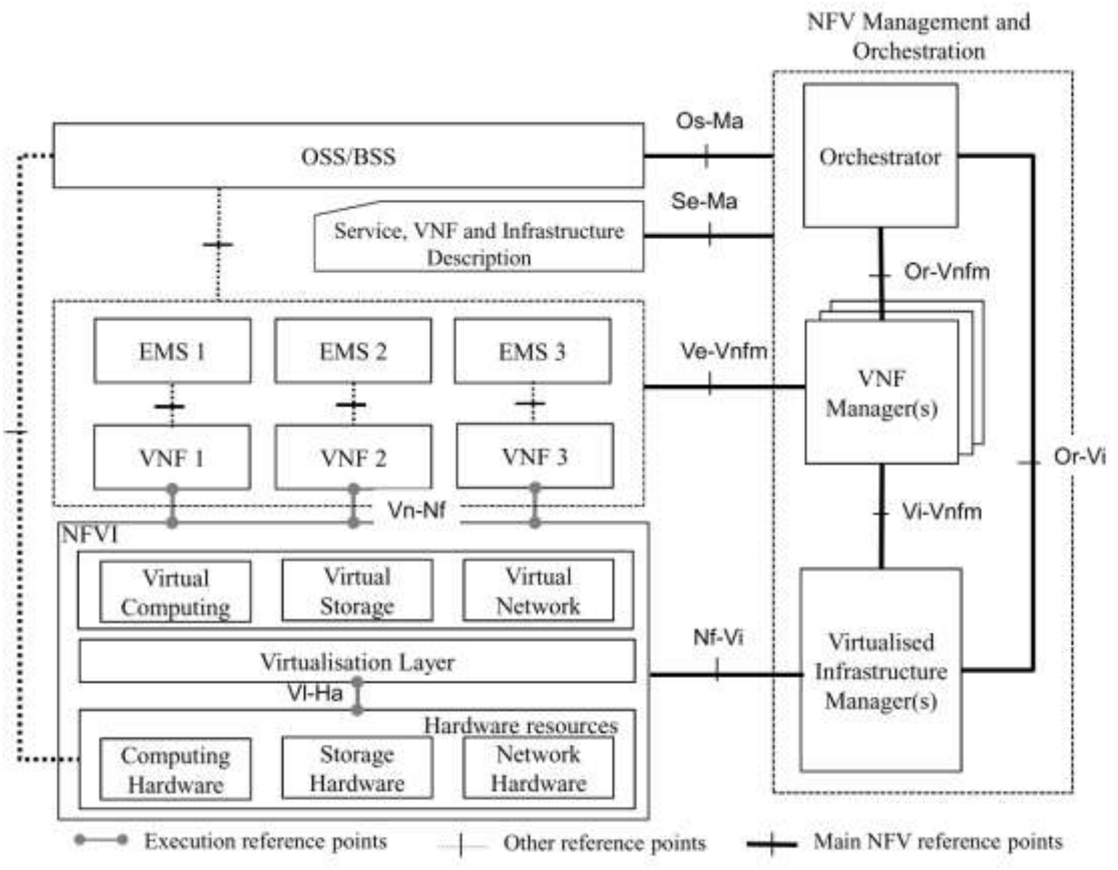


Figure 2.10 NFV reference architecture [43]

Key elements in the reference architecture are:

- *Virtual (or Virtualized) Network Function (VNF)*: a virtualized instance of legacy elements that offers a specific network function or service. This instance may be based on various virtual resource pools and is not strictly limited to a distinct component (VM).
- *Element Management System (EMS)*: manages one or more VNFs.

- *NFV Infrastructure (NFVI)*: refers to the entire deployment and execution environment for VNFs. This is comprised by the hardware substrate, the virtualization layer that provides pools of resources and necessary middleware components/software tools.
- *Virtualized Infrastructure Manager (VIM)*: interfaces with the NFVI to provision and manage the necessary resources (compute, storage network) for a VNF.
- *VNF Manager*: is responsible for deploying, scaling, monitoring and removing one or multiple VNFs.
- *Orchestrator*: Receives requests from other systems (e.g. OSS/BSS), maintains high-level overview and manages/orchestrates the NFVI and deployed VNFs.

Over the years, various organizations pursue NFV-related goals usually based on the reference architecture above or close adaptations. Notable mentions are OPNFV [44], ONAP [45], OSM [46] and CORD [47].

2.3 High Performance Packet Processors

The virtualization and softwarization of network functions introduced various challenges; important among them is performance. Software-based VNFs are usually based on Linux distributions that though more agile have to keep up with monolithic albeit performant hardware implementations. A common issue is the limited packet processing capability of Linux systems [48]. To that end, various approaches have been introduced that enable programmable and fast packet processing. These are mostly implemented using: (a) dedicated kernel modules, (b) kernel bypass techniques, (c) special-purpose systems and (d) programmable hardware devices.

Kernel Modules: Approaches based on customized kernel modules attempt to increase performance by attaching into the existing stack and perform specialized actions. This requires considerable care since potential bugs can severely affect a system. A notable example is the Open vSwitch (OVS) [49], perhaps the most popular virtual switch used in various use cases; additional examples are the virtual router frameworks Contrail [50] and Click [51].

Kernel bypass: High packet rate in general purpose operating systems is achieved via specialized toolkits such as Netmap [52], PF_RING [53] and DPDK [54]. In a nutshell, these tools typically bypass the kernel path and directly control the underlying hardware to avoid time consuming context switches between kernel space and user space. Often such solutions may dedicate CPU cores to poll for new packets, a technique commonly referred to as “BusyPolling”.

Special-purpose Systems: After the advent of NFV solutions, various efforts investigate special-purpose systems as VNFs to offer high performance packet processing. Prominent examples are ClickOS [55] and NetVM [56], based on Xen and KVM hypervisors accordingly. Specifically, ClickOS employs minimal packaged versions of the Click router framework and NetVM leverages on DPDK.

Programmable Hardware Devices: Network hardware can be reprogrammed to achieve high-performance processing. An indicative example is NetFPGA [27], an effort that precedes P4 and allows developers to program packet processing tasks on FPGAs. With the advent of P4, NetFPGA is a popular hardware *target* for P4.

XDP system: A promising alternative to the approaches presented above is the eXpress Data Path (XDP) framework [4], a softwarized data plane that harmonically co-exists with the Linux kernel. XDP actions are executed prior to costly networking stack operations and can be seamlessly ported across Linux machines. This high-performance yet flexible framework has been widely adopted in production network environments for various use cases and applications ranging from routing and load balancing [2] to data collection, DDoS Detection and Mitigation [6].

XDP programs, written in C, are executed either in software within the context of the network driver or offloaded directly in Network Interface Cards (NICs) [26]. Their execution is initiated upon the arrival of packets on the NIC. In turn, packet field values can be parsed, extracted and stored in persistent memory referred to as Berkeley Packet Filter (BPF) Maps. These are key-value stores defined when the XDP program is loaded. After processing, XDP returns an action for each packet which defines how it should be handled. The packets can be either (i) dropped - *XDP_DROP*, (ii) passed to the network stack - *XDP_PASS*, (iii) redirected to another interface - *XDP_REDIRECT* or (iv) transmitted back from the same interface - *XDP_TX*. As in all programmable data planes, the design and implementation of XDP applications require significant attention due to specific limitations. Indicatively only (i) bounded loops, (ii) fixed-size data structures (iii) 4096 BPF instructions per program and (iv) specific kernel functions are supported.

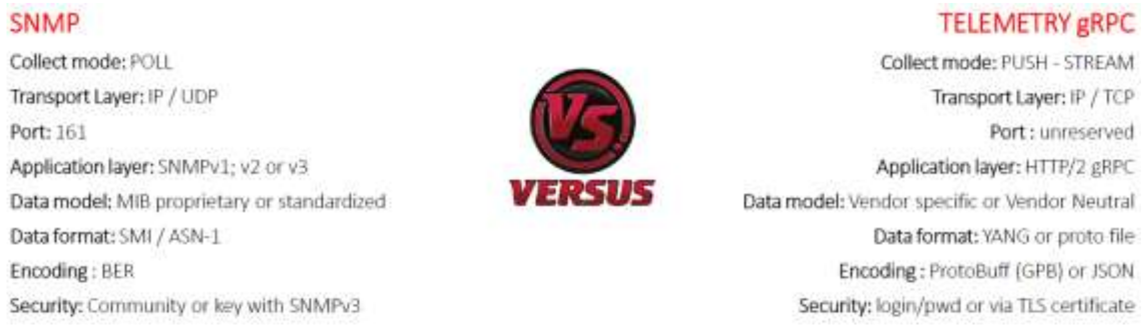
2.4 Monitoring Solutions

Accurate, performant and scalable monitoring solutions are of paramount importance to modern network environments. Rapidly changing traffic patterns and security incidents require in-depth network visibility based on various metrics such as interface counters, queue occupancy, flow records and packet samples.

2.4.1 SNMP and Streaming Telemetry

As mentioned SNMP is frequently employed as a standardized mechanism to poll network devices and collect important data for various management purposes. However, SNMP exhibits scalability limitations and modeling shortcomings inherited from SMI. The former limitations typically force network operators to set longer polling intervals, thus leading to coarse-grained monitoring data (e.g. interface counters). Various alternatives have been considered in that regard. Indicatively, the sFlow [57] protocol, may also send counters (*counter samples*) to the collector in addition to packet samples. Additionally, software mechanisms can be used to efficiently orchestrate the collection of monitoring data. Such a framework is presented in [58] employing a modern information schema and a master/worker architecture for distributing monitoring workloads, i.e. SNMP GET operations.

The state-of-the-art approach for continuously retrieving network measurements from devices is *streaming telemetry*. This approach refers to the act of pushing measurements or other events to appropriate equipment (collectors) for various purposes. Typical examples for time series measurements are interface counters and ingress/egress queue depths. In comparison to SNMP, telemetry introduces a performant Push-based alternative, that promptly sends information streams to collectors alleviating issues related to continuous polling. In Figure 2.11 below, an analogy of important terms between SNMP and telemetry is presented.



SNMP	TELEMETRY gRPC
Collect mode: POLL	Collect mode: PUSH - STREAM
Transport Layer: IP / UDP	Transport Layer: IP / TCP
Port: 161	Port : unreserved
Application layer: SNMPv1; v2 or v3	Application layer: HTTP/2 gRPC
Data model: MIB proprietary or standardized	Data model: Vendor specific or Vendor Neutral
Data format: SMI / ASN-1	Data format: YANG or proto file
Encoding : BER	Encoding : ProtoBuff (GPB) or JSON
Security: Community or key with SNMPv3	Security: login/pwd or via TLS certificate

Figure 2.11 SNMP vs Telemetry[2]

Telemetry has two different initiation mechanisms, Dial-In and Dial-Out. In both approaches, data are streamed to a collector by the network device.

- *Dial-In*: The client (collector) dynamically subscribes into specific information streams offered by the device (i.e. sensor paths).
- *Dial-Out*: The network device is statically configured to publish monitoring data to a specific collector.

Moreover, telemetry may operate using different transport / application protocols, i.e. UDP, TCP and gRPC (via gNMI). Note that, vendors such as Cisco, Juniper and Nokia have made significant progress in integrating related telemetry capabilities in their devices [59], [60]. Arguably the most prominent approach and also supported by all three vendors above, is gNMI/gRPC-based Telemetry operating over HTTP/2 and initiated in a Dial-In fashion.

2.4.2 In-band Network Telemetry

A similar but slightly different approach is *In-band Network Telemetry* ("INT"). INT is a framework that enables the data plane to monitor network services without intervention from the control/management plane. As defined in the INT architectural model [28], packet headers are used to convey "telemetry instructions", embedded within normal traffic or in specialized packets (probes). Initially, instructions are inserted by INT *traffic sources* (e.g. applications, servers, NICs) and subsequently processed by other INT-capable devices. These devices interpret instructions and record monitoring data within INT packets. Finally, INT *traffic sinks* retrieve related data and act appropriately. Monitoring data include but are not limited to *switch id*, *ingress port*, *ingress timestamp*, *egress port*, *queue occupancy*, *queue congestion status*. This

[2] http://junosandme.over-blog.com/2019/02/grpc-telemetry.html

approach enables a thorough End-to-End monitoring of the exact network state as observed by the packet across all intermediate devices. Though the architecture of INT is generic and may be used within different networking environments, the use case fits well into P4 [28], [61], and XDP [62].

2.4.3 Packet-level and Flow-level information

While the aforementioned techniques provide important traffic metrics, detailed data related to L3 (network layer) and above are invaluable. As an example, per-packet and/or per-flow data may be used for traffic engineering, troubleshooting and intrusion detection/prevention. Most production solutions heavily rely on monitoring information of such granularity (L3+) for various network management purposes. Operators typically use protocols such as sFlow [57] or NetFlow [63] to export packet samples or aggregated flow records. Such information is collected and processed by external systems, usually in a centralized fashion. Alternatively, virtual or hardware appliances can monitor the traffic directly if placed at central monitoring hubs or via mirrored traffic streams (e.g. SPAN ports). Various related research efforts in the literature investigate monitoring solutions within the context of SDN, Programmable Data Planes and NFV.

2.4.4 SDN and Data Plane monitoring solutions

OF-based approaches often poll the network devices for statistics [64] or intercept control messages, e.g. *PacketIn* and *FlowRemoved* [12]. With regards to the former, i.e. polling for monitoring information, as exhibited in [65], the management interface (and by extent access to monitoring information) is severely limited in comparison to the switching ASIC. To that end, monitoring data originating from sFlow may be also used to alleviate the controller as well as the device [66]. Other alternatives modify the OF-protocol to enable per-flow sampling on the network device [11] and direct samples to the controller. However, reprogramming OpenFlow devices is no easy task.

P4-based approaches leverage programmable hardware to perform in-network computations. This approach locally processes traffic on network devices to promptly obtain network measurements and identify various events, e.g. Heavy-Hitter detection. Most such approaches leverage on probabilistic data structures (i.e. sketches) to respect the constrained memory and CPU budget [31], [67].

Programmable traffic processors e.g. P4 [5], NETMAP [52], PF_RING(ZC) [53], DPDK [54] and XDP [4], have been actively used for monitoring purposes and in general alleviate the overhead of packet processing in Linux. Indicatively we mention nProbe [68] a solution that may operate as probe (i.e. generator), collector and proxy for NetFlow/IPFIX traffic, typically assisted by PF_RING_ZC kernel bypass module.

2.4.5 Monitoring-as-a-Service: NFV and Cloud Infrastructures

Monitoring, in the context of NFV may refer to: (a) checking the health (important KPIs) of deployed VNFs and (b) dynamically deploying VNFs that extract, collect and process measurements. The two are closely related and somewhat difficult to separate. The former usually involves interfacing with the NFVI and retrieving important metrics from the underlying substrate (e.g. OpenStack) [69]. The latter, typically relies on dynamically placed probes that receive traffic via traffic redirection/mirroring or act as a tap in the wire [70]. Both approaches may employ various mechanisms mentioned in section 2.3 above to extract monitoring information across the data plane of a VNF chain in a performant manner [71]. An important challenge is extracting and isolating user (tenant) data within a multi-tenant cloud/NFV infrastructures [72]. Note that, most major commercial cloud providers empower their tenants (customers) with monitoring metrics and related analytics [73]–[75] while third parties offer application monitoring as Software-as-a-Service (SaaS) [76].

2.5 Cyber Threats

The number of end-user devices and (inter)networked systems in general, increases in an ever-growing rate. Each networked system or device is a potential target for cyber-attacks; threats range from data exfiltration, malware propagation and Denial of Service attacks. Related incidents of data theft [77] and recent regulatory legislation such as GDPR has significantly raised awareness on the matter.

2.5.1 Malicious Software

There are different variations of malicious software such as *viruses*, *worms* and *trojans*. Typically, viruses require executing an infected file obtained via different channels, e.g. mail, file download, physical media; trojan variants usually disguise themselves as legitimate files. Worms exploit operating system or application vulnerabilities to self-propagate and infect different machines to be used for other illicit activities (e.g. DDoS attacks). Ransomware also exploits vulnerabilities to infect a computer and encrypts the system or files. Typically, the actors attempt to extort payment via crypto currencies to provide the user with the necessary decryption method/credentials. A system-centric study of malware is beyond the scope of this dissertation; instead we focus only on the network aspects of cyber threats and especially DDoS attacks.

2.5.2 Botnets

In general, the term *bot* refers to a system that automates various workflows, often interfacing with human end-users or other services. Bots can be benign such as chatbots used for customer support or malicious nodes that send spam e-mails and participate in DDoS attacks. Focusing on the latter, bots or zombie computers are systems that have been infected with malware and are under the control of an external (malicious) actor. This malware variant is usually able to self-propagate, continuously scanning for vulnerabilities and attempting to infect other networked systems.

Many zombies form a botnet and are used for various malicious activities orchestrated by Command and Control (CnC) servers. Instructions are conveyed to infected hosts using various communication patterns such as centralized, hierarchical or Peer-to-Peer (P2P). In general, cyber-security professionals and law enforcement organizations attempt to seize, virtually or physically, CnC servers in order to disrupt communications and prevent malicious activity. Static IP assignment for CnC servers makes a Botnet takedown easy, it is quite common for infected hosts to communicate with frequently changing domain names computed via Domain Generation Algorithms [78]. The CnC domain becomes a moving target, constantly in flux. Thus, even if a CnC is seized a new domain will be computed and the botnet lives on.

2.5.3 Denial-of-Service attacks

Botnets are typically used to conduct Distributed Denial of Service (DDoS) attacks that attempt to severely disrupt a network-based service, congest links and even cause widespread outages. DDoS attacks directly employ dispersed botnet nodes and also exploit vulnerable systems that are prone to abuse. The latter relies on two techniques called reflection and amplification [79], [80]. Reflection allows an attacker to spoof the source IP address of a request, thus responses are reflected back to the spoofed IP address (victim). Amplification techniques exploit vulnerabilities in well-known protocols to send large responses with minimal effort. Note that, almost all reflection and amplification attacks require not only IP spoofing but a connectionless protocol (i.e. UDP) [80]. An interesting exception is TCP-based reflection where misbehaving devices (mostly residential internet routers) send many TCP RST messages [81], in response to a TCP SYN.

DDoS attack vectors vary from high rates attacks, using amplification techniques and/or massive botnets, to sophisticated low rate, even stealthy attacks, that target specific applications. We present the following categories below and briefly provide indicative examples.

Volumetric attacks typically attempt to saturate important network links for the victim (e.g. upstream/peering links) causing congestion. To that end, attackers employ amplification techniques to create massive amounts of traffic. A thorough review of protocols used for amplification is available in [79], [80], [82]. Indicatively, these are: (a) DNS – "ANY" requests, (b) NTP – "monlist" requests, (c) CharGen – character generation request, (d) SSDP – "SEARCH" request. Another notable amplification vector is based on Memcached [83]; it was associated with multiple attacks over 1 Tbps in 2018 [84].

Protocol or **State Exhaustion** attacks target specific protocols in L3 and L4 of the TCP/IP stack. The goal of the attack is to starve application servers, load balancers, firewalls and even routers of valuable resources to render the victim unreachable. A typical example of such attack is the TCP SYN flood that sends a massive number of SYN packets with spoofed source IP addresses. The victim server (or even intermediate nodes) waste resources responding and/or tracking these bogus TCP sessions.

Application layer attacks (or Low and Slow) attempt to harm the application (most commonly HTTP-based) itself focusing on inherent vulnerabilities of the protocol and/or the server. Application attacks are typically "slow" restricted by the protocol used by the application and related handshakes. Notably, the source IPs cannot be spoofed, hence the attacks usually originate from infected clients, e.g. Internet of Things (IoT) devices. Typical examples are "Slowloris" and "R.U.D.Y – Are you dead yet?".

DDoS attacks are constantly morphing and evolving, ever-growing in scale and sophistication. Malicious actors often employ a wide variety of vectors from all three categories, creating formidable multi-vector attacks. Such attacks simultaneously target different aspects of a network infrastructure, thus complicating the defense effort. Moreover, DDoS attacks are also used as diversions for other malicious activity such as data theft and intrusion attempts (e.g. Worms) after knocking a firewall or IDS offline.

It would be fair to say that DDoS attacks constitute a major cyber security threat and one of the most prominent problems faced by network operators. This was recently emphasized by the Github (2018, 1.3 Terabits) and Dyn (2016, 1.2 Terabits) incidents. Both highlighted the growth of DDoS attacks in terms of scale, diversity (attack vectors used) and sophistication. The largest attack known today is a reflection/amplification attack against an undisclosed customer of a U.S. based Service Provider [84], peaked at 1.7 Terabits. Other notables DDoS incidents are:

- *June 2019 – Telegram*: Instant messaging service Telegram was hit with multiple attack vectors resulting to user connection issues.
- *September 2016 – Krebs on Security* [85]: Journalist Brian Krebs faced attacks on his blog "Krebs on Security".
- *July 2018 – Blizzard*: Gaming company Blizzard suffered massive DDoS attacks disrupting partially or entirely the availability of servers to players.
- *May 2018 – Danish Rail* [86]: Series of DDoS attacks that knocked offline ticketing and communication systems.
- *March 2013*: SpamHaus was targeted with a massive DDoS attack; the organization reached out to Cloudflare for aid [87].

According to NetScout [88], DDoS attacks between 100 and 400 Gbits have increased approximately by 700% in the first half of 2019 in comparison to the first half of 2018. Frequent and huge attacks regularly exceeding 500 Gbits that have been observed in 2018, are attributed to the Memcached vulnerability that has now been remediated.

The DDoS market is worth approximately $2 billion and is expected to grow even more [89]. Key players include but are not limited to NetScout-Arbor, Radware, Akamai Prolexic, Imperva, NexusGuard, Cloudflare and Fortinet. Interestingly enough the DDoS business is lucrative also for malicious actors, that offer DDoS attacks as-a-Service for quite a small fee. These service providers are commonly referred to as "Booters".

2.6 Mechanisms for Anomaly Detection and Mitigation – Interdomain Collaborative Schemas

As mentioned above, cyber threats, particularly DDoS attacks, constitute a major problem for modern environments. Important topics with regards to DDoS attacks are data collection in high-speed environments, efficient algorithms to timely detect anomalies and techniques to effectively mitigate malicious traffic. Subsection 2.4 provided an overview of monitoring mechanisms, while this subsection focuses on anomaly detection and mitigation [90]–[92].

2.6.1 Anomaly Detection

Anomaly detection efforts rely on different metrics ranging from coarse-grained traffic counters (bits/sec, packets/sec) to fine-grained analytics (flows, Heavy Hitters, top N destinations/ports). Counters are usually unable to accurately tell the difference between a DDoS attacks and a benign anomalous event. Fine-grained analytics typically rely on monitoring protocols that export packets and flows for processing and/or mechanisms for fully fledged Deep Packet Inspection. Important packet/flow fields include but are not limited to: source IP, destination IP, IP protocol, packet size, source port, destination port. DDoS detection techniques analyze measurements as well as related packet and byte counters on such traffic features using various methods.

Various approaches are based on Shannon entropy, in an attempt to find statistical anomalies on traffic features and packet fields [64], [66], [93], [94], [33]. Entropy may be used to identify various network anomalies, mapping entropy fluctuations of a packet field to specific anomalies. As an example DDoS attacks cause entropy to increase for source IPs and to decrease for destination IPs [94], [66]. A similar concept are change-point detection algorithms that track changes in statistical features of the traffic, typically caused by attacks. Closely related approaches might explore moving averages to track the time series evolution of traffic features [33]; confidence intervals may be set to track sudden changes in the moving averages and appropriately trigger alarms.

Often it is sensible to combine detection mechanisms for improved results in terms of detection and performance. Indicatively, such a two-level was introduced in [13]. Initially, entropy values are calculated for the number of destination IPs and ports, with sudden changes indicating an ongoing attack. Subsequently, the victim is identified and traffic destined towards it is redirected to an OF-enabled device. This device acts as a second, more refined level of detection, that uses packet symmetry to identify malicious flows. These flows are subjected to source IP-based aggregation in order to reduce the required rules for blocking the attack traffic, due to OF device capacity limitations.

Intrusion Detection Systems such as Snort [95], Suricata [96], Zeek [97] (formerly Bro [98]) are also applicable for DDoS detection operating on a broad set of rules. These rules track specific attack signatures, connection states and set specific thresholds for alarms. Commonly, they yield accurate results but struggle in large scale deployments due to performance issues. In that regard, Suricata and Zeek (Bro) may employ AF-XDP and Netmap respectively, for performant traffic processing.

Various efforts utilize Machine Learning algorithms to detect and identify network anomalies [99]-[100] in general, and DDoS attacks [64], [101]–[106] in particular; within the context on this dissertation considerable emphasis is placed on the latter. In summary, [101] is based on a Multilayer Perceptron (MLP) whereby traffic metrics related to flows and packet rates (UDP, ICMP) are collected and used to classify network traffic as benign or malicious. Another approach [102], periodically collects OpenFlow (OF) entries from OF-enabled devices, extracts flow-related features and classifies them using Self-Organizing Maps (SOM). In [103], sharp increases in the rate of OF *Packet-In* messages are considered as an indication of DDoS attacks and trigger a mitigation pipeline. Specifically, OpenFlow rules are collected from network devices and are classified via an appropriate Multilayer Perceptron that uses the same feature set as in [102]. Malicious flows are ultimately blocked via appropriate mitigation entries in OF-enabled devices. In [104], a large set of flow-related features is extracted from packets sent to OF Controllers. These are fed to a Stacked Autoencoder, which provides feature reduction and traffic classification of the flow as benign or attack.

In [64] *ATLANTIC*, a two-level framework for DDoS attack detection and mitigation was proposed. Entropy changes for specific flow features within consecutive time windows indicate the existence of an attack. Network flows responsible for entropy changes are fed in a traffic classification component that uses (i) K-means to create clusters of common flows and (ii) SVM to subsequently identify malicious ones. In [105] *DeepDefense*, a DDoS Detection schema based on Recurrent Neural Networks (RNN) was introduced. Traffic traces, collected within sliding time windows, are translated into arrays of packet features. These are fed to an RNN that segregates malicious from benign packets. Similarly, in [106] *LUCID* suggested classification of network traffic based on packet fields. These values are collected from different time windows and organized as arrays; subsequently these arrays are fed to a Convolutional Neural Network to identify time-dependent traffic patterns.

Though a very popular topic, machine learning and deep learning techniques for network use cases need to account for hardware capabilities (data extraction) and strict operational requirements pertaining to the prompt classification of attacks. Some of the approaches mentioned above focus only on the detection whereas other also apply filtering techniques (i.e. OF rules) for the flows classified as malicious. Another point of note is that traffic features are of strategic importance since network anomalies (e.g. DDoS attacks) may attempt to pass as legitimate traffic. In addition, researchers especially in academic institutions have no access to proprietary data feeds and thus face considerable difficulties in obtaining suitable data sets. If obtained, such datasets are unlabeled, raising additional challenges for supervised learning methods. Internet projects such as CAIDA [107] and WIDE [108] provide data from different sample points within their network infrastructure. Regarding datasets containing attack traffic, we refer to the efforts of the University of Twente [109] that performed an extensive experiment, purchasing UDP-based DDoS attacks from Booter services. The experiment was done with the collaboration of the Dutch NREN, SURFnet.

2.6.2 Mitigation Mechanisms

Commercial mitigation solutions may be categorized as on-premise and cloud-based. The former typically use hardware or VM-based appliances operating in-line (always on) or on-demand in case an attack is detected. Organizations may also opt to redirect their traffic to dedicated infrastructures, i.e. scrubbing centers, whereby malicious traffic is filtered and benign traffic is forwarded back to them via dedicated connections. These cloud-based services grow in popularity; however, they may raise privacy concerns and introduce significant latency. Powerhouses in DDoS protection services such as Arbor and Imperva, offer both solutions to their customers. Commercial mitigation solutions (cloud or appliance-based) employ proprietary algorithms and/or hardware but offer moderate or no flexibility while requiring considerable capital expenses.

In addition to appliances and related services, network operators may use additional mitigation techniques to defend against DDoS attacks. A brief overview of typical techniques is presented below:

Destination-based RTBH [110]**:** This mechanism is primarily used to prevent potential collateral damage during a DDoS attack (e.g. bandwidth and CPU utilization, degradation of other services). It is a destination-based filtering mechanism, in which the traffic destined to the victim is redirected to an edge router's null interface. The dynamic redirection from the victim AS is triggered using a device that retains BGP (iBGP) peerings with edge routers. Subsequently, the blackholed route (usually /32) is also propagated to peers or upstream providers ([111]-[112]) to alleviate stress on peering/upstream links. As a result, both malicious and benign traffic destined to the victim is dropped.

Source-based RTBH [113]**:** Unlike the destination-based RTBH which renders the victim unreachable, the source-based RTBH drops packets from specific source IPs via the unicast Reverse Path Forwarding (uRPF) feature [114]. Source-based RTBH also relies on BGP updates which contain routes to malicious IPs; attack packets from these sources are dropped on the uRPF-enabled router interface. Although it offers more granularity than destination-based RTBH, outgoing packets to legitimate destinations may be blocked if attackers employ en route and fixed route spoofing [90].

Access Control Lists: Access Control Lists (ACLs) are commonly used to implement firewall policies. Filtering rules are typically implemented in specialized hardware such as TCAM that enables traffic processing in line-rate. ACLs may be propagated and installed in network devices using various protocols and mechanisms, that are further discussed in section 5.

BGP Flowspec [115]**:** This mechanism extends the Network Layer Reachability Information (NLRI) field of BGP to disseminate traffic flow specification rules. These rules are transported over BGP and dynamically installed on appropriately configured devices. The exact hardware implementation of Flowspec rules is the responsibility of the vendor. In comparison to ACLs, Flowspec rules provide a unified specification of rules transmitted over BGP. However, Flowspec is not widely deployed; a notable exception is the Firewall on Demand (FoD) [116], a service offered by GÉANT that was initially developed by GRNET.

OpenFlow [9]**:** OpenFlow-enabled devices contain flow tables and determine packet forwarding based on flow rule entries matched against packet headers. There is a plethora of matching capabilities and actions/instructions, used to instantiate typical firewall operations such as packet rejection/redirection.

The above techniques are summarized in the following table, with emphasis on their adoption and granularity:

Mitigation techniques	**Distribution Protocol**	**Granularity**	**Adoption**
Destination–based RTBH	*BGP*	Low	High
Source-based RTBH	*BGP*	Medium	Medium
OpenFlow Firewall	*OpenFlow*	High	Low
Flowspec	*BGP*	High	Low
ACL	*SSH/NETCONF*	High	High

Table 2.6 Mitigation Techniques

The entries of Table 2.6 were partially inferred from "*DDoS using BGP Flowspec*" by Juniper Networks[3]. The mitigation techniques presented above need to account for the following:

- *Scalability*: Sources of DDoS attacks (spoofed or not) might be considerably large. Thus, approaches that rely on source filtering might struggle to implement the required mitigation rules, due to hardware constraints. This was emphasized in [13], where prefix aggregation techniques were employed to limit the number of required rules.
- *Spoofed Sources and Impact on benign traffic*: Filtering techniques typically use source IP addresses to mitigate malicious traffic rendering them ineffective or even harmful to benign sources due to widespread spoofing. Destination-based RTBH, frequently used for its simplicity, renders the victim inaccessible.

Considering these shortcomings, programmable data planes may be used to provide on-demand, robust and fine-grained filtering mechanisms for DDoS mitigation. Specifically, the XDP framework seems a promising candidate to create filters tailored

[3]https://www.slideshare.net/apnic/ddos-mitigation-using-bgp-flowspec

to specific DDoS vectors. These filters may be dynamically deployed and filter malicious traffic based on unique characteristics, not only source IP addresses. P4 offers similar capabilities as well, but implementations have to account for potential downtime in case reconfiguration of the pipeline is required.

Another mechanism is *anycast* traffic diffusion. In short, this technique uses BGP to advertise the victim from many different points of presence, distributed across the internet. Consequentially, the traffic is dispersed to many sites whereby it is processed and filtered. However, this approach assumes adequate points of presences and sufficient link capacity and processing power to each one. These requirements can be met only by a very small number of cutting edge organizations. Such an example is Cloudflare that maintains a point of presence in over 200 different cities and 90 countries, fields vast processing power, peers directly in various locations and has special bandwidth agreements.

2.6.3 Collaborative Schemas

As mentioned, organizations typically contract third party scrubbing providers or implement mitigation within their own network domain. However, the sheer volume of present-day cyber threats may overwhelm an individual provider, thus the emerging need for collaborative mitigation efforts as malicious attacks are more efficiently mitigated closer to their sources. Interdomain collaborations are manifested as part of multilateral agreements or within trusted federated environments. An indicative example is the hierarchical federation for the European research community: GÉANT, NRENs (National Research and Education Networks) and Campus networks. Other schemas can be formed, provided that collaborators follow agreed upon admission procedures and adhere to standards such as MANRS [117]. However, defense collaborations might be hindered by operator concerns such as unwillingness to share victim-related information to preserve sensitive client data, lack of incentives for cooperation and shortcomings of incident handling mechanisms.

Various research efforts have been explored for the collaborative detection and mitigation of cyber threats. Indicatively, CoFence [118] is a framework that enables collaborating NFV-enabled infrastructures (i.e. ISPs) to mitigate DDoS attacks using available compute and network resources. These are allocated in a reciprocal manner based on past mitigation collaborations. In [119], an SDN approach was proposed, featuring inter-domain collaboration via the exchange of IODEF [120] messages on top of BGP; reputation score for neighbors is evaluated via the Beta Reputation system [121]. 3DCoP [122] is also a P2P system for DDoS detection and mitigation whereby network domains collaborate to provide monitoring, alerting and ultimately mitigation of malicious flows. IETF proposed the DDoS Open Threat Signaling (DOTS) protocol [123] that specifies interactions between domains under attack and potential mitigators while considering adverse network conditions and related limitations of the signaling channel. The establishment of business relationships and collaboration incentives is not a main objective in DOTS activities.

Approaches based on Distributed Ledger Technologies (DLTs) have recently been proposed as a promising way to enhance the coordination between collaborators for detection and mitigation of security incidents. Indicatively, in [124] authors introduced a federated schema where monitoring data are exchanged to collaboratively detect network anomalies. The federation is based on a permissioned blockchain framework that ensures transparency and business regulation. Similarly, in [125] the use of a private blockchain was proposed to avoid verification delays commonly occurring in public ledgers. Another approach aims at providing DDoS mitigation services to third parties via sharing of user resources; this was orchestrated in Gladius, an Ethereum-based platform [126] that verifies web requests and drops illegitimate ones.

In [127] a cross-domain collaborative schema for DDoS mitigation was introduced, whereby the cooperation signaling relies on an Ethereum network. Malicious IPs are advertised in blockchain-based Smart Contracts (SCs) issued by the victim. These are retrieved by interested ASes which in turn may trigger mitigation actions. However, inserting large blocks of IP addresses in the blockchain may introduce significant latencies to the mitigation process. As an extension to [127], authors presented in [128] a reputation scheme based on the Beta Reputation system in order to rate mitigation services and prevent abuse/misuse by modeling customer strategies. Additionally, [129] explored mechanisms for verifying an attack was indeed mitigated to prevent false reporting.

3 Traffic Monitoring and Anomaly Detection based on Dispersed Vantage Points

This section considers network monitoring and anomaly detection mechanisms offered as a service to users (tenants and administrators) within shared network infrastructures. Measurements are collected from scattered monitoring agents and directed to a data pipeline for processing and enrichment.

3.1 Problem Statement

Traditionally, network environments follow a hierarchical structure organized in core, distribution and access/edge layers, essentially defining distinct vantage (observation) points for monitoring network traffic. Within such environments, network traffic is aggregated at the core and exhibits localized characteristics near the edge. Selecting a vantage point for information extraction directly influences network visibility, since monitoring is commonly implemented via sampling mechanisms e.g. NetFlow and sFlow.

Monitoring services are valuable to authorized users (tenants and administrators) who own distinct subsets of networked resources (physical and/or virtual) within shared infrastructures. Each tenant "owns" a basket (slice) of virtualized resources drawn from distributed physical resources and normally requires measurements pertaining to these virtual or physical resources. Slice monitoring requirements in federated computer/networking infrastructures [130] raise considerable problems in terms of agent location, data storage, filtering/processing and access control.

In this section a framework for data collection and processing is proposed, offering on-demand network monitoring data and related analytics services as VNFs to users within shared environments. Our implementation focuses on sampled measurements from various vantage points. Collected data are associated with relevant parties and may be accessed based on predefined policies to project dynamic monitoring views and personalized analytics.

3.2 Background and Related Work

Several approaches investigate monitoring system/network resources and services within multi-tenant cloud/NFV infrastructures. MonPaaS [131] is an OpenStack based

Platform-as-a-Service (PaaS) for monitoring resources owned by both cloud providers and consumers (tenants) alike. In [69] monitoring mechanisms are presented that extract data from the NFV Infrastructure and software agents within the VNFs. D-StreaMon [70] is an NFV-capable distributed framework that uses containerized probes for traffic monitoring and analysis. Most approaches interact with the infrastructure (traffic redirection/ monitoring probes) or are tightly coupled with the cloud management system. Our approach is decoupled from the underlying network architecture and is applicable in hierarchical campus, spine-leaf network architectures (datacenters and cloud environments) as well as NFV Infrastructures.

Internet measurement projects [132] deploy active/passive probes and Looking Glass tools in a wide variety of networked infrastructures e.g. campus Local Area Networks (LANs), Internet Exchanges (IXs) and internet provider networks. Probes and tools essentially define scattered vantage points, useful for troubleshooting and verification. In [133] the diversity and geographical distribution of measurement points is evaluated, comparing results obtained from fixed network locations against measurements from agents running on end-user equipment. In [134] a mechanism is presented for inferring relations among Autonomous Systems (ASs) using partial views based on different observations via looking glass tools. In [135] the suitability of a large European IX as an advantageous observation point is investigated, offering prime visibility not only into European but also global network behavior.

3.3 Design Principles

Our proposed architecture is based on an NFV-compliant framework, offering sampled network monitoring data and related detection for security incidents. These capabilities are implemented as on-demand services available to authorized users (tenants and administrators) within shared network environments. Network traffic monitoring emanates from vantage points that users can dynamically select from predefined key network locations e.g. the network core, the network edge, etc. The benefits are twofold; (i) administrators in Network Operation Centers (NOCs) can dynamically select monitoring views to detect, pinpoint and verify network anomalies, while (ii) tenants are capable of requesting on-demand access to raw data pertaining to their slice. In addition, our schema provides users with a selection of customized tools for monitoring data warehousing, analytics extraction and visualization, enabling them with actionable intelligence. Our design adheres to:

- *User Specific Monitoring Data*: Initially data streams are collected on a centralized data warehouse after being tagged appropriately (per user or vantage point). Additionally, users are able to request data collection from specific devices (i.e. vantage points) within the network, thus achieving a granular monitoring view. The data are presented to authorized users, based on predefined access control policies and attached tags.
- *Monitoring Data Enrichment*: Raw data streams are ingested and analyzed in a data pipeline. To that end, we leverage on popular message brokers facilitating asynchronous processing. Notably, a user is able to choose a number of enrichment tasks that should take place upon the data pertaining to their monitored slice.
- *Anomaly Detection based on Monitoring Views and Data Analytics*: Following the drill-down and roll-up concepts of On-line Analytical Processing (OLAP) [136], the proposed framework offers zooming in and out between user-focused and centralized data. This may improve network visibility for specific tenants and localized anomaly detection mechanisms. To that end, we may deploy on-demand lightweight anomaly detection services, each pertaining to a given monitoring view.
- *NFV-Compliant Approach*: Monitoring and anomaly detection functionality is offered via the deployment and orchestration of containerized services. Our approach leverages on a popular unified management system of virtualized resources i.e. Kubernetes [137] that offers advanced resource allocation, orchestration and service chaining capabilities.

3.4 Architectural Components and Implementation Details

In this section we discuss implementation details pertaining to the architectural components and related sub-modules. Depicted in Figure 3.1 is an indicative setup for our proposed architecture.

Monitoring data are exported from agents within network devices assumed to be vantage points. This operation is performed via the *sFlow* protocol, widely implemented by vendors of L2 devices employed in LANs. Traffic traces are fed to an sFlow-enabled Open vSwitch mounted in our laboratory. This switch exports packet samples to an sFlow collector which also dispatches samples to an external cloud infrastructure through a network tunnel. The core of our NFV-compliant architecture was deployed on ~okeanos [138], the Greek National Research and Education Network (GRNET) cloud infrastructure.

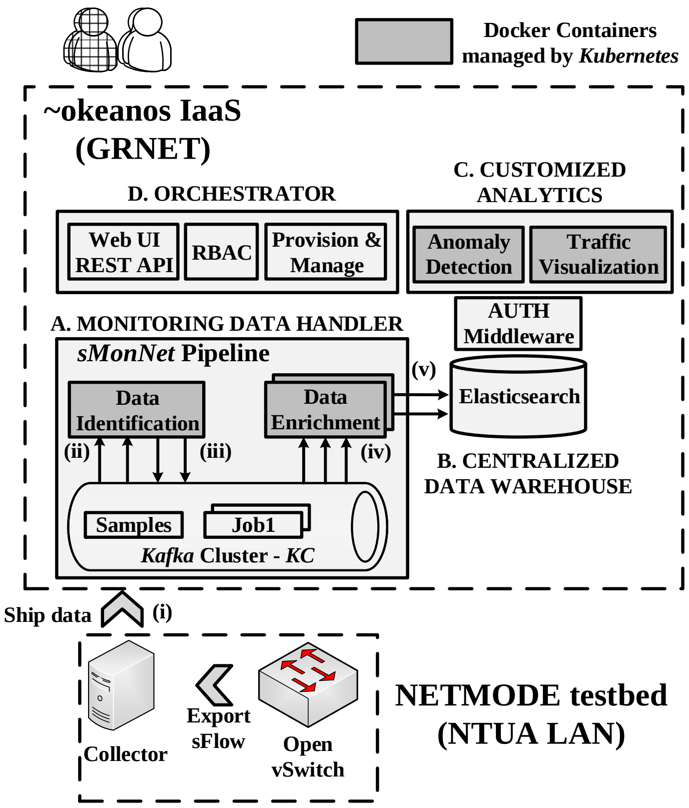


Figure 3.1 sMonNet Architectural Setup

3.4.1 Monitoring Data Handler

Our architecture requires message broker components between the various modules in the Monitoring Data Handler shown in Figure 3.1. To account for the vast amounts of data, expected in a large-scale networking infrastructure, an option would be to use standard big data processing frameworks as suggested in [139]–[141]. Instead of these approaches we opted for a lightweight customized solution based on *Docker*. Consequently, we developed *sMonNet* (*s*Flow *Mon*itored *Net*work), a Python module tailored to data identification (tagging) and enrichment. Regarding the latter, users may select an enrichment such as inserting ASN and GeoIP metadata (e.g. from MaxMind).

sMonNet leverages on *Kafka* [142], a fast, distributed and fault tolerant messaging system supporting large volumes of I/O per second from and/or to multiple systems. The Kafka broker enables: (i) acceptance of incoming message streams, (ii) temporary data storage and (iii) shipping data upon request, to the corresponding processing &

enrichment instances. To improve performance during identification and enrichment we used C libraries *cjson*, *libmaxminddb*.

As shown in Figure 3.1, Kafka brokers are organized in a cluster, denoted as *KC*, which contains monitoring data separated in different *topics*. (i) Switches export samples to an *sFlow* collector that produces (ships) all samples to *KC*, in a dedicated *topic*, named *samples*. (ii) The identification component of *sMonNet* consumes (reads) from the *samples* topic and inserts appropriate tags per user and vantage point (device). Users are associated based on L2/L3 sample headers whereas a vantage point is identified via the sFlow agent's IP. (iii) Tagged data are shipped back to *KC* in a dedicated topic per selected enrichment task. (iv) For each new task an *sMonNet* instance is created; such instance consumes data from the dedicated topic. (v) After enrichment, processed samples are stored within the Centralized Data Warehouse.

3.4.2 Centralized Data Warehouse

This component maintains tagged and enriched data per user and vantage point. Users have full access rights on their data in order to drive customized management applications e.g. traffic analysis, performance verification, intrusion detection and prevention. In addition, network administrators are able to query the Data Warehouse for infrastructure-wide data.

The centralized data warehouse is based on an *Elasticsearch* cluster; a search and analytics engine featuring parallel data insertion (i.e. indexing) and rapid queries. Towards our requirements for Authentication and Access Control, we implemented a middleware component using the *Elasticsearch* client Python library. This middleware enforces related tags and dispatches queries (requests) to the *Elasticsearch* cluster. These tags behave as filtering mechanisms, and are applied based on either user credentials or the originator IP associated with a request.

3.4.3 Customized Analytics

Stored data are available for a multitude of applications e.g. traffic visualization and analysis services. Building on earlier efforts [66], we focus on identifying network security incidents from appropriately selected vantage points. Specifically, we alternate between a user-specific localized monitoring view and a global infrastructure view, in order to improve incident detection mechanisms.

As a proof concept we have implemented two distinct services for Anomaly Detection and for Traffic Visualization. Additional services may be developed using the data gathered in the *Elasticsearch* Data Warehouse.

- The *Anomaly Detection* service implements entropy-based algorithms. As mentioned, data are gathered from multiple vantage points typically, located in core, aggregation, and edge switches in LAN environments.
- The *Traffic Visualization* service is based on *Kibana*, a software component natively able to visualize data stored in *Elasticsearch*. Users are provided with a pre-defined but easily configurable dashboard for their network resources.

3.4.4 Orchestrator

From a high level standpoint, this component deploys and configures data collection, processing and analysis services offered by the Monitoring Data Handler and Customized Analytics components.

In Figure 3.2 below, we illustrate a container-based NFV Infrastructure (NFVI) which is based on adapting the NFV framework [43] to deliver user specific Monitoring and Analysis services. The Management and Orchestration (MANO) functionality was implemented on *Kubernetes* [137].

The Orchestrator component: (i) receives and validates service requests, (ii) configures running services and (iii) deploys *Docker* containers (*Pods*). The Monitoring Data Identification & Enrichment modules in the Monitoring Data Handler 3.4.1 and the Anomaly Detection and Traffic Visualization services in the Customized Analytics component 3.4.3 are implemented as pre-packaged and configurable *Docker* containers orchestrated by *Kubernetes* depending on user requests.

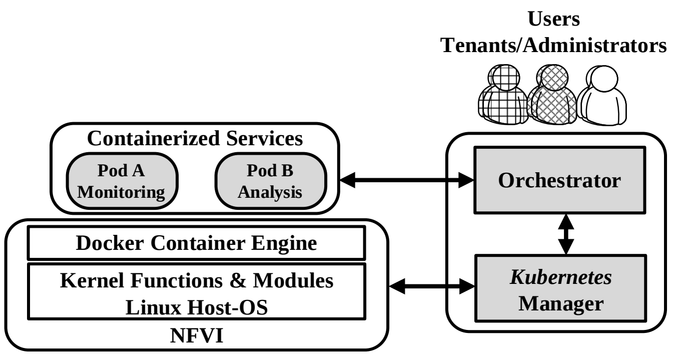


Figure 3.2 Container-based NFV architecture

In the sequel we analyze the orchestration workflows depicted in Figure 3.1. Service requests are received by an external-facing web/RESTful interface and are validated based on user credentials (username, password, token). A Role Based Access Control (RBAC) component maintains associations between users, network segments (L2/L3 headers), and tags used for data identification. Such information is conveyed to the AUTH middleware component that enforces access control to monitoring data.

Orchestration of Monitoring Requests: Users may dynamically request monitoring data collection combined with enrichment tasks. Based on user credentials, packet headers and related tags are retrieved and conveyed to the Identification component. Regarding *Kafka topics,* we define a general purpose topic as the default i.e. no specific enrichment job. Else, a *KC topic* is specified to the Identification component, overriding the default. In such case individualized enrichment, dedicated containers are deployed, consuming data from the corresponding *KC topic*. Enrichment instances are provided with connection details for the *Elasticsearch*.

Orchestration of Anomaly Detection and Traffic Visualization Requests: The Orchestration component instructs *Kubernetes* to create containers implementing Customized analytics services defined in section 3.4.3 above. These containers may request monitoring data from various vantage points, projecting different monitoring views. As mentioned, data filtering and isolation is enforced by the AUTH middleware component. Granularity and localization is enabled by smart selection of sampling rate in each vantage point as well as the algorithms used for Anomaly Detection.

3.5 Evaluation

In this section, we present our testbed setup and discuss experimental results.

3.5.1 Experimental Setup

We provisioned 11 VMs for our Kubernetes, Elasticsearch and Kafka cluster. Identification, enrichment and anomaly detection instances are implemented as ephemeral containers managed by Kubernetes. Kafka and Elasticsearch clusters were mounted on stand-alone VMs due to their considerable performance and deployment requirements.

Experiments were conducted by emulating real network conditions using benign traffic traces from NTUA Campus LAN (core) and our laboratory switch (edge), essentially defining two distinct vantage points. Moreover, we utilized Scapy in order to construct network anomalies emulating: (a) worm propagation from a /24 NTUA laboratory subnet and (b) port scans targeting the entire /16 campus network.

Our framework is modularly designed, thus allowing for a wide variety of anomaly detection methods. In our experiments, we adopted a commonly used approach [143] calculating the normalized entropy values for network traffic. We focused on source IP addresses; other values may be considered as well, including but not limited to L4 headers. Learning mechanisms for anomaly detection services within our Customized Analytics module, were based on average entropy values from normal (reference) traces. Alerts are triggered if entropy values deviate from established references, either above or below a threshold. We set thresholds empirically to 3% for the core and 5% for the edge, to account for different traffic patterns. Near the edge traffic is less diverse thus a higher threshold is needed to reduce false positive alarms. Anomalies are usually subtler near the core that aggregates all different subnets. Hence, a lower alert threshold might be preferable.

3.5.2 Experiments on Multi-Vantage Point Anomaly Detection

In our experiments, we considered two Anomaly Detection services focusing on data gathered from core and edge vantage points in 30 second detection windows. For each detection window, both services periodically query the *Elasticsearch* cluster for samples (*documents*) stored during the last 30 seconds in order to calculate the entropy values. We show the results for both applications for a 5-minute observation period (10 distinct detection windows).

In Figure 3.3 we illustrate the normalized entropy for the source IP address during the worm propagation and in Figure 3.4 during port scans. Grey and black bars refer to edge and core vantage points respectively. Their height is significantly different (approximately 0.2) attributed to the characteristics (traffic diversity) of each vantage point. In both cases, anomalies were detected and alarms were issued in detection windows 3 to 5, exactly fitting our attack injection scenarios.

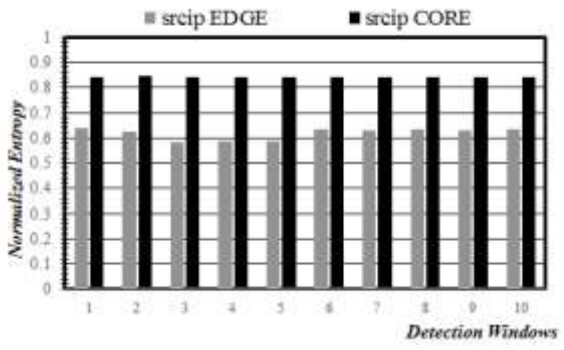


Figure 3.3 Entropy Values for Source IP, Worm Propagation, 30 second detection windows

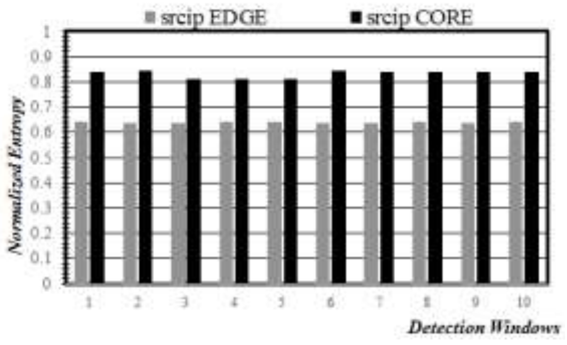


Figure 3.4 Entropy Values Source IP, Port Scan, 30 second detection windows

These results demonstrate the benefit of a multi-vantage point approach in anomaly detection. Worm propagation is better highlighted in a localized context due to lower traffic volumes and proximity to the problem. Thus, as shown in Figure 3.3 anomaly detection based on data obtained from the edge vantage point yields better results than the network core. However, vantage points near the network edge are somewhat sensitive to slight changes in traffic patterns, thus prone to false positive alarms. Moreover, during a network-wide attack e.g. port scan, a generic/global monitoring view is more suitable for anomaly detection. Hence, as shown in Figure 3.4, core-centric approaches are able to highlight global issues whereas edge approaches are less conspicuous due to limited (localized) visibility.

4 Multi-Feature DDoS Detection on Programmable P4 Hardware

The approach presented in section 3 focuses on collecting and processing monitoring data exported from dispersed vantage points in hierarchically layered network environments. Extensions to that work are addressed in this section, specifically the migration of monitoring and anomaly detection tasks to programmable data plane devices. Each such device may be considered as a distinct vantage point that performs related tasks in a distributed fashion.

4.1 Problem Statement

As mentioned, anomaly detection relies on packet samples or flow records. These are exported from agents within network devices (routers, switches) and relayed for processing to external collectors (servers), typically deployed in a centralized fashion. Similarly, SDN setups (e.g. OpenFlow [9]) employ control plane signaling between network devices and controllers to retrieve information, detect anomalies and subsequently deploy mitigation actions.

In contrast, continuously evolving programmable hardware, e.g. SmartNICs [26], enable the migration of anomaly detection workloads to hardware. This distributed approach attempts to be one step ahead of related schemas usually associated with a form of centralized collection, processing and/or control.

In this section an in-network DDoS detection mechanism is considered that offers rapid detection, while enabling control plane triggers to external mitigation systems. Our approach leverages the P4 language [5] and combines important traffic features to increase accuracy while adhering to performance penalties. In a nutshell, the proposed mechanism: (i) inspects network traffic and computes related metrics (features) per protected subnet, (ii) evaluates feature values within time-based epochs to identify potential threats and (iii) conveys alarms to external systems (P4 “digests”).

4.2 Background and Related Work

There are various efforts exploring performance capabilities of advanced network applications implemented in programmable hardware. In [144], the impact of basic P4 operations (packet parsing, headers modifications) on packet processing performance is explored. Experiments are based on P4-enabled Netronome SmartNICs [26] (*Agilio CX*) and illustrate the effect on processing latency introduced by various P4 constructs. Similarly, in [145], the impact of XDP operations on various system resources is investigated. Specifically, results demonstrate packet processing limitations and scaling capabilities (number of CPU cores) considering different flavors of XDP. The main contributors of XDP [4] also present indicative performance metrics for different use cases such as IP routing, DDoS Mitigation and Load Balancing.

Recent research efforts on data plane programmability applied to detection of DDoS attacks are reported in [33], [146]. In the former, a P4-based DDoS detection approach is proposed; counting Bloom Filters are used to track the per-flow ratio of TCP SYN to regular TCP packets in order to detect SYN flood attacks. In the latter, a DDoS detection schema is presented that estimates entropy values of source and destination IP addresses. These values are compared to appropriately defined thresholds and upon their violation DDoS alarms are triggered, without however further indicating the victim.

We provide an integrated framework able to promptly detect generic DDoS attacks to specific victim subnetworks, possibly alerting external DDoS mitigation systems via P4 digests. Furthermore, we deploy our P4 schema in hardware SmartNICs and assess its performance in terms of attainable packet processing rate and detection accuracy.

4.3 Architectural Design and Selected Traffic Features

Suitable environments for the proposed schema are both transit provider networks (e.g. ISP, Research & Education Network backbones) and customer/edge network domains (e.g. Data Centers, Campus Networks). Upstream providers may detect network anomalies that target downstream organizations. Similarly, customer organizations may deploy the same functionality with fine granularity for specific internal subnetworks.

Such an indicative architectural setup is presented in Figure 4.1: Traffic originating from various Internet sources is directed towards a P4-enabled edge domain, possibly via a P4-enabled transit provider. We precisely consider the use case of National Research and Education Networks (NRENs) and their Pan-European interconnection GÉANT. NRENs, e.g. GRNET, may offer DDoS Protection services to universities and data centers downstream. These services are implemented in P4-capable devices, placed at important vantage points to monitor traffic at different levels of granularity.

Specifically, P4 devices: (i) forward network traffic, (ii) maintain important statistics for monitored (sub)networks, (iii) perform anomaly detection tasks and (iv) raise alarms to external mitigation systems.

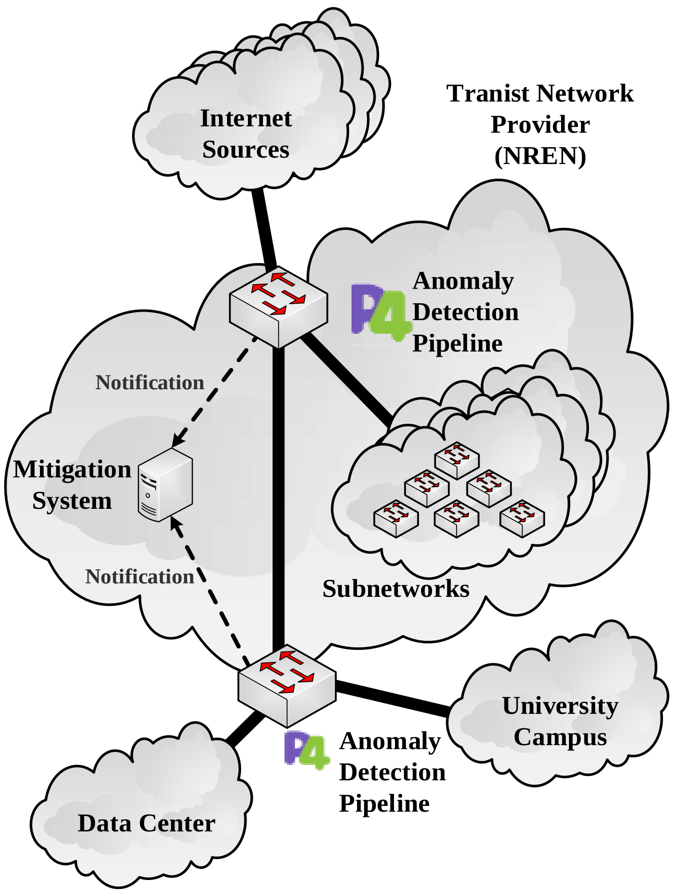


Figure 4.1 High-level Overview of P4-based Anomaly Detection

Our schema maintains a list of specific monitored (sub)networks and/or hosts, depending on the desired granularity level. DDoS attacks are detected by combining the following traffic features: (i) total number of incoming traffic flows (*srcIP*, *dstIP*, *Protocol*, *srcPort*, *dstPort*), destined to monitored subnets in a distinct time interval henceforth denoted as "epoch", (ii) significance of a network, characterized by the percentage of flows directed towards it out of the total incoming flows and (iii) symmetry ratio of incoming to outgoing packets. These features have correlated characteristics and may provide localized alarms for each protected network under generic DDoS attacks.

Typically, massive DDoS attacks consist of a considerable amount of flows [147]. Thus, we consider the number of total flows as an attack indicator. We adopt a moving average approach as in [33] to track for each epoch n the number and the dispersion of Total Incoming Flows (TIF_n). Specifically, we define M_n as the Exponentially Weighted

Moving Average (EWMA) and D_n as the Exponentially Weighted Moving Difference (EWMD):

$$M_n = a \cdot TIF_n + (1-a) \cdot M_{n-1} \text{ with } M_1 = TIF_1$$

$$D_n = a \cdot |\, M_n - TIF_n \,| + (1-a) \cdot D_{n-1} \text{ with } D_1 = 0$$

The parameter a is a smoothing coefficient to dampen short-term fluctuations. Network anomalies are considered in case TIF_n exceeds a threshold that depends on the values of M_{n-1} and D_{n-1}:

$$TIF_n > M_{n-1} + k \cdot D_{n-1} \qquad (1)$$

where $k \geq 0$ is a sensitivity coefficient that scales the detection threshold [33].

In order to further pinpoint the victim destination subnetwork, we also incorporate two additional features, namely Subnet Significance and Packet Symmetry.

- *Subnet Significance* is expressed as the percentage of Incoming Flows $SIF_n^{(i)}$ destined to a subnet *i* in epoch *n* out of the Total Incoming Flows TIF_n. We indicate an alert if this percentage exceeds a significance factor f that identifies major flow recipients as potential victims:

$$\frac{SIF_n^{(i)}}{TIF_n} > f \qquad (2)$$

- *Packet Symmetry* is an insightful metric [148] to avoid classification of a subnet as a victim while it may be a recipient of heavy benign traffic, to which it generates responses. The Current Packet Symmetry Ratio $CR_n^{(i)}$ is defined as the fraction of incoming to outgoing packets for subnet *i* during epoch *n*. These are evaluated based on per subnet *i* counters and compared against a pre-computed Normal Packet Symmetry Ratio $NR^{(i)}$. We consider traffic to a subnetwork anomalous, in case the corresponding fraction exceeds a heuristic threshold *r* as described in the following condition:

$$\frac{CR_n^{(i)}}{NR^{(i)}} > r \qquad (3)$$

Values for f, r and $NR^{(i)}$ are defined based on operational experience under normal (non-attack) network conditions. Note that, these parameters could be set by Machine Learning algorithms that learn from past traffic patterns.

4.4 P4 Implementation Details

In this section we elaborate on implementation details of the proposed DDoS detection pipeline. Our mechanism utilizes P4 registers to implement counters, arrays and probabilistic data structures. We do not use P4 counters since their values are only accessible via control plane signaling and may not be used directly in data plane interactions [34]. Table 4.1 contains indicative register definitions:

Functionality	Indicative Definition	Usage
Counters	register<bit<32>>(1) epoch	Epochs, Total Flows
Array of counters	register<bit<16>>(256) flow_dst	Per Subnet Flows, Packets
Probabilistic Data Structures	register<bit<32>>(65536) sketch	Flow Tracking

Table 4.1 P4 Registers: Functionality, Indicative Definition and Usage

The processing pipeline is depicted in Figure 4.2. Traffic arriving at the P4-enabled device is filtered to include only relevant packets. Subsequently, we apply our multi-feature approach in distinct serial steps to identify potential attacks. In case all violations are observed, we generate alarms (i.e. P4 digests [34]) to an external mitigation system.

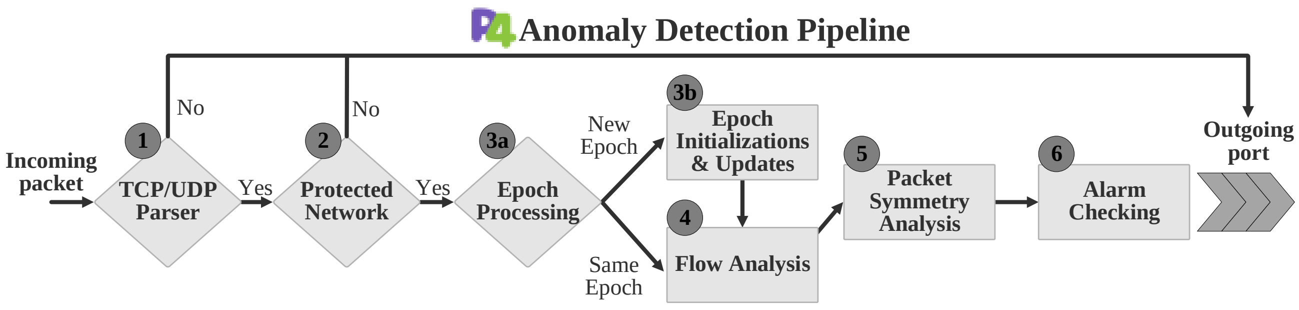


Figure 4.2 P4 Anomaly Detection Pipeline

Step 1 selects only TCP or UDP packets to be considered within the DDoS detection pipeline, since they are typically utilized by most attack vectors [149]. This is achieved using simple checks on parsed headers.

Step 2 further isolates traffic originating from or destined to a monitored network (protected network). To that end, we employ a dedicated match-action table that contains one rule for each protected network. Each rule adds a unique identifier to matching packets as P4 metadata. The added metadata headers are used to access and update the equivalent memory areas of various registers e.g. per subnet measurements such as flows and packet statistics. Note that, traffic that does not meet the aforementioned criteria (i.e. TCP/UDP and source/destination in “monitored” networks), bypasses the DDoS detection pipeline and is appropriately handled.

Step 3a delimits time-based epochs, each defined by a start time and duration. Packets are associated with an epoch using the *ingress_global_timestamp* packet metadata. This denotes the exact time a packet arrived at the P4-enabled device. If a packet’s timestamp fits within the current time window [start_time, start_time + epoch duration), it is directly fed to **Step 4**. Otherwise, the packet is assigned to a new epoch and proceeds to **Step 3b**. The latter performs the following: (i) update the new epoch start time, (ii) increment the index tracking the current epoch, (iii) compute the new EWMA and EWMD values as described in section 4.3 and (iv) reset the number of total flows.

Step 4 performs flow traffic analysis and maintains appropriate flow counters for packets exiting from either **Step 3a** or **3b**. This operation is based on modified *Bloom Filters* [150], used to track unique active flows within an epoch. Specifically, we calculate hash values from the following packet headers (*srcIP*, *dstIP*, *Protocol*, *srcPort*, *dstPort*) that identify a flow tuple. We employ hash functions available in the P4 pipeline, i.e. *CRC32*, *CRC16* and *CSUM16*. The resulting hash values are used as indices to access distinct memory areas of P4 registers. Each area stores the last epoch a flow was observed. A flow is considered “active” in the current epoch when all indices point to register areas containing values equal to the current epoch. Else, the flow is considered as newly observed within this epoch and the register contents for these indices are set to the value of the current epoch. Additionally, when a new flow is observed, counters pertaining to total flows and per subnet flows are incremented. Based on these counters, conditions pertaining to inequalities (1), (2) of section 4.3 are evaluated. In case a threshold is violated, the equivalent flag is stored in distinct packet metadata headers.

Step 5 performs packet symmetry analysis employing incoming and outgoing packet counters from/to a monitored network. We maintain separate per-subnet packet counters for TCP and UDP traffic, as well as historical normal packet symmetry ratios for both protocols. These are used to evaluate the $CR_n^{(i)}$ against the $NR^{(i)}$ as depicted in inequality (3). In case this fraction exceeds the threshold r, a flag is raised similarly to the ones for threshold violations (1), (2).

The final **Step 6** of our pipeline checks packets for metadata headers corresponding to identified anomalies. In case all metadata headers are set to “True”, an appropriate alarm is generated pinpointing the network under attack and the current epoch. These alarms were implemented as P4 packet digests that enable the communication between the data plane and external systems; in our case appropriate mitigation mechanisms able to enforce countermeasures.

Note that, P4 is a programming language with specific restrictions, e.g. no support for floating point arithmetic or division operations. We needed to adapt to P4 limitations using various workarounds since our approach uses real values e.g. the smoothing coefficient a in EWMA, EWMD values and divisions, e.g. $CR_n^{(i)}/NR^{(i)}$, for its calculations.

The former are approached by multiplying all elements of an equation with a power of 2 and subsequently dividing them by the same factor. The latter, are conducted via appropriate bitwise shifting operations. We present an example for the EWMA equation; specifically, for the smoothing coefficient a, we selected the value of 1/256 (~ 0.004):

$$M_n = \frac{1}{256} \cdot TIF_n + \left(1 - \frac{1}{256}\right) \cdot M_{n-1} \Leftrightarrow$$

$$M_n = (TIF_n + 255 \cdot M_{n-1}) \gg 8$$

where eight bits right shifting corresponds to division by a factor 2^8 = 256. We satisfied requirements for division via a plain comparison between two numbers. Note that, we are not interested in the quotient of a fraction but whether it is greater or lower than another value. For example, the threshold evaluation in inequality (3) was implemented as:

$$CR_n^{(i)} > r \cdot NR^{(i)}$$

4.5 Evaluation

4.5.1 Experimental Setup

In order to validate our DDoS detection framework, we implemented the proposed pipeline in P4 and evaluated it in the testbed illustrated in Figure 4.3. We used as a P4 target the 10G version of Netronome *Agilio CX SmartNIC* [26]. Programs were developed and compiled via the Netronome *Programmer Studio* and ultimately loaded to the NIC. Additionally, we used two VMs operating as the Sender and the Receiver, equipped with 10G Intel-based NICs, able to generate and count packets in high packet rates. The evaluation process and related results focus on the detection accuracy and packet processing performance of our DDoS detection schema.

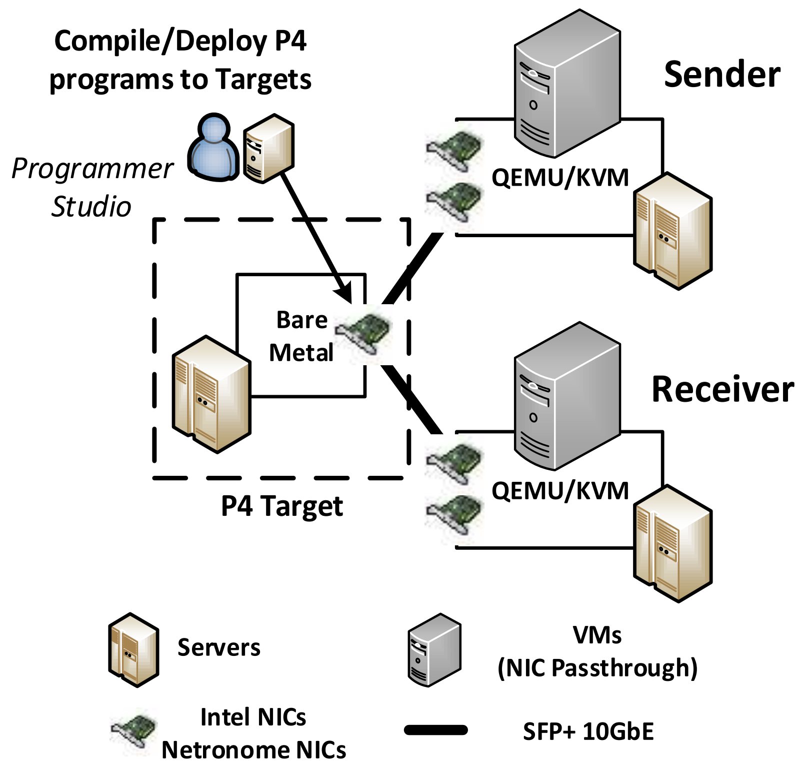


Figure 4.3 P4 testbed equipped with 10G SmartNICs

4.5.2 DDoS Detection Accuracy Assessment

In order to create realistic conditions for our experiments, we used publicly available network traces both for benign and malicious traffic. The benign traffic is based on traces available from the WIDE backbone [108]; specifically traffic from a 10G transit link between WIDE and DIX-IE, an experimental Internet Exchange (IX) in Tokyo. The traces contain network traffic between 12:00 - 12:15 on 09/04/2019.

For malicious traffic traces we used the fourth dataset, B4, as reported in [109]. This contains a DNS-based reflection attack generated by Booter services. Protected subnetworks were identified based on an analysis of the benign dataset. We selected the top 255 networks, assuming /24 prefixes, as ordered by the total number of packets traversing each subnetwork.

The experimentation process considered 1 second epochs and was conducted as follows: We injected the benign traffic and ignored alarms for the first 30 seconds, considering them as a "learning" period for the moving averages. Between seconds 30 and 60 we observed alarms for False Positives. At the 60th second, we launched the attack targeting an IP address within one of the 255 subnets that we monitor. Attack traces were injected between seconds 60 and 90. Packet digests were collected via the *Run Time API* offered by Netronome and used to calculate the detection accuracy. Note that each subnet is able to send a digest only once during a given epoch to avoid floods of digests that DoS the control plane. The exact number may be appropriately tuned. Accuracy in binary classification is defined as:

$$ACCURACY = \frac{TP + TN}{TP + TN + FP + FN} \qquad (4)$$

where *TP*, *TN*, *FP* and *FN* are defined for each subnet in any given 1s epoch:

- ***TP***: Number of True Positives i.e. digests received for a subnet when the subnet was the victim of an attack
- ***TN***: Number of True Negatives with no digests generated in non-attack cases
- ***FP***: Number of False Positives i.e. digests received for a subnet when the subnet was not the victim of any attack
- ***FN***: Number of False Negatives with no digests generated in attack cases.

The malicious traces were replayed at different rates to showcase the detection capabilities of the proposed mechanism. These correspond to three different attack scenarios: (i) an Underscaled attack, i.e. 10% of the reported Booter trace, (ii) the Booter trace as was originally reported and (iii) an Overscaled attack, comprised of 5 times the volume of the reported Booter trace. For all scenarios the benign traffic was injected as it was originally captured.

In the charts of Figure 4.4 we depict accuracy of our framework, evaluated using (4), according to the following values $\alpha = 0.004$, $k = 3$, $f = 0.15$ and $r = 2$, for two cases:

- *Two-feature case (F2)* that combines conditions (1), (2) corresponding to Flow Analysis features
- *Three-feature case (F3)* that also incorporates the Packet Symmetry feature based on condition (3)

For the Underscaled attack scenario, F2 performs slightly better than the F3. The former is more sensitive and thus able to identify attacks that generate small fluctuations on the number of flows. The latter due to the added traffic symmetry feature does not identify the attack in every epoch resulting in a greater number of *FN*s.

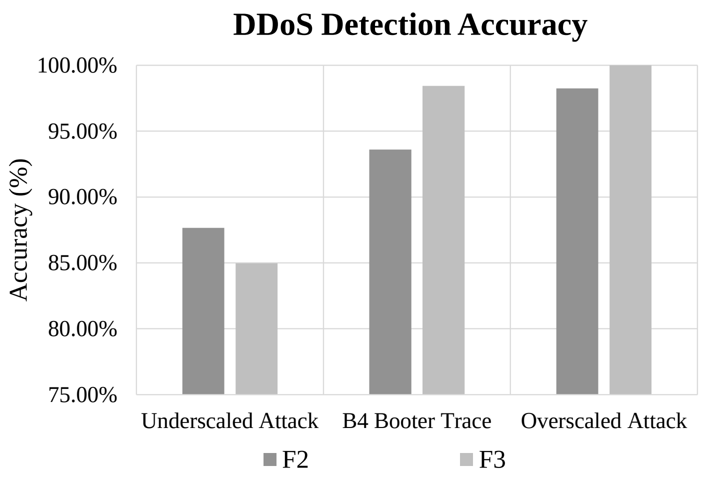


Figure 4.4 DDoS Detection Accuracy for different detection approaches and varying volumes

For the original Booter trace scenario, both approaches detect the victim, with F3 achieving higher detection accuracy as it has a reduced amount of *FP*s in comparison to F2. Finally, for the Overscaled Attack scenario *FN*s are eliminated due to the vast volume of the attack, achieving accuracy close to 100%. In general, using either two or three features (F2 or F3) we successfully detect ongoing attacks and identify the victim subnetwork within a single epoch.

4.5.3 Packet Processing Performance Capabilities

Additional stress tests were conducted to evaluate the processing capabilities of the Netronome cards. To that end, we synthesized traffic in various packet rates to (i) assess the performance capabilities of our pipeline and (ii) measure its impact on forwarding throughput. We use the same testbed setup but employ *pf-send* and *pf-receive* utilities of the *PF_RING* framework [53] on the sender and the receiver respectively.

In our experiments we considered the following use cases:

- *Plain forwarding case* whereby the target performs only switching (SW)
- *One-feature case (F1)* that incorporates anomaly identification based on Total Flow evaluation using condition (1) only
- *Two-feature case (F2)* that combines both Flow Analysis features based on conditions (1), (2)
- *Three-feature case (F3)* that also incorporates the Packet Symmetry feature based on condition (3)

Note that, the synthesized traffic we used does not bypass our DDoS detection pipeline, thus stressing to the limit the capabilities of the SmartNIC.

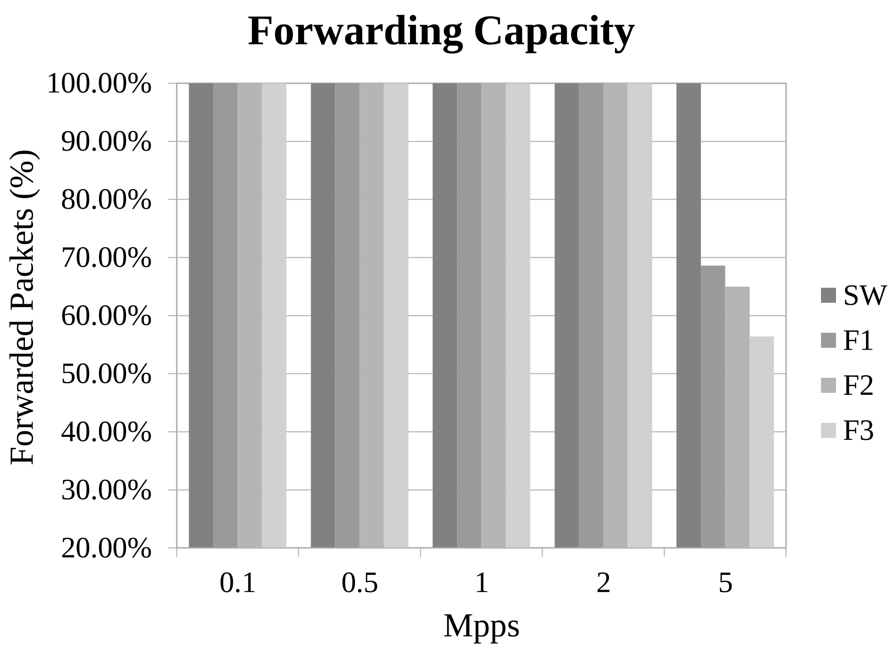


Figure 4.5 SmartNIC Forwarding Capacity

Figure 4.5 depicts the forwarding capacity of Netronome cards for various packet rates ranging from 0.1 to 5 Million packets per second (Mpps). The forwarding capacity is calculated as the fraction of traffic that successfully traverses the card.

Traffic rates of 0.1, 0.5, 1 and 2 Mpps show no performance degradation for all four cases. A higher traffic rate of 5 Mpps exhibits considerable degradation of the Netronome SmartNIC for adding the DDoS detection pipeline in cases F1, F2 and F3. These amounts to degradation between 35% to 45%. However, our detection pipeline is relevant in many enterprise and/or carrier networks since 10G links usually correspond to packet rates ranging between 1-2 Mpps [151].

The proposed DDoS detection schema heavily depends on accurate packet measurements through SmartNICs. To assess the impact of adding the DDoS detection pipeline, we further investigated the packet counting measurements available in the data plane via P4 registers. These were observed for all cases (SW, F1, F2 and F3) and attainable packet rates (from 0.1 to 5 Mpps), as depicted in Figure 4.6.

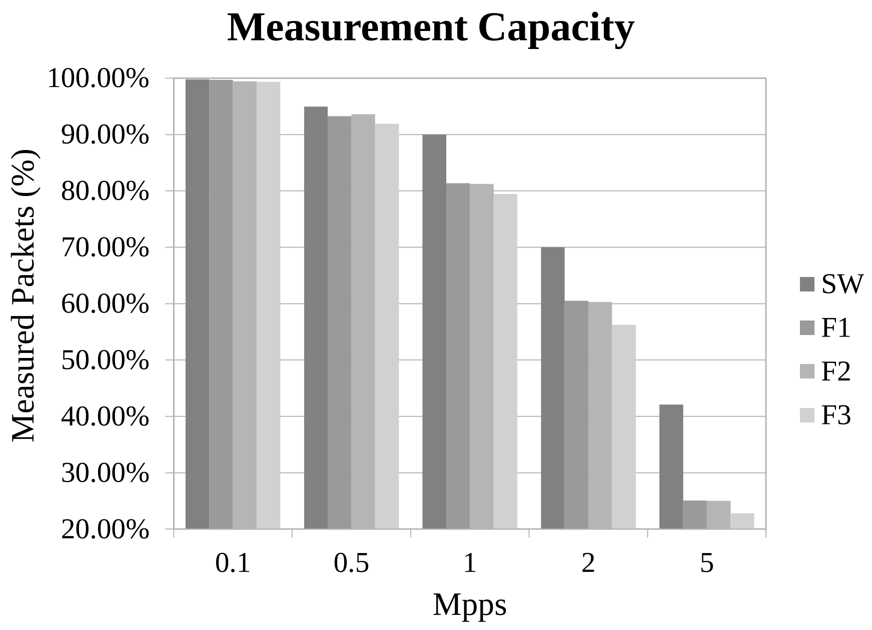


Figure 4.6 SmartNIC Measurement Capacity

For all cases even moderate packet rates of 0.5 Mpps start to exhibit degradation of measurement capabilities even in the simplistic “SW” scenario. Our DDoS pipeline successfully detects attacks with high accuracy despite measurement limitations of the SmartNICs. As also illustrated in Figure 4.5 packet forwarding is not degraded for rates up to 2 Mpps, a typical value for a fully utilized 10G link.

These measurement limitations are present only in P4 registers. We attribute this problem to simultaneous accesses of the memory areas used for packet counting. We have performed additional experiments using P4 counters and observed significant improvement. However, as mentioned in section 4.4, counters are only accessible from external controllers and thus of limited use for our efforts.

5 Placement and Automated Distribution of Access Control Rules to Heterogeneous environments

Sections 3 and 4 emphasized on collecting monitoring data for traffic analysis and anomaly detection. This section moves to the topic of attack mitigation, focusing on techniques to appropriately assign and subsequently deploy filtering rules via a unified abstraction layer.

5.1 Problem Statement

Modern DDoS attacks consist of multiple attack vectors, presenting a big challenge to network operators since traditional mitigation mechanisms struggle against such diverse and dynamic attacks. Important considerations for commercial DDoS solutions are: CAPEX/OPEX (procurement costs, licenses, support contracts), limited interoperability, privacy concerns and increased latency especially for cloud-based scrubbing. An analysis of various mitigation techniques for network anomalies is presented in section 2.6.2.

In this section, we propose an on-premise, distributed and non-proprietary mitigation schema. Our approach offers flexibility and cost-effectiveness by distributing access control rules over an existing enterprise network topology, consisting of diverse network nodes and operating at various protocol layers. Our framework leverages on the SDN paradigm that disassociates control-plane functionality from data-plane forwarding. Anomaly mitigation policies are implemented as separate northbound applications and employ diverse control/management plane mechanisms to communicate with the network substrate. Our motive is to enable network operators to appropriately mitigate network anomalies by enforcing custom security policies tailored to specific enterprise networks, while adhering to performance objectives and device-specific constraints.

5.2 Background and Related Work

A closely related approach is the Bohatei framework [152] that provides an elastic, NFV-based platform offering DDoS *Mitigation-as-a-Service* in ISP environments. Incoming traffic is inspected and according to the type and the scale of the attack, *Bohatei* determines and deploys appropriate mitigation resources (i.e. VMs). While not excluding virtualized middleboxes, we consider a more generalized approach that appropriately allocates generic mitigation rules to appropriate layers of an enterprise network for simultaneous attack vectors.

VNGuard [153] defines a high-level firewall policy for virtual networks configured within a cloud environment. Firewall instances are created and appropriate rules are placed within them based on an integer program formulation. Their objective is to minimize the number of virtual firewalls to be provisioned while respecting constraints on the number of rules, as specified by the cloud provider. Our work stems from a different perspective, notably the rule placement in existing attack mitigation resources (e.g. firewall instances) tailored to specific attack types based on capacity constraints and reward values guided by operator policies.

In [154] the *VGuard* DDoS mitigation mechanism is introduced. Traffic is classified according to the likelihood of malicious nature. Malignant flows are blocked, benign flows are routed to their destination as high priority traffic and suspicious flows are routed via low priority links. Another approach, *CoFence* [118], enables reciprocal sharing of compute and network resources to handle large volume DDoS attacks. Note that all approaches described above, heavily rely on a virtualized infrastructure to redirect and filter traffic. Related customized solutions for attack mitigation are further discussed in section 7. In this section we consider an agile mitigation schema for multi-vector attacks that blocks attack traffic at various stages of an enterprise network.

Cloudflare uses a similar system, Gatebot [155], that distributes mitigation rules across its PoPs around the world. While limited information is available on Gatebot’s internal specifics (e.g. anomaly detection algorithms) the XDP [4] is used as mitigation mechanism to perform advanced filtering of malicious traffic on commodity edge servers. Our approach is based on an integrated SDN framework to interface with distributed mitigation capable resources (either physical or virtual) via diverse/heterogeneous southbound protocols (not only *OpenFlow*).

5.3 Architectural Overview: Principles and Components

A high-level overview of the proposed framework, labeled “Orchestrator of Distributed Rule Placement” (*ODRP*) is depicted in Figure 5.1: Environment capabilities and constraints together with security events generated by anomaly detection mechanisms, e.g. Intrusion Detection System (*IDS*) are periodically collected by a *Pre-processor* (*PP*) component. This component correlates network security policies modeled as Event-Condition-Action (ECA) with security incidents, to formulate an optimization problem for distributing anomaly mitigation actions. In turn, this problem is assigned to the *Mitigation Resolver* (*MR*), a component tasked with computing a solution, i.e. an assignment of generic mitigation rules to network devices. These are converted to device-specific Access Control Rules (*ACRs*) and conveyed to the network infrastructure by the *Rule Handler* (*RH*) component through a range of supported southbound protocols.

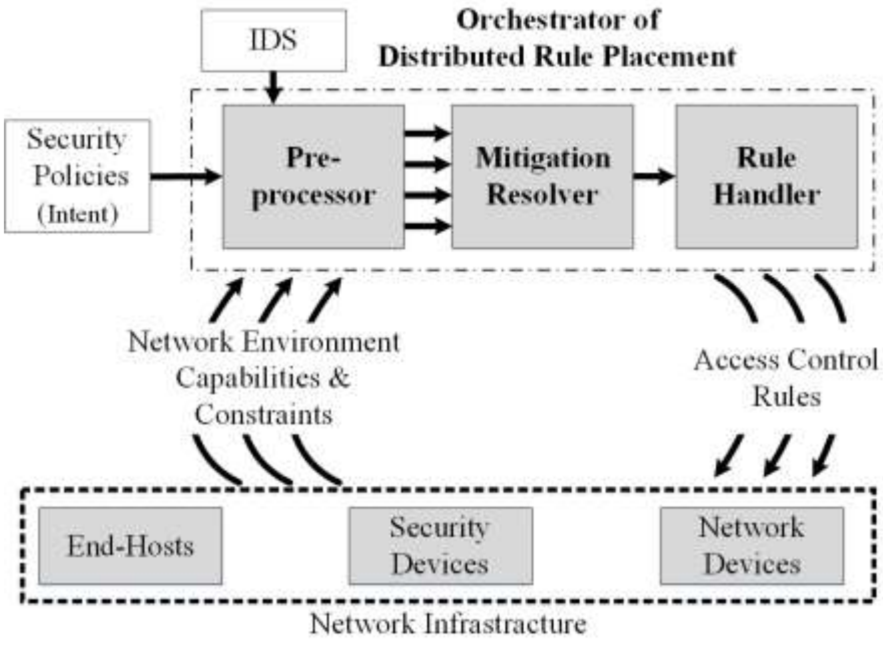


Figure 5.1 Operational Lifecycle of Orchestrator of Distributed Rule Placement - ODRP

Our proposed framework embodies the following principles:

- *High-level abstraction of Access Control Rules (ACRs)*: For typical heterogeneous multi-vendor environments, we create an abstraction layer for high-level mitigation actions (primitives). Our framework maps these generic actions to device-specific *ACRs* through standardized operations and protocols (e.g. *Blackhole Routing*, *BGP Flowspec*) or vendor-specific implementations. Thus, network operators may leverage on the capabilities offered by the existing infrastructure to mitigate network anomalies in a uniform manner.
- *Orchestration of mitigation resources driven by optimization considerations*: Our proposed approach assigns generic mitigation rules to relevant network devices in various hierarchical layers. This was abstracted as a Generalized

Assignment Problem (GAP) [156], whereby the process of assigning mitigation actions to network devices yields specific rewards. Reward values can be based on: (i) firewall capabilities of a specific network environment, (ii) policies for different network anomalies/attacks and (iii) the actual network attack characteristics. Based on the computed solution for the described GAP, attack traffic is blocked across the attack path at the most appropriate stage given a network architecture model (e.g. hierarchical). Note that in terms of the GAP complexity, there is a trade-off between input size (e.g. size of attackers) and solution time.

- *Adaptive mitigation of multi-vector network attacks*: Enterprise network infrastructures are commonly structured in architectural hierarchies essentially defining distinct defense stages (i.e. core routers/switches, distribution switches, access switches and hosts). Modern sophisticated attacks consist of multiple attack vectors (volumetric, protocol-based, application-based) typically targeting specific network/host resources. Our approach implements an automated network workflow (Figure 5.1) whereby security incidents trigger mitigation actions assigned to various stages across the attack path.

The 3 main components of ODRP are presented briefly below:

- *Pre-processor (PP)*: The purpose of this component is to properly formulate the GAP for a multi-vector attack tailored to a specific network environment. The Pre-processor component considers: (i) security events exported from an *IDS*, (ii) network security policies and (iii) environment-specific mitigation capabilities and constraints. Based on these, an appropriate input for GAP is structured and assigned to the *Mitigation Resolver*.
- *Mitigation Resolver (MR)*: This component (i) receives data to be fed as input to the GAP from the *PP*, (ii) attempts to reduce the input size of the algorithm to account for scalability issues, (iii) computes a solution for the GAP through a modular framework and (iv) exports the solution to the *Rule Handler*. The computed solution describes the generic mitigation rules to be distributed across the network elements.
- *Rule Handler (RH)*: This component is responsible for: (i) mapping the abstract mitigation rules to substrate-specific *ACRs* and (ii) distributing them to the network elements. An abstract mitigation rule may be defined as a typical

firewall rule with the following 6-tuple: (*source_ip*, *destination_ip*, *source_port*, *destination_port*, *protocol*, *action*). Different variations may exist due to device limitations; these variations contain at least the malicious source that should be blocked. Subsequently, the abstract mitigation rule is mapped to device-specific ACR employing common mitigation techniques (see section 2.6.2). The *RH* may employ both multi-protocol SDN controllers and automation frameworks to manage heterogeneous devices via a wide variety of supported southbound protocols (e.g. *OpenFlow*, *BGP*, *NETCONF*, *SSH*, *APIs*).

5.4 Detailed Architecture

In this section we elaborate upon implementation details pertaining to the architectural components and related sub-modules of the *ODRP*. An indicative setup for our proposed architecture is depicted in Figure 5.2 consisting of the components described in subsections 5.4.1, 5.4.2, and 5.4.3.

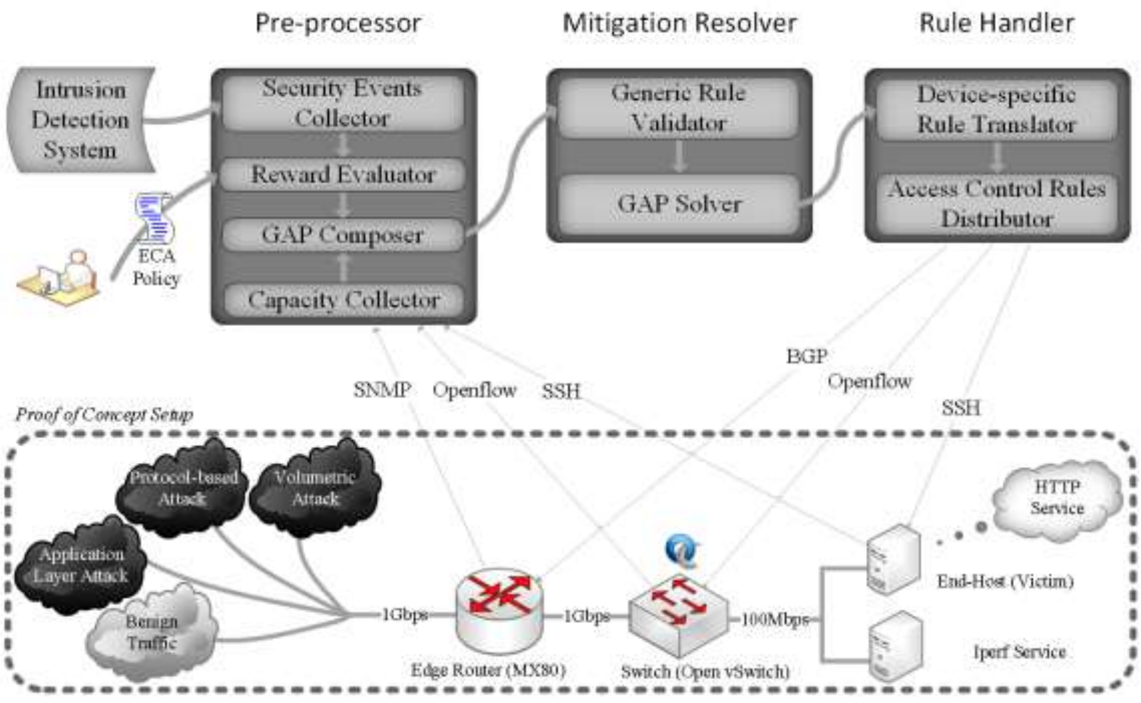


Figure 5.2 ODRP: Detailed Architecture

5.4.1 Pre-processor (PP)

This component consists of four modules: the *Security Events Collector*, the *Reward Evaluator*, the *GAP Composer* and the *Capacity Collector*. These correlate security events, management policies and network environment details to formulate the input of the GAP algorithm.

Security Events Collector: This module extends the event handling capabilities offered by *Ryu*. It receives alerts generated by an external *IDS* e.g. *Suricata* [96] and processes them to extract relevant information pertaining to a network anomaly. Specifically, the *Security Events Collector* identifies the exact type of the network anomaly and the network attributes required to create a mitigation action (e.g. source IP of the attacker). During a given time window (e.g. every 30 seconds), it generates structured data pertaining to observed alerts aggregated by the attack type and containing all assumed malicious source IPs. Finally, for scalability reasons, such information is conveyed periodically (every time window) to the *Reward Evaluator* module.

Reward Evaluator: This module receives the information from the *Security Events Collector* and in combination with operator-defined policies, maps anomaly types to a specific defense stage among Core, Distribution, Access and Host. We provide below a simplified model based on Event-Condition-Action (ECA) policies that enables network operators to express mitigation policy statements.

ECA Syntax of high-level security policy
Event = {Network Anomaly Alert}
Condition = {Attack type}
    Attack type = {Volumetric | Protocol | Application}
Action = {Block Core | Block Distribution | Block Access | Block Host}

The security policy above associates anomaly alerts triggered by the *IDS* with a generic mitigation rule conditioned to the attack type. This action selects the stage at which the specific type of attack should be mitigated as defined by the network operator. The mitigation action specified by the security policy is used to generate a reward array for each type of anomaly.

In order to appropriately mitigate multi-vector attacks the *Reward Evaluator* formulates a reward array based on the desired defense stage for each attack vector. For volumetric attacks, it is reasonable to select the most upstream core stage, whereas application layer attacks may be blocked at the downstream access or host stages. Interim cases such as protocol attacks may be handled in transit stages. The *Reward Evaluator* assigns the highest reward to the selected defense stage, while lower values are assigned to the remaining defense stages that may be activated in case the selected stage cannot accommodate all required generic mitigation rules. In case the selected defense stage is the transit distribution stage, we assign the highest value to this stage, a lower interim value to the upstream core stage and the lowest to the downstream access stage. This way, links of the access network, generally of smaller bandwidth than the upstream links, are better protected. Note that, exact reward values are not important, just their relative order.

Capacity Collector: The *Capacity Collector* module obtains the number of currently installed *ACRs* from routers, switches and end-hosts via *SNMP*, *OpenFlow* and *SSH* respectively, defining a residual capacity vector. The maximum capacities may be tuned based on device specifications and operational experience related to performance degradation per specific network device.

GAP Composer: This module receives the reward array along with generic mitigation rules, generated per attack type, for all malicious IP sources observed during the previous time window. In addition, the residual capacity vector is received from the *Capacity Collector*. It subsequently exports to the *Generic Rule Validator* (i) generic mitigation rules per attack type, (ii) the reward array and (iii) residual capacities of network devices.

5.4.2 Mitigation Resolver (MR)

MR receives the generic mitigation rules, computes a solution to the problem formulated as a Generalized Assignment Problem (GAP) and exports to the *Rule Handler*. It consists of two modules: (i) the *Generic Rule Validator* and (ii) the *GAP Solver*.

Generic Rule Validator: This module compares candidate generic mitigations rules with the ones previously computed and maintained within the module. These are filtered to isolate new malicious source IPs identified during the previous time window.

GAP Solver: The Generalized Assignment Problem assumes n items to be assigned to m bins. Assignment of item j to bin i yields a reward r_{ij} and carries a weight of w_{ij}. A feasible solution is an assignment in which for each bin i the total weight of assigned items is at most c_i (the capacity of bin i). The goal is to assign each item to exactly one bin in order to maximize the sum of rewards. In our case, *items* are the generic mitigation rules per malicious source IP and attack type, *bins* are the defense stages, *weights* are assumed equal to 1, *capacities* are the available resources of each defense stage and *rewards* are provided by the reward array. The problem can be formulated as an integer program:

$$maximize \sum_{i=1}^{m}\sum_{j=1}^{n} r_{ij}x_{ij} \quad (5)$$

$$subject\ to \sum_{j=1}^{n} w_{ij}x_{ij} \le C_i\,, i = 1, \dots ,m \quad (6)$$

$$\sum_{i=1}^{m} x_{ij} = 1\,, j = 1, \dots, n \quad (7)$$

$$x_{ij} \in \{0,1\}, i = 1, \dots ,m\ and\ j = 1, \dots ,n \quad (8)$$

As GAP is an NP-hard problem, in order to reduce its solution time, we decrease the size of input to the integer programming algorithm. As an example, we may apply various prefix aggregation techniques [157] on malicious IP sources. Additionally, generic mitigation rules can be further organized into groups of equal size *g*, equal to the minimum between all the residual capacities across the attack path. In case of an attack requiring a small number of generic mitigation rules (equivalent to a small number of malicious source IPs), the rules are treated as a single group. In the reduced problem, the integer program formulation is transformed with items x_{ij} corresponding to assignments of groups *j* to stages *i* and *n*, the maximum value of *j*, divided by the group size *g*. Weights w_{ij} are equal to the number of the rules they contain.

The reduced problem is solved via a *branch price and cut* [158] method implemented on *Dippy* [159], a framework for advanced integer programming problems. The computed solution consists of the assigned groups of access control rules to the defense stages of the attack path and is conveyed to the *RH* component.

5.4.3 Rule Handler (RH)

The *RH* component may be implemented using various approaches. We precisely use the *Ryu* SDN Controller that employs BGP, OpenFlow and Iptables (configured via SSH) to conduct all experiments presented in section 5.5. We also demonstrate a separate, more generic alternative in section 5.6, based on *Salt* and *NAPALM* that fits neatly into modern heterogeneous environments.

The role of the *RH* is to: (i) translate the *MR* solution to device-specific filtering capabilities, (ii) withdraw *ACRs* that are considered obsolete e.g. no attack reported from a specific source in two consecutive time windows (iii) avoid violation of the existing traffic flows and (iv) maintain and disseminate access control rules for each device. *RH* is configured appropriately to distribute mitigation via the available southbound protocols on the specific network environment. It consists of the *Device-specific Rule Translator* and the *ACR Distributor* modules.

Device-specific Rule Translator: This module maps groups of generic mitigation rules to each network device along the attack path. Specific rules, actual device and southbound protocol are conveyed to the *ACR Distributor*.

ACR Distributor: This module uploads device specific Access Control Rules (*ACRs*) through a corresponding southbound protocol specified by each mitigation technique. The proposed approach should override existing network management policies (e.g. already deployed *OpenFlow* rules, routing preferences, end host firewalls). Indicatively:

- *BGP*: our *ACRs* should match a predefined route policy which sets a high local preference, based on the IP address of the *BGP*-peer (*i.e.* the *BGP speaker* of *Ryu*) or rely on more specific (/32) announcements.
- *OpenFlow*: we use the optional fields for flow identification: cookie, cookie-mask and flow priority. Our *ACRs* (i.e. *OpenFlow* rules) are tagged with a specific cookie identifier and a specified value for high priority.
- *End-hosts*: we consider Linux-based machines and utilize the *iptables* firewall. A special-purpose *chain* labeled *ODRP* is used consisting of all the *ACRs* inserted by our framework. Their distribution on the end hosts is performed on top of the *SSH* protocol.

5.5 Evaluation

5.5.1 Experimental Setup

We implemented a proof-of-concept testbed deployed in our laboratory based on Ryu Controller. Offered functionalities were implemented as distinct customized applications modules. We incorporated in our testbed a Juniper MX80 router and an OF-enabled *OVS* [49]. End-hosts were implemented as Linux Containers (LXC), Virtual Machines and physical hosts. The *Ryu* Controller was customized to support *RTBH* over *BGP* on the MX80 router. Additional southbound interfaces were enabled on *Ryu*, notably *OpenFlow* for the *OVS* and *SSH* for Linux-based end-hosts.

The proof-of-concept testbed used to validate our proposed framework is illustrated in Figure 5.2, depicting an enterprise network section with a router placed in the upstream position, a switch as a transit device and Linux-based machines in the downstream end. Connectivity between these devices were selected to relatively represent typical hierarchical enterprise networks. Namely we assumed 1Gbps for upstream links and 100Mbps for the access network downstream.

5.5.2 Traffic Profiles for Anomaly Mitigation Experiments

The experimentation process considered the following types of traffic: (i) background traffic (benign) generated by the *iperf* [4] traffic generator emulating a dedicated TCP stream with bandwidth demand, (ii) benign HTTP requests via the *wrk*[5] benchmarking tool and (iii) artificially generated attack traffic via the *bonesi*[6]and *slowloris*[7]. Our artificially generated attack scenario was tuned by considering attack characteristics inferred from the Booter traffic traces [109].

To better highlight the potential benefits of mitigating the attack in distinct defense stages, we conducted an in-depth analysis of the B9 dataset, consisting of a high bandwidth CHARGEN attack [109]. Specifically, considering the amount of total bytes sent and total packets sent, we clustered the attack sources into 3 distinct groups via the *k-means classifier* [160], as shown in Figure 5.3.

Note that clustering is commonly suggested to classify network traffic [161] into benign and malicious traffic groups based on various network metrics such as quantity of bytes sent/received. Specifically, in [13] clustering based on malicious source IP prefixes was employed as a means to significantly reduce the number of flow entries.

From Figure 5.3 we considered the following mapping of attack types to groups:

[4] https://iperf.fr/
[5] https://github.com/giltene/wrk2
[6] https://github.com/Markus-Go/bonesi
[7] https://github.com/gkbrk/slowloris

- *Volumetric attacks* correspond to the *green* group exhibiting higher values of bytes per second (bps) and packets per second (pps).
- *Protocol-based attacks* correspond to the *red* group exhibiting interim values for bps and pps.
- *Application layer attacks* correspond to the *black* group, characterized by the lower values for bps and pps.

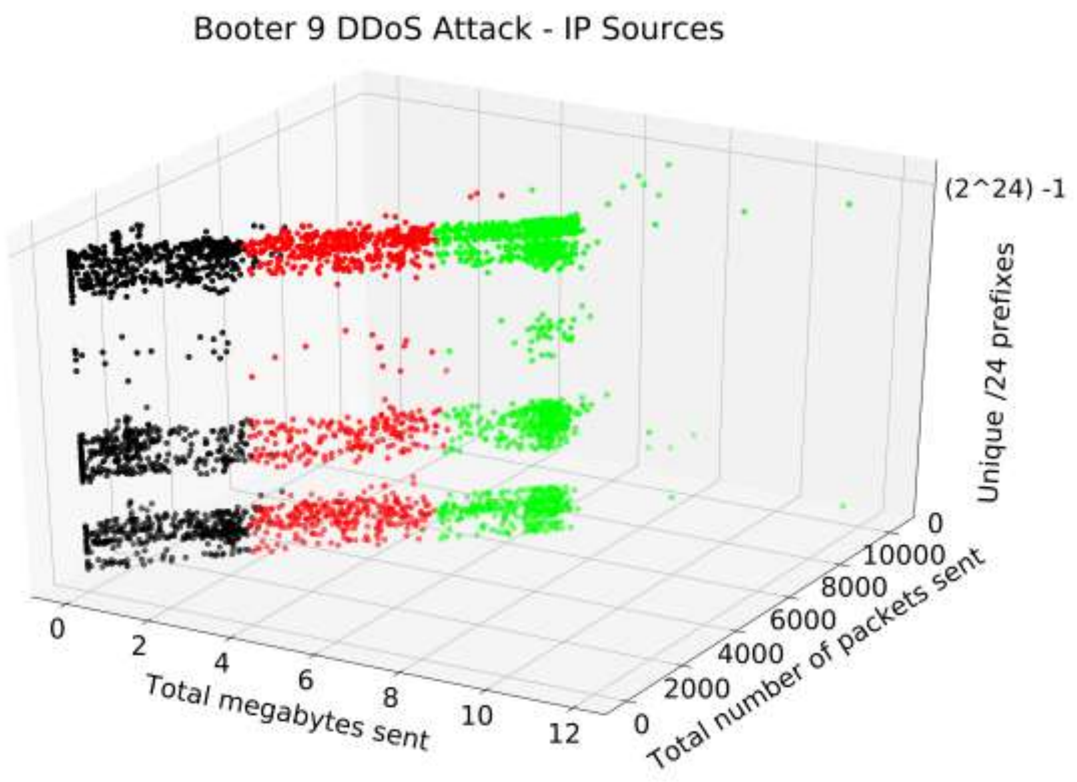


Figure 5.3 Malicious Source distribution (unique IPv4 /24 prefixes) clustered based on the total megabytes/packet sent (B9 dataset)

In our experiments, malicious attack traffic was generated via the *bonesi* attack simulator following the mapping above from 3,779 malicious IP sources. In particular, for application layer attacks we assumed malicious HTTP requests that reserve application level resources; this was emulated via the *Slowloris* attack tool from additional 1,500 different unique IP addresses. The resulting attack mix was combined to emulate a multi-vector attack scenario, in which the total amount of unique IP sources was 5,279 and the attack rate was 350 Mbps. Note that we downscaled the attack rate to this value due to the link size limitations and traffic generation constraints of our testbed.

Benign traffic was generated as 200 HTTP requests/sec to an Apache Web Server and a continuous 50 Mbps TCP stream via the *iperf* tool.

5.5.3 Experimental Evaluation of Anomaly Mitigation Mechanisms

Our experimental evaluation was conducted during 300 seconds per mitigation mechanism. During the first 30 seconds only benign traffic was present in the network; subsequently we launched the multi-vector attack targeting a victim host. We considered that malicious sources are detected within the next 30 seconds interval. Subsequently, mitigation countermeasures are instantiated (at the 60th second) and remain deployed for the duration of the experiment.

In order to evaluate our framework, we compared three different mitigation mechanisms: (i) *Single Firewall*, (ii) *Arbitrary Distribution* of *ACRs* and (iii) *ODRP* - Orchestrator of Distributed Rule Placement.

We considered that the *Single Firewall* mechanism is implemented with the *OVS* acting as a security middlebox [13]. We assumed that a moderate cost switch, emulated by *OVS*, can adequately support up to 4,000 flow entries.

For the distributed mitigation mechanisms (*Arbitrary Distribution* and *ODRP*), we considered that all network devices across the attack path (see Figure 5.2) may be used for the mitigation process. The device capacities are 2,000 routes for the router, 2,000 flow table entries for the *OpenFlow*-enabled switch and 1,500 rules for the *iptables* at the end-host firewall. Thus, all 5,279 malicious IP sources of subsection 5.5.2 can be blocked with an appropriate rule distribution.

The *Arbitrary Distribution* of *ACRs*, was modeled by rules assigned in a round robin fashion while respecting the capacity constraints of our devices. Our proposed schema, *ODRP*, translates operator policies into rule assignments to stages based on the attack type. The *ODRP* preference is to block volumetric attacks on the router, protocol-based attacks on the switch and application layer attacks on the end host.

In Figure 5.4 we illustrate the amount of attack traffic that is delivered to the victim while employing different mitigation approaches. The generated attack traffic is characterized by a bandwidth of 350 Mbps. However, only 94 Mbps actually reach the victim since the access link capacity in our testbed is 100 Mbps. While using the *Single Firewall* mitigation mechanism 62 Mbps of the attack traffic reaches the victim, since the installed 4,000 mitigation rules do not block all 5,279 malicious IP sources. The *Round-Robin* placement of rules performs slightly better, blocking all but 40 Mbps of attack traffic. Although this mechanism distributes mitigation rules to block all malicious sources, some were arbitrarily placed on the victim end-host. Thus, part of the attack reached the victim's access network. By contrast, *ODRP* distributes *ACRs* on the most appropriate defense stage, with volumetric attacks blocked early in the attack path. Hence, *ODRP* better protects the victim from malicious traffic.

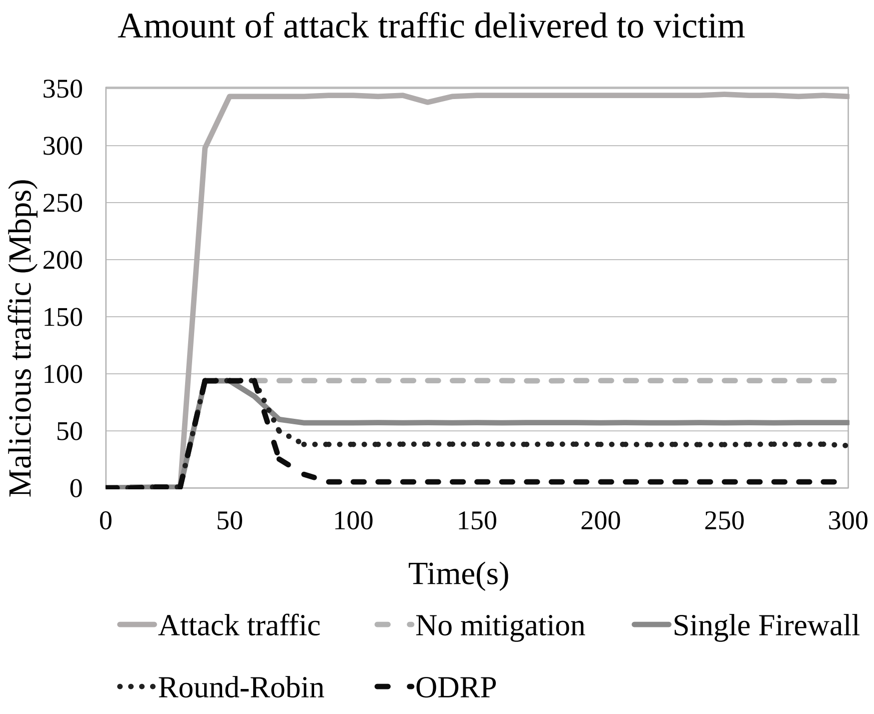


Figure 5.4 Total Attack traffic delivered to the victim

In Figure 5.5 we depict the impact of the attack on a benign TCP stream by plotting its throughput in various scenarios. The baseline stream value with no attack present is 50 Mbps. Unmitigated attacks result in TCP stream throughput dropping to almost 0 Mbps, due to congestion caused by the attack. The *Single Firewall* mechanism is able to block 4,000 of the malicious sources, thus the TCP stream is stabilized at 18 Mbps (36% from its baseline value). Placement of *ACRs* in an *Arbitrary Round-Robin* fashion improves performance with rates varying from 30 to 33 Mbps, (60% and 66% respectively). Finally, *ODRP* outperforms both previous mechanisms, since volumetric and protocol-based interference is blocked early in the attack path, thus preserving the bandwidth of the access link utilized by the TCP stream.

In Figure 5.6 we present an evaluation of the different mitigation mechanisms based on the percentage of successful HTTP transactions to a Web service running on the victim. Such a transaction was considered successful if completed within 1 second.

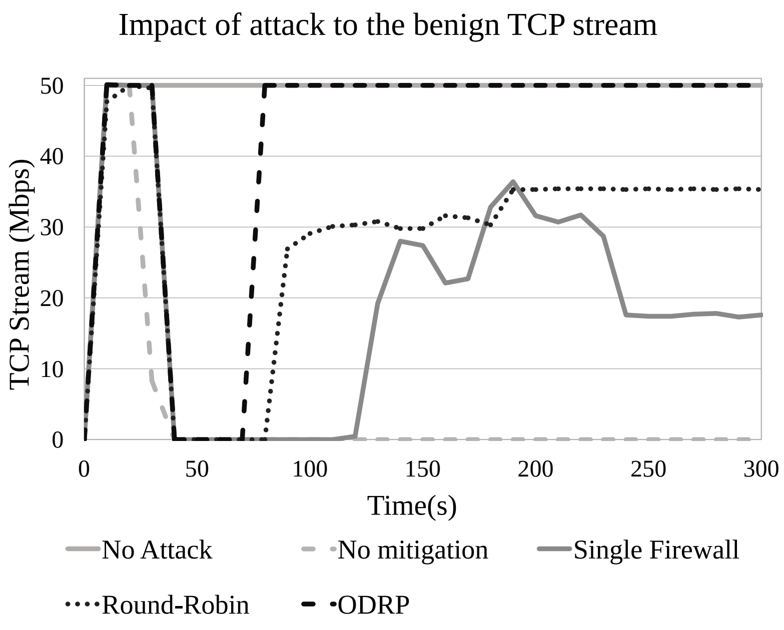


Figure 5.5 Benign traffic throughput (*iperf*)

Naturally with no attack injected, the percentage of successful HTTP transactions is 100%. Generated attacks had a two-fold impact: (i) HTTP Requests are dropped due to congestion and (ii) active sessions are consumed by maliciously crafted packets of *Slowloris* application-layer attack.

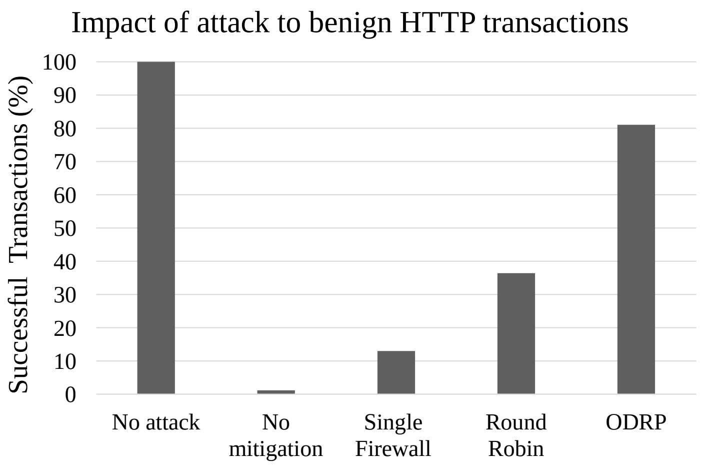


Figure 5.6 Attack Impact to benign HTTP transactions: Percentage of Successful HTTP transactions

Thus, unmitigated attacks exhibit a very low percentage of successful transactions, close to 1%. The amount of attackers blocked by the *Single Firewall* is not sufficient; unblocked attackers still manage to consume valuable bandwidth and server resources. The *Round-Robin* placement performs considerably better at 36% success. Finally, *ODRP* significantly outperforms both mechanisms reaching to 80% success. Note that in general, the effect of *Slowloris*-based attacks on the HTTP service persists even after blocking all malicious sources. This occurs because sessions reserved under false pretenses, stay open for a period of time even after a *Slowloris* attack is mitigated. Thus legitimate transactions attempted in the meantime might still fail. This affects every mechanism used in our experiment including our *ODRP* approach that could not exceed the 80% successful transaction rate.

5.5.4 Complexity of Generalized Assignment Problem

The efficiency of our *ODRP* approach depends on solving GAP in a fraction of a detection time window. As already mentioned, GAP is an NP-hard problem, thus it may present scaling issues according to its input size. To address them, we have implemented a customized solver whereby the generic mitigation rules (*items*) are split into groups and assigned to 4 defense stages (*bins*). We are presenting below the execution time of our solver considering indicative values for the number of generic mitigation rules and possible groupings.

Group Size	**Generic Mitigation Rules**		
	1000	5000	10000
1	0.5	2.74	5.72
10	0.09	0.25	0.56
50	0.008	0.05	0.1
100	0.004	0.03	0.05

Table 5.1 GAP Execution Time in seconds

The results in Table 5.1 above, demonstrate that the grouping technique we employed, reduces the execution time of GAP solution in reasonable values within a 30 second time window. Even the worst case scenario (10,000 generic mitigation rules in 10,000 groups) find a solution in 5.72 seconds. In a case that a large botnet is used to launch an attack e.g. 100,000 malicious sources require 100,000 generic mitigation rules to block the attack. With a group size of 10 items, the assignment problem would be solved in approximately 6 seconds. Thus, our proposed *ODRP* approach is expected to solve GAP in a fraction of a detection time window for attacks emanating from a large number of unique malicious sources.

5.6 Automated Rule Distribution via Salt & NAPALM

In this subsection we discuss a close parallel effort that focuses on vendor-agnostic network management for attack mitigation purposes. The alternative approach is based on the *Salt* automation framework [14] to distribute ACRs in a streamlined, device-agnostic manner. *Salt* may be considered as an even more generic, drop-in replacement of the Ryu SDN controller. In a nutshell, *Salt* follows an event-based architecture where minions (i.e. softwarized agents) communicate with the *master* to receive data (e.g. ACRs) and tasks for execution (e.g. enforce a new state). Usually devices cannot host a minion, thus specialized Proxy minions are employed that interface with network elements via a southbound driver. To that end we utilize the *NAPALM* Python library providing high-layer abstractions for device/vendor agnostic programmability.

Depicted in Figure 5.7 below, is the modified version of our testbed, managed by *Salt* and *NAPALM*. Indicatively we consider 3 defense stages: (i) a Juniper MX80-48T router, (ii) a Cisco IOS Catalyst 9300 multi-layer switch in L3 mode and (iii) a Linux server hosting various services. During a detected DDoS attack, the mitigation process deploys: (a) *firewall terms* for the MX80, (b) *access control lists* for the Catalyst 9300 and (c) *iptables rules* for the Linux server operating as the victim.

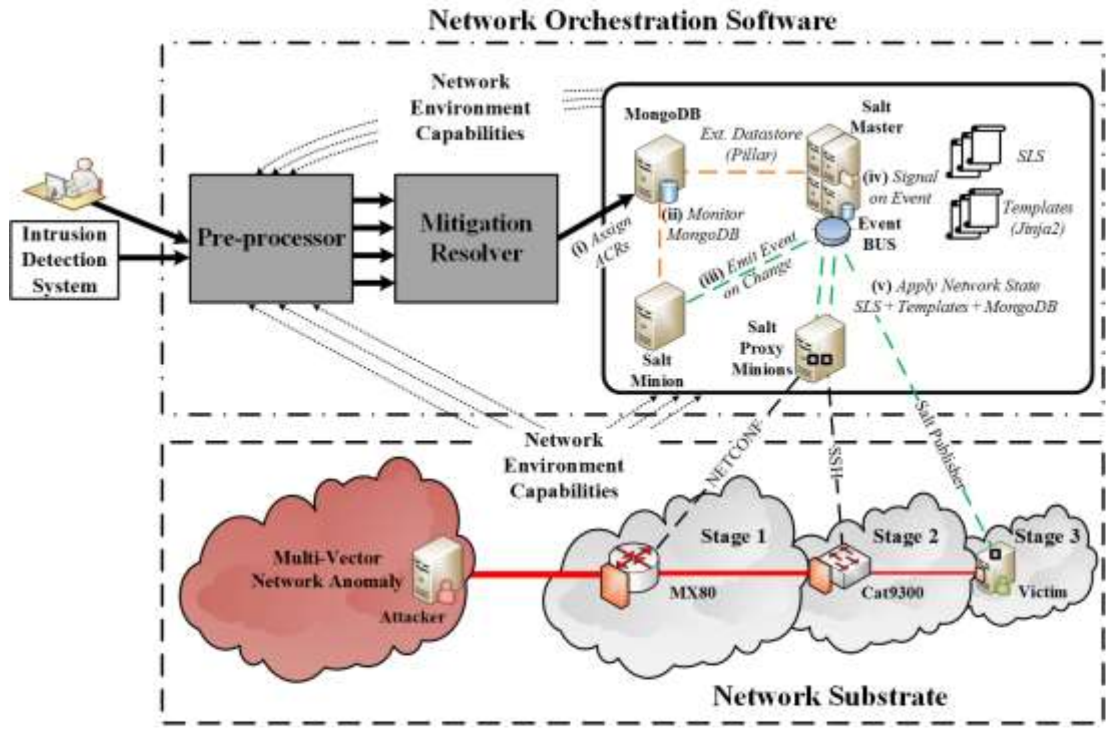


Figure 5.7 Rule Handler implemented via an Automation/Orchestration Framework

After the *MR* component computes a feasible solution that meets device constraints and operator preferences, it assigns ACRs to devices in different defense stages. As shown in the figure, ACRs are inserted in a MongoDB, as distinct *documents*. A *document* associates malicious sources with the device (network or host) on which ACRs should be deployed. Furthermore, the MongoDB is also used by *Salt* as an external *Pillar*, i.e. a data store that maintains *minion* specific data. Note that a device is directly mapped to a distinct *minion*. A separate *minion* continuously monitors the MongoDB and upon change triggers an appropriate event. This event is published in the *Salt* event bus and received by the *master*. In turn, the *master* instructs *minions* to apply the desired state (i.e. *ACRs* for each device) based on: (a) *Pillar* data i.e. malicious IPs, (b) *Jinja2* templates that describe *access control list*, *firewall* terms, or *iptables* rules and (c) *SLS* files defining the configuration rendering and distribution process.

Proxy minions hosted within a dedicated server use *NAPALM* to convey mitigation policies to network substrate devices via southbound protocols, i.e. NETCONF, SSH. In turn *NAPALM* employs the *eznc* library (NETCONF-based) for the MX device and the *netmiko* library (SSH-based) for the Catalyst 9300 accordingly to deploy rendered configuration commands. An indicative example for the rendering process is presented in Table 5.2 below.

Jinja2 template	**Rendered ACL**

ip access-list {{ **filter_name** }}
  {%- for **prefix** in **prefixes** %}
  deny ip host {{ prefix }} any
  {%- endfor %}
  permit ip any any
end
	ip access-list acl-malicious
	  deny ip host 1.2.3.4 any
	  deny ip host 5.6.7.8 any
	  deny ip host 9.10.11.12 any
	  permit ip any any
	end

pillar data	

prefixes = [
  “1.2.3.4”,
  “5.6.7.8”,
  “9.10.11.12”
]	

Table 5.2 Rendering Jinja2 templates into ACLs

6 DDoS mitigation via network provider collaborations

In section 5, we proposed a mechanism that appropriately assigns mitigation actions to existing on-premise devices based on operator preferences and device constraints. Massive attacks may be comprised of a considerably large number of sources or endanger important links. To that end, this section further extends the assignment of mitigation actions to an Interdomain schema for the collaborative mitigation of massive DDoS attacks.

6.1 Problem Statement

As a general observation, DDoS attacks are better pinpointed near the victim and more efficiently mitigated closer to their sources. Moreover, the sheer volume of present-day cyber threats may overwhelm an individual provider, thus the pressing need for collaborative mitigation efforts. However, defense collaborations might be hindered by operator concerns such as unwillingness to share victim-related information to preserve sensitive client data, lack of incentives for cooperation and shortcomings of incident handling mechanisms.

In this section, we propose an automated mechanism to orchestrate the collaborative mitigation of distributed attacks. Our schema fits best to Network and Wholesale Network Providers that share relevant and credible incident reports within a trusted federation. Network providers serve client organizations in their domains via interconnected Autonomous Systems (ASes). Our proposed solution enables a member of the trust federation under attack, directly or as the upstream provider, to (i) receive and process all available mitigation offerings pertaining to a particular attack scenario, (ii) identify the optimal sets of flows that should be mitigated based on costs and historical records i.e. reputation scores, and (iii) announce these sets by issuing Smart Contracts towards the appropriate mitigation collaborators.

The actual mitigation is undertaken by appropriately deployed filtering appliances within each collaborating domain. Similarly to our efforts described in section 5, this was modeled as a combinatorial optimization problem (Generalized Assignment Problem - GAP) aiming at distributing defense resources in multi-domain attack paths. We further introduce a reputation score that is calculated and maintained to evaluate the behavior of the respective collaborator on past incidents, while updating its credentials for future incident handling. We have also considered a verification mechanism, capable of maintaining and exposing necessary monitoring information for dispute settlement.

6.2 Background and Related Work

An analysis of Collaborative Schemas is provided in section 2.6.3. Summarizing notable mentions, [118] and [119] employ NFV and SDN techniques respectively. In both architectures mentioned above, DDoS mitigation is based in a reciprocal, *quid pro quo* schema. Approaches based on Distributed Ledger Technologies (DLTs) attempt to dampen such arguments against collaboration by enhancing, automating and tokenizing coordination efforts between collaborators [124]–[129].

Inspired by [119], [127], [128] we designed a modular collaborative, Blockchain-powered platform for mitigating massive DDoS attacks. Similarly to related efforts on collaborative DDoS defense [119] and [128], we adopted the Beta Reputation system and appropriately modified it to weigh partner contributions in the mitigation process (i.e. number of malicious sources blocked). Furthermore, other approaches [118], [119] employ scores that characterize a domain's past behavior to decide on how to allocate mitigation resources, or whether to provide assistance. Alternatively, given specific offerings from potential mitigators, we focus on enabling the victim to distribute mitigation actions (i.e. sources to be blocked) to the most reliable (e.g. higher reputation) AS/ASes. The distribution process is formulated as a cost optimization problem that considers the reputation of potential mitigators and may also include additional operational parameters and business policies.

Finally, disputed mitigation claims may be settled via a verification procedure relying on network monitoring data as provided by the victim and mitigator ASes; other mechanisms have been considered in [129] that require infrastructure access and specialized equipment for verification purposes. Note that we considered a trusted federated environment whereby collaborators follow agreed upon admission procedures and adhere to standards as in MANRS [117], a global initiative amongst network providers.

6.3 Overview and Baseline Design

6.3.1 Design Principles

A high-level overview of our schema is depicted in Figure 6.1. Malicious flows (continuous lines) that collectively form a DDoS attack, originate from - or transit through - interconnected federated ASes (dashed lines). Instances of the Collaborative Incident Response Manager (CIRM), deployed in each AS, are responsible for coordinating the mitigation of the malicious flows, and registering all interactions within two distinct data stores based on Distributed Ledger Technology. The entire process is orchestrated using data (logged messages, encrypted sensitive information) stored in a federation-wide Distributed Data Store Service (DDSS). CIRM also takes advantage of ad-hoc instances, realizing the respective Private Data Store Services (PDSS), between sets of ASes. These are used to support the exchange of additional (private and/or sensitive) information and to automatically settle the resulting agreements. Malicious flows are accurately detected by the victim or the Network Provider of the victim (*AS1*) via an independent detection mechanism (see sections 3, 4 and 2.4.3).

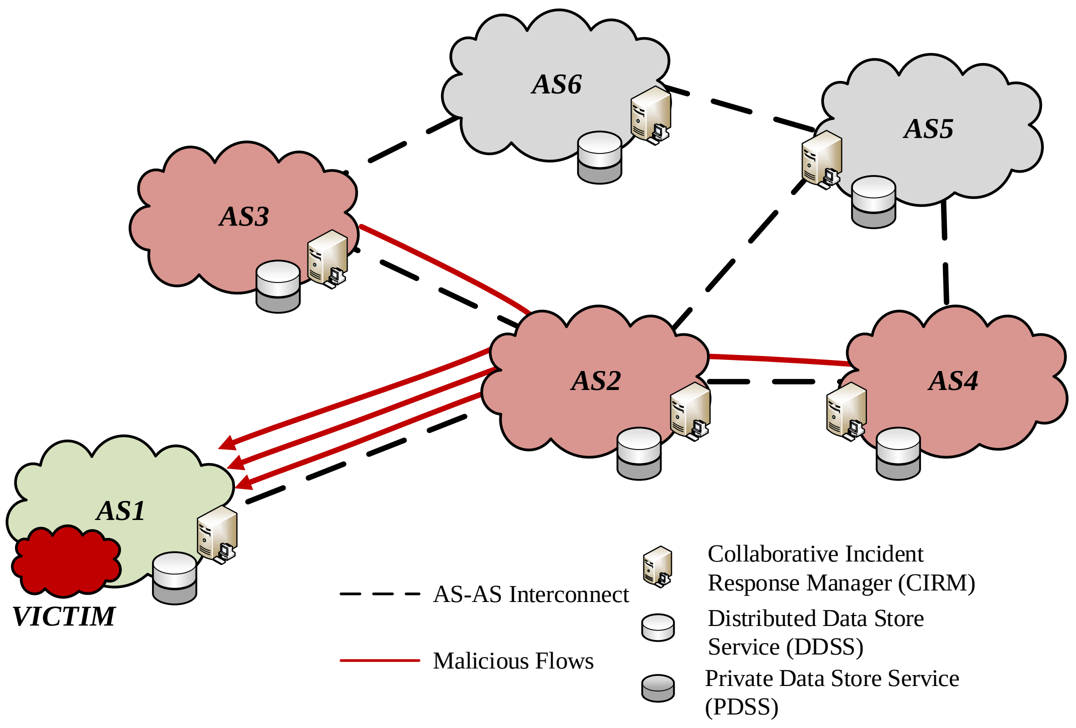

AS6
AS5
AS3
AS2
AS4
AS1
VICTIM
AS-AS Interconnect
Malicious Flows
Collaborative Incident Response Manager (CIRM)
Distributed Data Store Service (DDSS)
Private Data Store Service (PDSS)

Figure 6.1 High-level Overview of Collaborative DDoS Mitigation

Upon detecting a distributed attack, the *victim AS* (*vAS* henceforth) issues an SC for the incident; this digital agreement includes among others (i) the adjacent neighboring domains that forward attack traffic, (ii) a URL pointing to a document containing the DDoS attack sources stored within a distributed file storage system (e.g. Inter-Planetary File System – IPFS [162]) and (iii) the encrypted IP address(es) of the victim. ASes update the agreement above to include their adjacent ASes located in the attack path. Once the initial SC is issued, each AS located in the path of the attack is considered as a possible *mitigator AS* (*mAS* henceforth) and may offer via relevant SCs to fully or partially block malicious traffic. The *vAS* is then able to coordinate with the federated ASes leveraging the various types of SCs available in DDSS and PDSS. The resulting mitigation plan is obtained by feeding appropriate data to the cost optimization and reputation algorithms implemented as applications on top of each CIRM. The different types of SC implemented in our framework, as well as details on the related applications are further documented in section 6.4.

The design principles of our federated schema are:

- *Privacy-aware propagation of malicious network events and out-of-band communication*: Our schema enables a *vAS* to: (i) report a detected network attack respecting the privacy of sensitive information, and (ii) propagate the report to ASes along the attack path. We consider that ASes under attack potentially face adverse network conditions (e.g. link saturation), severely impairing communication. To that end, participants may employ some sort of dedicated secure channels for collaboration.
- *Collaborative mitigation of security incidents:* Federated partners are able to collaboratively deploy appropriate mitigation mechanisms. Our decentralized approach tends to push filtering rules near attack sources thus alleviating the victim domain from the total burden of a highly distributed attack. This schema refines commonly used blackholing techniques that may lead to blind service disruption.
- *Reputation-based Federation:* Our proof of concept implementation reflects a federated multi-domain network environment whereby partners are rated by reputation scores for past incident handling. The reputation score depicts the quality of the mitigation service provided, i.e. the capability of a collaborator to consistently block malicious flows.

- *Accountability and Consensus:* Collaboration schemas need to account for scenarios whereby partners may have diverging incident handling priorities or unintentionally report inaccurate information. As such, federation members should be able to: (i) log transactions, (ii) verify logged transactions, (iii) enforce Service Level Agreements (SLAs) and (iv) settle disputes via appropriate verification mechanisms.

Our vision is that such capabilities will introduce business workflows with potential financial benefits (rewards), thus incentivizing members of the federation to deliver high quality mitigation services.

6.3.2 Architectural Components

Figure 6.2 presents an overview of our proposed framework, as well as the interactions between the main components, namely (i) the CIRM and related applications built on top, (ii) the distributed ledgers realizing the Data Store Services (i.e. DDSS and PDSS), and (iii) the Attack Mitigation Appliance.

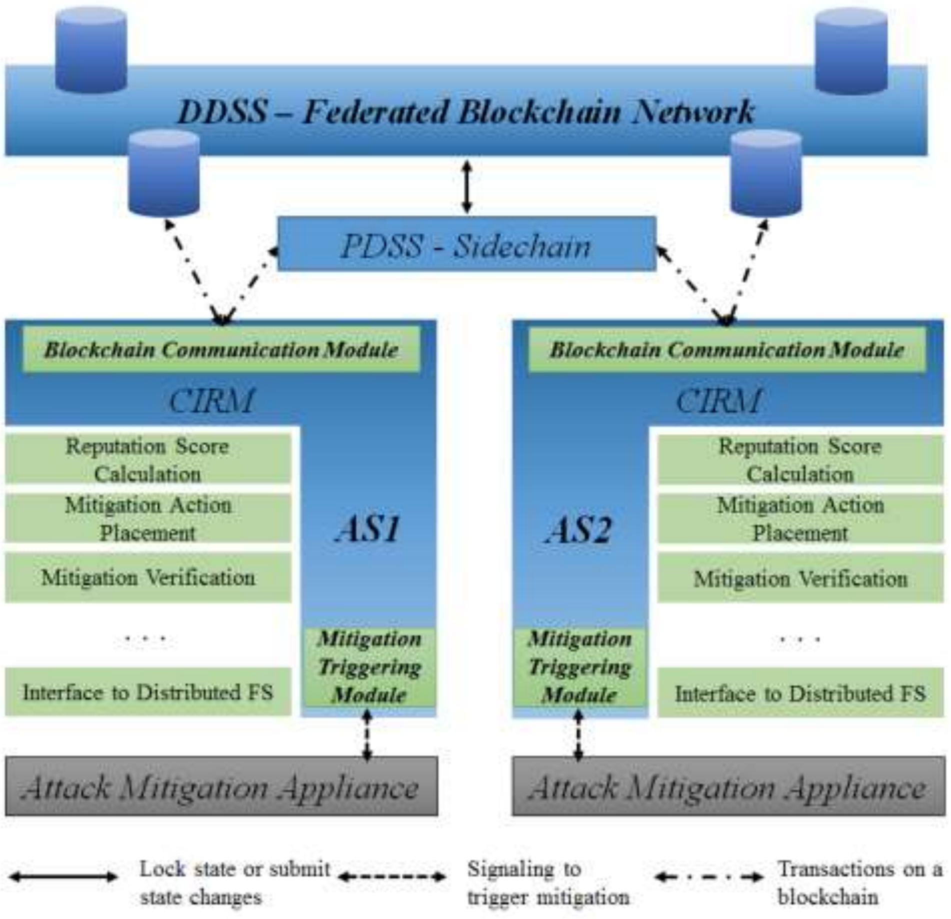

DDSS – Federated Blockchain Network
PDSS - Sidechain
Blockchain Communication Module
CIRM
Reputation Score Calculation
Mitigation Action Placement
Mitigation Verification
. . .
Interface to Distributed FS
AS1
Mitigation Triggering Module
Blockchain Communication Module
CIRM
Reputation Score Calculation
Mitigation Action Placement
Mitigation Verification
. . .
Interface to Distributed FS
AS2
Mitigation Triggering Module
Attack Mitigation Appliance
Attack Mitigation Appliance
Lock state or submit state changes
Signaling to trigger mitigation
Transactions on a blockchain

Figure 6.2 Collaborative Framework for DDoS Mitigation and Component Interactions

6.3.2.1 Collaborative Incident Response Manager

CIRM components are responsible for coordinating the mitigation effort within the federation. A CIRM is initially triggered by alerts, generated by DDoS detection schemas e.g. [96]. Our approach assumes that incident alert messages contain the following information: (i) a set of network flows related to the incident extracted via various mechanisms (e.g. sFlow/NetFlow) and (ii) the initial incident timestamp. CIRM then creates an Attack Information Document (AID) containing the incident flows, as well as its hashed digest (e.g. SHA-256) to be used for AID integrity check. This is stored off-chain, in a distributed file storage system (IPFS) accessible by all federation members.

The CIRM design is modular, consisting of two core elements, namely the Blockchain Communication Module (BCM) and the Mitigation Triggering Module (MTM). Specifically, BCM is used to register and perform transactions to the DDSS and the relevant PDSS instances in order to receive, acknowledge, or forward mitigation requests (as well as the related information) to other ASes. The communication between CIRM and the Distributed Data Stores is facilitated by deploying or updating instances of the various types of SCs we have implemented, as described in section 6.4. MTM is responsible for the communication with mitigation appliances upon the conclusion of transactions in DDSS and PDSS blockchain networks, triggered by a particular incident. In our implementation, MTM conveys malicious sources to an XDP-based mitigation appliance that implements the actual mitigation process.

Apart from the above core modules, the proposed framework relies on the following set of support applications, built on top of CIRM, that enable *vAS* to select the appropriate collaborators within the federated network and orchestrate the mitigation:

- *Reputation Score Calculation:* The cooperation track record of each AS in the federation is evaluated using SC attributes related to DDSS and PDSS past transactions. These are supplemented with monitoring metrics (e.g. verified blocked flows per incident) and dispute resolution results.
- *Mitigation Action Placement - Cost Optimization*: *vASes* are able to retrieve and evaluate offers posted in the PDSS for the mitigation of malicious sources related to a specific incident. The evaluation of offers is formulated as a

Generalized Assignment Problem based on the flows to be dropped and the respective reputation scores of potential collaborators (*mAS*).

- *Mitigation Verification*: This application facilitates business-oriented verification i.e. verify that the appropriate flows were actually blocked in our specific use-case, indicating whether the victim should consider the respective SC as fulfilled and consequently close it.
- *Interface to Distributed File Storage:* Large blocks of information are stored and retrieved from an external distributed file storage (e.g. IPFS) interworking with blockchain-based architectures to separately store files such as AIDs which may include hundreds or thousands of malicious sources. The CIRM application facilitates the communication with the distributed file storage system.

6.3.2.2 Data Store Service

Our architecture requires a form of distributed transactional data store available to all participants, accompanied by a consensus mechanism. As mentioned, the platform is not open to the public internet but only to approved members of the federation. Thus, there is no specific need to use complicated or computationally intensive consensus mechanisms (e.g. Proof-of-Work). We adopted a permissioned blockchain in which predefined miners (i.e. federated participants) are authenticated and authorized within the blockchain infrastructure. To that end, our implementation is based on a blockchain-based federation [163] using the Proof-of-Authority (PoA) [164] consensus mechanism, to fulfill the requirements mentioned above. Another objective is to preserve the victim's privacy (e.g. IP address) even within a trusted federation, while enabling on-demand reward and settlement of verified transactions. This is achieved via dedicated ad-hoc communication channels between collaborating ASes, forming private blockchain networks i.e. Sidechains [165]. Such Sidechains enable exchanging and updating of information related with an SC stored on the public blockchain network (Mainchain henceforth). Note that in our architecture, Mainchain realizes the DDSS component, while each Sidechain is a PDSS instance between two collaborating ASes. Recall that, both types of blockchain networks use dedicated out-of-band connections.

6.3.2.3 Attack Mitigation Appliance

The Attack Mitigation Appliance (AMA) represents an important component of the proposed architecture, responsible for the actual mitigation of malicious traffic. While any mitigation appliance could be an appropriate candidate, we implemented and deployed a flexible fine-grained and high performance mitigation mechanism based on XDP. Triggered by the MTM, this component is deployed and drops all IP sources described by MTM as malicious.

6.4 Proposed Architecture: Implementation Details

6.4.1 Blockchain-based Smart Contracts

The proposed approach leverages both public and private distributed ledger instances (i.e. *Mainchain* and *Sidechains*) as transactional distributed data stores. For our implementation we selected Ethereum [166] as the underlying platform of our distributed application, mostly due to the sizable resources and community surrounding the project. However, the proposed framework is not conceptually intertwined with a particular DLT.

Upon initialization, all federation members have a unique Ethereum address and a private key, authenticating them against their Ethereum account. Their public Ethereum address is explicitly configured within the Genesis block of the network (i.e. a JSON file containing static configuration parameters related to the network). Regarding the Mainchain, all federation members assume the role of a *Sealer* i.e. they can validate any network-generated block. Notably, in a Proof-of-Authority (PoA) deployment such as the one described for Mainchain, only accounts specified on the Genesis block as *Sealers*, are eligible for block verification. The Ethereum architecture also utilizes the *Bootnode*, that assists the respective BCM nodes of each member in discovering the DDSS network. Thus, each BCM can connect to the DDSS as an authorized member of the network. The PDSS is deployed following the same principles as the DDSS, with the only difference being that each PDSS is a PoA network between only two ASes, and only these two are configured as *Sealers*.

The business logic in both DDSS and PDSS networks is introduced via different archetypes of Smart Contracts (SCs) developed in *Solidity* Programming Language. Once an SC is deployed to the appropriate PoA network via the BCM, a unique address (hexadecimal number) is assigned to it. In order to interact with an SC, Ethereum defines an *Application Binary Interface* (*ABI*), as the data encoding scheme. Thus, SC data are encoded according to their type, based on the information described in the respective ABI. The ABI specification is shared among all federation participants upon initialization of a blockchain network, since, without this information the BCM modules would not be able to decode and interact with any deployed SCs.

We implemented four types of SCs: (i) $SC^{(A)}$ is issued (owned) by the victim AS and contains basic information about the incident; (ii) $SC^{(B)}$ is used to recursively notify potential mitigators; (iii) $SC^{(C)}$ is issued by transit ASes, offering to block malicious flows; (iv) $SC^{(D)}$ is issued by the victim AS to finalize collaborative mitigation assignments. All four SCs are used to orchestrate collective actions across an incident attack path.

In Table 6.1 we present specific attributes of the aforementioned SC types. Specifically, the current status attribute may be modified by the *vAS* and/or *mAS* as denoted in the table. In the last column, we indicate the specific component of the Data Store Services (DDSS or PDSS) that each SC type should be deployed on.

SC Type	Attributes	Short Description	Deployed
$SC^{(A)}$	Owner	Public Ethereum Address	DDSS
	Status	Current Status - Modification Rights ● Seek Assistance - *vAS* ● Attack Graph Update - *vAS*, *mAS* ● Bidding closed - *vAS* ● Under Mitigation - *vAS* ● Under Verification - *vAS* ● Fulfilled - *vAS* ● Not Fulfilled - *vAS*	
	vIP	Encrypted Victim IP Address	
	Graph	Attack Graph	
	AID_URL	IPFS URL to retrieve Attack Information Document (AID)	
	AID_Digest	AID Hashed Digest	
$SC^{(B)}$	SCA_Address	Address of relevant $SC^{(A)}$	PDSS
	Owner	Public Ethereum Address	
	Status	Current Status - Modification Rights ● Sent - *vAS* ● Acknowledged - *mAS*	

	M_Address	DDSS Address of a potential Mitigator	
	D_KEY	Decryption key	
SC$^{(C)}$	SCA_Address	Address of relevant SC$^{(A)}$	PDSS
	Owner	Public Ethereum Address	
	Status	Current Status - Modification Rights ● New bid - *mAS* ● Approved - *vAS* ● Rejected - *vAS*	
	SIP_URL	IPFS URL for the list of Source IP addresses to block	
	Reward	Requested Reward for Mitigation	
SC$^{(D)}$	SCA_Address	Address of relevant SC$^{(A)}$	DDSS
	Owner	Public Ethereum Address	
	Status	Current Status - Modification Rights ● Offer - *vAS* ● Fulfilled - *mAS* ● Verified- *vAS*	
	M_Address	DDSS Address of a potential Mitigator	
	Reward	Reward for Mitigation	

Table 6.1 Types of Smart Contracts

6.4.2 Orchestration Workflow

The steps described in this subsection constitute a recursive process between adjacent ASes within a federation-wide attack graph. This includes: (i) the notification mechanism of all ASes in the graph, and (ii) the propagation mechanism of decryption keys.

The Collaborative Incident Response Manager (CIRM) instance advertises the necessary information (e.g. malicious flows within *AID*, hiding privacy sensitive victim identification) to DDSS via an SC$^{(A)}$. Subsequently, the CIRM deploys SC$^{(B)}$ instances on the PDSS of each adjacent AS found in the path of the attack; these include the decryption key for the victim IP. Initially the decryption key is sent by the victim to adjacent nodes and then it is recursively propagated to all potential mitigators (i.e.

adjacent ASes that forward attack traffic). Each SC$^{(B)}$ is fulfilled upon acknowledgement by the adjacent AS.

As mentioned, federated members additionally maintain secure out-of-band communication channels to exchange signaling messages. Optionally, we could also employ asymmetric encryption (e.g. PKI, GPG) to encrypt and distribute the decryption key. Note that, the decryption key itself is kept secure within the infrastructure of a federated domain, using well-known and accepted security practices (e.g. hardened systems, firewalls, physical/logical access control and auditing).

ASes in the attack path should update the disclosed attack graph within SC$^{(A)}$, to include their relevant neighbors that forward malicious traffic. The attack graph is represented and stored as an array of linked lists. Each array record represents a given AS of the federation, while each list represents a valid branch of the attack graph. In addition to SC$^{(B)}$ related communication, each AS may retrieve blockchain-verified SC$^{(A)}$ instances via their respective BCM. Thus, by inspecting the attack graph an AS may decide on their role in forwarding ongoing attacks (i.e. whether they belong to the attack path).

ASes on the attack path may offer to assist (placing a bid) the *vAS* in the mitigation of the attack by deploying an SC$^{(C)}$. This SC maintains an attribute pointing to a list of source IP addresses that can be blocked by the bidding AS. The list of source IP addresses that can be blocked by the *mAS,* is also stored in the IPFS [162]. All SC$^{(C)}$ instances are stored within the PDSS; they may be retrieved by the CIRM and conveyed to the Mitigation Action Placement application to select appropriate bids.

The original SC$^{(A)}$ owner will create SC$^{(D)}$ contracts in the DDSS for all the approved SC$^{(C)}$ bids. Upon the creation of the relevant SC$^{(D)}$, each *mAS* should start the mitigation process and update the status attribute of the SC$^{(D)}$ to “Fulfilled”. All Fulfilled SC$^{(D)}$s should then be verified by the owner of SC$^{(A)}$ in order to complete the transaction allowing bidders to claim the agreed reward. Finally, once all the SC$^{(D)}$s of the initial SC$^{(A)}$ have been verified, the SC$^{(A)}$ is fulfilled and updated accordingly in the DDSS.

6.4.3 Reputation Schema for Collaborating Entities

Each AS is characterized by a reputation score, representing the quality of mitigation service provided to members of the federation in past incidents. The proposed reputation schema is based on the *Beta(a,b)* distribution [121]. The reputation value after n incidents is equal to the mean value of the beta distribution, $rep_n(AS) = a_n / (a_n + b_n)$, and represents a "forecast" for the outcome of future mitigation tasks. Such information is available to all ASes in the federation and may be used to appropriately assign mitigation tasks. Similar approaches [119], [128] that utilize Beta distribution consider only whether an AS provides mitigation service, disregarding the magnitude of its contribution. Hence, we extended the update process for the values of shape parameters, a_n and b_n, quantifying an AS's assistance, according to the following equations:

$$a_{n+1} = \gamma \cdot a_n + B_n \text{ , } b_{n+1} = \gamma \cdot b_n + N_n$$

The variables B_n and N_n represent the flows blocked and not blocked respectively, for a given instance n of an SC$^{(D)}$. Both values are based on the flows that a specific AS is assigned to block via SC$^{(D)}$ and not the total flows involved in the attack incident. Note that γ is a discount factor affecting past reputation scores; $\gamma = 1$ means that the reputation score incorporates the cumulative contribution of a contributor along the past n incidents, whereas $\gamma = 0$ indicates that only the last mitigation service offered may affect the reputation score. We used the former in our experiments.

In a typical scenario B_n should be equal to the length of the list of flows the respective SC$^{(D)}$ is pointing to. However, in case of disputes between a *mAS* and *vAS*, the two ASes calculate and compare the exact number of blocked sources for an ad-hoc bilateral settlement. This is further discussed in section 6.4.6.

6.4.4 Cost Optimization - Mitigation Action Assignment

We treat the aforementioned evaluation of available offerings to assist in the mitigation of an attack event as a Generalized Assignment Problem (GAP) as in section 5. From the *vAS* perspective, the GAP is considering the assignment of n items to m bins. Assignment of an item j to bin i yields a reward r_{ij} and carries a weight of w_{ij}. A feasible solution dictates the total weight of assigned items for each bin should not exceed the capacity of the bin (C_i) and each item should be assigned to exactly one bin. The optimal solution is an assignment maximizing the sum of all the rewards. In our case, each item represents the filtering of a malicious flow via a respective entry on a mitigation appliance with weight equal to one. Network providers within the described federation are represented by bins while their capacity denotes the available mitigation resources. Finally, the GAP reward r_{ij} is calculated as the product of reputation score of a collaborator i and the importance of the flow j, where the importance denotes the amount of bytes corresponding to the flow j. Then:

$$r_{ij} = rep(i) \cdot flow\_imp(j)$$

Intuitively, our approach tends to assign the most important flows, e.g. heavy hitters, to the collaborators with the higher reputation values. Consequently, the victim tends to receive better mitigation service for a specific incident. In turn, collaborators apart from any direct rewards as part of the agreement, establish and maintain a high reputation score increasing the possibility to be selected in the future.

GAP is an NP-hard problem and its input size affects the time needed to be solved. The input size of the algorithm can be reduced using grouping and/or prefix aggregation techniques (section 5). Such or similar methods are also applicable to the aforementioned formulation and can be used for reducing the execution time of the solver i.e. Dippy integer programming tools [159].

6.4.5 Implementation of Mitigation Mechanisms

As noted in previous sections, various mitigation mechanisms may be employed to block a set of malicious source IP addresses. Each *mAS* maintains one or more Attack Mitigation Appliances (AMAs) capable of filtering malicious sources. The Mitigation Trigger Module (MTM) module is responsible to communicate and configure an AMA via protocols including but not limited to: BGP, NETCONF, OpenFlow, SSH.

We implemented the AMA leveraging on the XDP capabilities that allow matching of specific packet fields in the filtering process. Specifically, our implementation parses packet headers, matches the source IP addresses of malicious actors coupled with the destination IP of the victim and drops it, whereas legitimate traffic is forwarded normally to its destination. Note that we could also utilize the capabilities offered by XDP to implement a DDoS mitigation solution that is not dependent only on IPs but combines other packet features, tailored to specific attack vectors (see section 7).

6.4.6 Verification of Mitigation Agreements

The proposed framework relies on the existence of a trusted federation, composed of ASes operating according to a predefined policy e.g. [117]. Hence, we assume that mishaps in contract fulfillment should not occur due to malicious intent but as a result of defense mechanism malfunction and/or misconfigurations. However, operating in this environment cannot completely alleviate the need for mitigation verification of $SC^{(D)}$s.

To that end, we consider an approach that leverages monitoring data collected from the peering (interconnection) links of the ASes. Specifically, each *mAS* should maintain flow information for the egress traffic directed towards the *vAS*. At the same time, *vAS* should maintain the respective data for ingress traffic on each corresponding link. Malicious flows pertaining to a particular incident are extracted from appropriate monitoring data and are stored in distinct timeframes i.e. 60 seconds); subsequently they can be retrieved upon request, for dispute settlement purposes. Note that flows are identified by their source IP; additional information may include destination IP, IP protocol type and TCP/UDP ports.

Moreover, aiming to tackle potential storage issues that might arise in cases of extremely distributed attacks, we opted for maintaining these malicious flows within a Bloom Filter probabilistic data structure [150], similar to the DNS approach in [167]. We based this selection on the observation that the actual counters are not required for the verification process. Hence, the malicious flows related to an attack incident are hashed in space-efficient Bloom Filters, each pertaining to a particular time-window of an incident. Indicatively, a Bloom Filter of 176 KBytes and 10 Hash Functions would be sufficient to accurately store 100,000 malicious sources (e.g. Memcached DDoS attack [168]), with a false positive probability of 0.1% according to [169].

Typically network operators employ widely adopted protocols for network monitoring, such as NetFlow or sFlow; these are usually combined with sampling mechanisms to alleviate load on network devices. We employed NetFlow monitoring data with varying sampling rates; since there is a trade-off between sampling rate and monitoring accuracy, there might be inconsistencies in the data collected that are not due to faulty mitigation but to visibility limitations of the sampling process. However, we do not expect the accuracy under massive attacks to be impacted by sampling [13], [94].

6.5 Evaluation

6.5.1 Experimental Setup

We implemented a proof-of-concept testbed for experimental evaluation deployed at the NETMODE laboratory of the National Technical University of Athens, Greece. We considered the following areas for experimentation: (i) Reputation Score Calculation, (ii) Mitigation Action Placement and (iii) Mitigation Verification.

Our testbed incorporates three Linux machines operating as distinct elements: (i) the Traffic Generator, (ii) the Mitigation Box (emulating the Attack Mitigation Appliance) and (iii) the Victim (*vAS*). As depicted in Figure 6.3, machines are connected in a star-like topology, with a Cisco Catalyst 9300 Multilayer Switch at the center. All the links operate at 10G.

The Traffic Generator is responsible for replaying and multiplexing benign and malicious network traffic, emulating traffic originating from *AS2*, *AS3* and *AS4* as depicted in Figure 6.3. The traffic is directed to the victim via the Mitigation Box with two bridged interfaces, each in a separate VLAN. Appropriate XDP programs, applied in the ingress interface of the Mitigation Box (VLAN B), are used to block malicious sources while allowing benign traffic. Verification points (*vAS*, *mAS*) are implemented as separate flow exporters on the NetFlow-enabled switch.

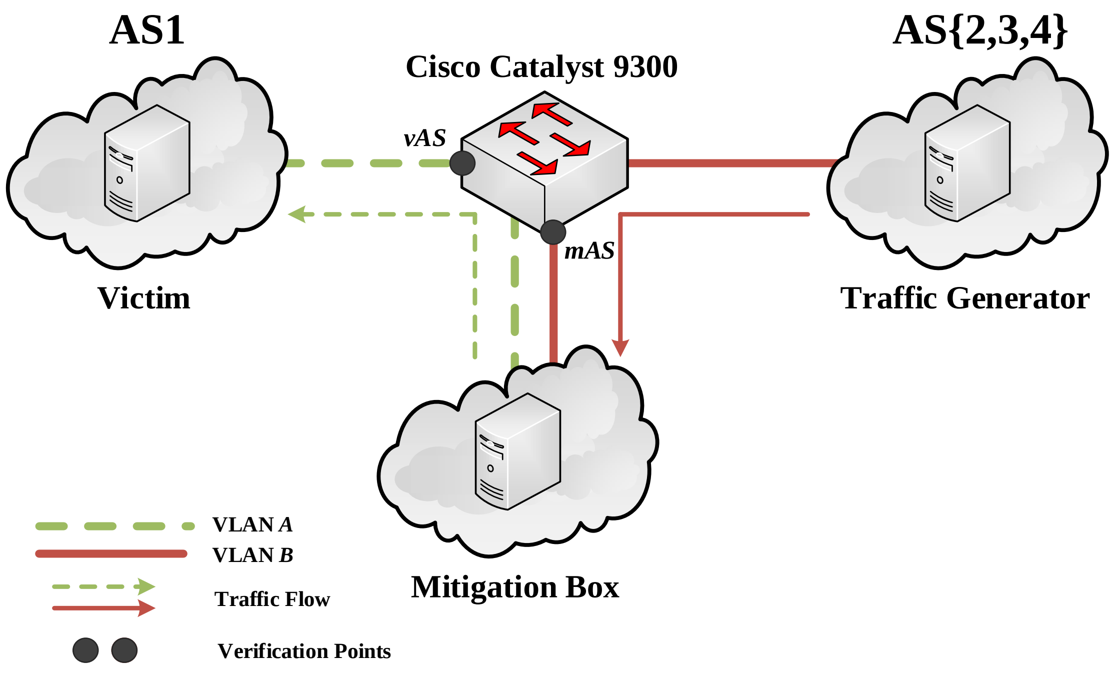


Figure 6.3 Proof of Concept Testbed Setup

In our experiments we used both benign and malicious traffic. The legitimate traffic is based on the *CAIDA Anonymized Internet Traces 2016* [107] dataset while the attack traffic utilizes the Booter traces [109]. We synthesized a total mixture of traffic that emulates a production network environment under attack; it has an approximately 1.77 ratio of malicious to benign traffic, with the total average packet rate approximately at 584 Kpps. In our experiments malicious flows are aggregated via their source IP attribute; we further assume that each AS blocks a flow (source) assigned to it with a probability equal to its reputation score since the latter ranks an AS mitigation service quality. We expect some incidents for which the mitigation fails to meet the promised requirements. As discussed in section 6.4, we considered that such incidents might occur due to human errors (misconfiguration of the security appliance) or hardware malfunction.

Moreover, aiming to evaluate the DLT-based orchestration of the proposed federated schema we deployed an Ethereum blockchain network consisting of six nodes, based on the *Go Ethereum* implementation [170]. Along with each node, we deployed a CIRM instance and a *Sealer* service, with the authority to validate each block on the blockchain network. Additionally, we deployed two more nodes for coordination and monitoring purposes: (i) *Bootnode* that facilitates the interconnectivity of the blockchain nodes by providing them with the necessary details to connect to the correct blockchain network; (ii) *Monitoring-node* that retrieves statistics and displays information about the current status of the blockchain network as in [171] and [172].

6.5.2 Reputation Score Calculation

To evaluate our proposed reputation mechanism, we selected three types of federated collaborators: (i) a *reliable* mitigator blocking a high percentage of the agreed malicious flows, (ii) an *unreliable* mitigator blocking a low percentage of the agreed malicious flows and (iii) an *unpredictable* mitigator, whose performance might vary both regarding the success rate and the volume of the promised flows. We considered two simulation experiments to assess the impact of mitigation service to the reputation score.

In the *first experiment*, we assume that a *reliable* mitigator AS (*mAS*) blocks the agreed malicious sources (i.e. 1000 flows) with a probability ranging from 90% to 100% for the first 500 incidents. For the next 500 incidents the *mAS* behaves as an *unreliable* mitigator, blocking the agreed sources with a probability ranging from 40% to 50%. We also consider the opposite i.e. an unreliable mitigator that turns reliable. In both cases, we calculate the reputation score and present its evolution during 1000 consecutive attack incidents.

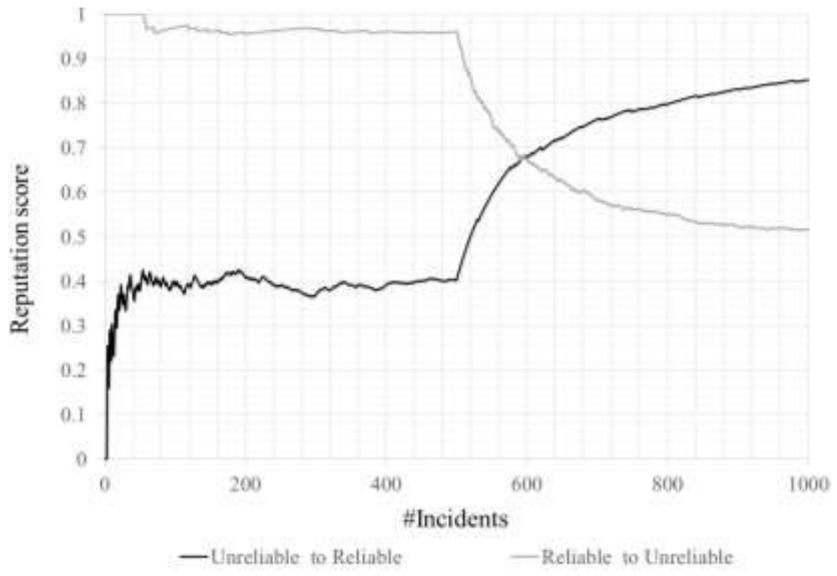


Figure 6.4 Reputation Score Evolution for Different Types of Federated Collaborators

As expected, Figure 6.4 indicates that the reputation score closely resembles the performance of the *mAS* during past attacks. Note that, the proposed schema allows collaborators to recover from low reputation scores in case they start providing high-quality mitigation services, while low-quality ones result to a significant drop in the reputation score. The rate of recovery depends not only on their reliability but also on the volume of promised flows. Thus, an AS is able to establish a high reputation score faster by seeking and fulfilling Smart Contracts for incidents involving large numbers of malicious sources.

In the *second experiment* we compare our proposed schema against approaches that consider only binary outcomes i.e. a collaborator is providing assistance or not [119], [128]. Specifically, we showcase the reputation score of the *unpredictable* collaborator mentioned above, that exhibits inferior performance in terms of (i) mitigation reliability and (ii) inability to fulfill excessive mitigation promises in a SC. The former indicates the probability that an attack will be mitigated; for simplicity we consider that the entire batch of agreed flows in a SC are either mitigated successfully or not. The latter defines a multiplier for the number of flows the collaborator promised but was not able to block. Specifically, in our simulation experiments we used 1000 flows as the base value.

In Figure 6.5 we compare the “binary” reputation schema (black surface) and our approach (gray surface) that considers the number of malicious flows promised but not blocked. Each point in the figure is the average reputation score calculated after 1000 consecutive DDoS attacks emulated in our testbed. The binary approach is only affected by the mitigation reliability and yields significantly higher reputation scores. Our approach also accounts for the actual number of mitigation actions successfully deployed by a collaborator and yields lower reputation scores, especially when the *mASes* fail to block large numbers of malicious flows (i.e. high factor of excessive promises). As an example a mitigation reliability of 0.9 and a factor of excessive promised flows of 10 means that a collaborator successfully mitigates an attack blocking 1000 flows with probability 0.9, while it fails to block excessive numbers of promised flows (10,000 flows) with probability 0.1. For this case, our schema would rank the *mAS* with a reputation score of 0.5, whereas “binary” reputation schemas would yield a reputation score of 0.9.

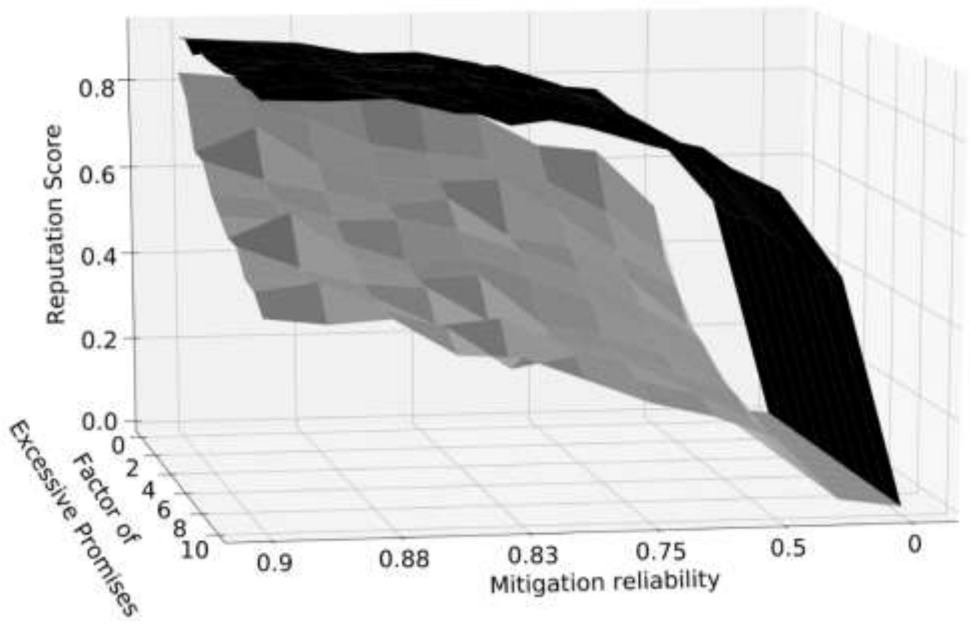


Figure 6.5 Reputation Score Comparison – Binary Reputation (Black), Proposed Approach (Gray)

6.5.3 Mitigation Actions Placement

To evaluate the mitigation actions placement on collaborating ASes we considered the topology depicted in Figure 6.1. We analyzed the first Booter dataset, B1, as reported in [109] and allocated all IP sources within distinct /24 subnets, distributed uniformly to

AS2, AS3 and *AS4*. This coupled with the actual attack dataset resulted to attack traffic emanating from the three ASes in proportion to 33%, 31% and 36% of the total attack traffic. Note that, attack traffic originating from *AS3* and *AS4* may be also blocked by *AS2*. In our experimentation process, the *mASes* offer to block the malicious traffic that flows through their network and the *vAS* assigns mitigation actions based on the aforementioned approaches. The cumulative capacity of all *mASes* is sufficient to collectively block the attack flows.

We consider two options for the mitigation actions placement using the following reward functions: $r_{ij}^{a} = rep(i)$ and $r_{ij}^{b} = rep(i) \cdot flow\_imp(j)$. Note that, in both cases all malicious flows are assigned to some *mASes* but possibly in a different manner: The former tends to assign mitigation actions to the *mASes* with the best reputation score. The latter additionally considers the importance of a flow for the mitigation process (i.e. flow importance). Flow importance is a metric identifying the danger a specific flow represents and may be calculated via sophisticated Intrusion Detection and Intrusion Prevention systems that evaluate multiple parameters across various attack vectors. This formulation enables operators to define policies focusing on traffic features they consider as the highest risks to their infrastructures. In our experiments we adopted the total amount of bytes that correspond to each malicious flow, considering typical volumetric DDoS attacks.

We showcase the mitigation of a real-time attack scenario, employing again the same traffic mixture of benign and malicious traffic. We consider four different mitigation approaches: (i) *vAS* assigns mitigation actions to all ASes based on: r_{ij}^{a}, (ii) *vAS* assigns mitigation actions to all ASes based on: r_{ij}^{b}, (iii) *vAS* assigns mitigation tasks only to *AS2* based on: r_{ij}^{b} (i.e. top 1000 sources) and (iv) *vAS* assigns mitigation tasks only to *AS2* based on: r_{ij}^{a}. The last approach is a random assignment, since all rewards r_{ij} are equal; we illustrate the worst case scenario in which the assigned 1000 malicious sources have the least contribution to the attack.

For approaches (i), (ii) we assumed *AS2, AS3* and *AS4* have a reputation equal to 0.9, 0.6 and 0.5 respectively. For (iii), (iv) the reputation score of *AS2* is assumed to be 1, thus blocks every flow assigned to it. The attack is detected and countermeasures are deployed after 30 seconds. This accounts for solving the Generalized Assignment Problem (GAP) and applying the mitigation actions to the XDP-enabled mitigation box.

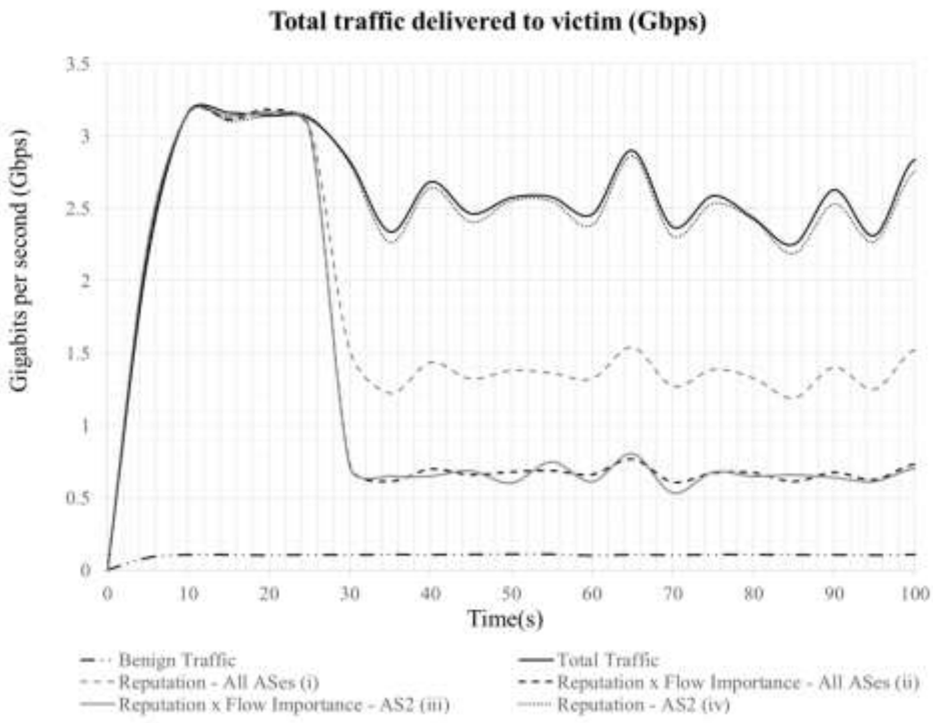


Figure 6.6 Total Malicious and Benign Traffic reaching the Victim

Figure 6.6 depicts the amount of traffic that is delivered to the victim. The attack traffic ranges between 2.5-3 Gbps. When multiple ASes collaborate, i.e. approaches (i) and (ii), the assignment of mitigation actions based on r_{ij}^{a} is significantly inferior to r_{ij}^{b} that incorporates flow importance. As shown in the figure approach (i) results to twice the volume of the malicious traffic reaching the victim, as opposed to (ii). Similarly, this is also illustrated in approaches (iii) and (iv), whereby only *AS2* is contributing to the attack mitigation. Assigning mitigation tasks in a random fashion as in approach (iv) might lead to an unsuccessful mitigation while considering flow importance in approach (iii) mitigates a larger volume of malicious traffic. Note that, both approaches (iii) and (ii) perform comparably well in terms of the total attack traffic reaching the victim. However, approach (iii) assumes that there exists a top notch strategically placed collaborator that is able to consistently block all malicious sources and has adequate capacity to sustain massive DDoS attacks. On the contrary approach (ii) relies on collaborative efforts and smart distribution of mitigation actions across multi-domain attack paths. Thus, we consider approach (ii) as the most appropriate for our collaborative schema.

6.5.4 Mitigation Verification

Additional experiments were conducted to evaluate the applicability of propabilistic data structures in the verification process. As mentioned in subsection 6.4.6, the verification relies on malicious flows as monitored by SC participants, both *vAS* and *mASes*. We used NetFlow data exported by separate flow exporters within the multilayer switch, sampled at two different rates: (i) 1 out of 100 and (ii) 1 out of 1000. NetFlow data are sent to a collector machine running two distinct processes of the *nfdump* toolset [173].

In Table 6.2, we represent the percentage of identified sources out of the agreed 1000, for successful and unsuccessful mitigation considering different sampling rates. The exact number for which assistance has been requested is 1000 (the top 1000 hitters in B1 dataset).

	Successful Mitigation		**Unsuccessful Mitigation**	
Experiment parameters	**Set**	**Bloom Filter**	**Set**	**Bloom Filter**
mAS – sampling 1/100 packets	0 sources	0 sources	88.86%	88.9%
vAS - sampling 1/100 packets	0 sources	0 sources	88.62%	88.74%
mAS - sampling 1/1000 packets	0 sources	0 sources	78.02%	78.1%
vAS - sampling 1/1000 packets	0 sources	0 sources	76.8%	76.88%

Table 6.2 Percentage of Malicious Sources Observed under varying Sampling Rates and Mitigation Performance

As expected, none of the 1000 sources are observed when mitigation is performed successfully, since the Bloom Filters only maintain the agreed upon malicious sources which in this case do not reach the victim and have no False Negatives. In the unsuccessful mitigation scenario, we expect most of the sources that were not mitigated to be hashed in the *vAS* Bloom Filters. However, packet sampling limits our *vAS* identification rates for non-mitigated malicious sources to 88.62% for a sampling rate of 1/100 and 76.8% for a sampling rate of 1/1000. In massive DDoS attacks though, we expect the identification of flows at the *vAS* to be closer to 100%, since the probability of malicious traffic being selected in the sampling process would be significantly higher. Note that the small discrepancies between the *vAS* and *mAS* identification rates are due to hardware limitations in our testbed (e.g. Netflow processing and cache size).

Apart from the sampling process, Bloom Filters also affect the accuracy of our identification process due to their inherent limitation of producing some false positives [150]. Thus, a flow might be identified as present (not blocked) although it has been successfully blocked. Note that false positive probabilities in Bloom Filters can be reduced to acceptable levels with very modest space consumption. Thus, we selected this option to fulfill our major concern, i.e. the verification of flows blocked by collaborating ASes. Specifically, in our experiments for 1000 distinct sources, we opted for a fairly small false positive of 1% that according to [169] is attainable with a Bloom Filter of 1.17 Kbytes and 7 hash functions.

7 Fine-Grained Traffic Classification and Attack Mitigation based on Programmable Data Planes

Previous sections investigated mechanisms for anomaly detection mechanisms (sections 3 and 4) and mitigation schemas that assign source-based filtering rules to on-premise devices (section 5) and external collaborators (section 6). Source-based approaches may raise cumbersome issues primarily in terms of scalability (i.e. volume of source IPs) and effectiveness (e.g. IP spoofing). In this section we build upon previous efforts and implement a two-level network architecture focusing on fine-grained traffic classification and customized attack mitigation for each vector based on programmable data planes.

7.1 Problem Statement

DDoS protection solutions often maintain statistics based partially (network flows) or entirely on IP sources in order to detect and ultimately mitigate malicious traffic. Source-based filtering requires significant memory resources which increase proportionally to the number of IPs. Moreover, IPs can be spoofed, further affecting source-based data collection and mitigation solutions [174]. Recent advances in data plane programmability enable customized solutions tailored to various network applications such as attack detection and mitigation.

To that end, this section introduces a two-level architecture based on P4 and XDP that (i) continuously monitors the network for suspicious traffic and (ii) reactively deploys softwarized appliances to further identify and filter malicious traffic. The first level uses the P4 approach presented in section 4 to implement a coarse-grained detection mechanism. The second level uses XDP to dynamically create portable appliances (middleboxes) tailored to the detected anomaly i.e. attack vector. These appliances collect monitoring data for classification purposes and ultimately filter packets with malicious characteristics. Packet classification is conducted via a supervised Machine Learning (ML) algorithm, i.e. *Random Forest*, appropriately trained with benign and malicious traffic focusing on distinct packet fields (features). Though not strictly limited to a particular attack vector, our approach emphasizes on reflection and amplification attacks that typically employ connectionless protocols over UDP (e.g. DNS and NTP).

7.2 Background and Related Work

There are various mechanisms in the literature related to DDoS protection. In general, these may (i) detect on-going attacks, (ii) identify the victim, (iii) segregate malicious from benign traffic and (iv) ultimately mitigate the attack. Such mechanisms for detection and mitigation were discussed in subsections 2.6.1 and 2.6.2 respectively. Summarizing the most closely related efforts [13], [64], [101]–[106], traffic features are extracted with OF signaling (stats requests and Packet-In messages) or via traditional monitoring methods (e.g. sFlow). Subsequently different techniques are applied ranging from statistical analysis, e.g. entropy, to supervised and unsupervised Machine Learning algorithms.

In comparison to similar approaches, we employ a DDoS detection schema offloaded in the data plane that provides rapid attack detection. Detected attacks are on-demand redirected for finer-grained processing. This approach significantly reduces processing and communication overhead of mechanisms used in similar approaches [13], [102]–[104] such as sFlow, OpenFlow and NetFlow. Additionally, most of the reported efforts in the literature employ metrics aggregated by IP addresses or network flows for traffic classification [13], [102]–[104]. In contrast, we focus on various packet features to classify traffic as malicious or benign in an IP agnostic fashion; the same packet fields are used to block the malicious traffic, without relying on aggregated/numerous IP-based filtering rules (ACL entries, OF rules) [13], [103]. All stages of our mechanisms are implemented in programmable hardware (P4) and software (XDP) data planes, realizing a dynamic, tunable yet high-performance detection and mitigation pipeline. XDP-based middleboxes may be easily deployed in COTS hardware in various points of a production network and scale horizontally to handle large amounts of traffic. Additional background information on XDP is available in subsection 2.3.

7.3 High-level Design

In Figure 7.1, we present a high-level overview of the proposed architecture for DDoS Detection and Mitigation, applicable either in transit provider networks or customer/edge network domains. This mechanism bundles together 4 separate components that offer: (i) Attack Detection & Identification, (ii) Fine-Grained monitoring, (iii) Traffic Classification based on Machine Learning (ML) algorithms and

(iv) Anomaly Mitigation. These are coordinated via a DDoS Mitigation Orchestrator component that plans the DDoS protection workflow.

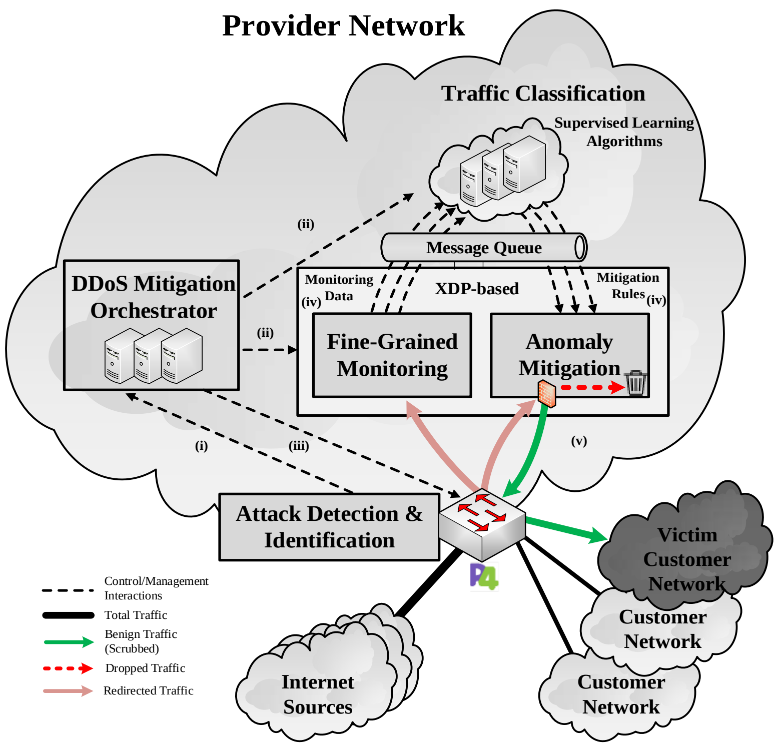


Figure 7.1 High-Level Overview of the DDoS Detection & Mitigation schema

Benign and malicious traffic originating from various Internet Sources traverses through or is destined to a network infrastructure equipped with programmable devices. Traffic metrics are extracted and analyzed to detect ongoing DDoS attacks targeting internal or downstream networks. Upon detecting an attack, appropriate alarms are generated for the victim network. This service is implemented and offloaded entirely in the data plane using P4-enabled devices. As an alternative, other solutions for DDoS detection can be incorporated [13], [64], [103].

Based on these alarms, the DDoS Mitigation Orchestrator spawns ephemeral components used for monitoring, classification and filtering. Subsequently, all traffic related to the attack vector protocol, is destined to the victim network is mirrored to the Fine-Grained Monitoring component for further analysis. Original traffic is redirected to the Anomaly Mitigation component (e.g. for DNS attacks we mirror/redirect all DNS traffic). The Fine-Grained Monitoring component employs high-performance programmable mechanisms (i.e. XDP) to extract appropriate packet fields depending on the attack vector used.

Extracted monitoring data are subsequently relayed to the Traffic Classification component, to be categorized as either benign or malicious. This component relies on classification methods based on supervised Machine Learning algorithms. These have been trained a priori both with attack traffic from commonly used attack vectors as well as benign traffic. According to the resulting classification, packets classified as malicious are used to create mitigation rules. These rules are in turn conveyed to the Anomaly Mitigation component that: (i) drops malicious packets and (ii) returns benign traffic back to the P4 device, in order to be forwarded appropriately. This component is also based on XDP.

DDoS attacks are mitigated focusing on distinct packet feature combinations (signatures) exhibited by offending traffic, ignoring altogether potentially spoofed source IPs. As mentioned these features should be engineered and optimized depending on the attack vector. Note that, the same features are used both for classification and mitigation purposes.

Packet field extraction, traffic classification and mitigation rule generation are performed continuously in distinct intervals (time windows). Selected intervals should be small (e.g. 10 seconds) to enable rapid propagation of information and ultimately prompt and accurate traffic scrubbing. After the corresponding components are spawned, operations are conducted independently in an event-driven fashion, via a Message Queue.

We opted to use COTS hardware (i.e. moderate-cost NICs) as programmable appliances powered by the XDP framework without compromising on packet processing performance. These can be instantiated on-demand and scaled according to traffic and application requirements, seamlessly integrated within NFV environments. However, the proposed schema consists of modular components that operate independently and can be replaced by other solutions of equal functionality. Netflow and sFlow can be used for traffic monitoring and packet analysis instead of P4 and XDP. Similarly, any traffic classification method may be used instead of the Random Forest algorithm.

7.4 Architectural Components and Implementation Details

Our Traffic Classification and Attack Mitigation schema consists of five distinct modular components: (a) Attack Detection & Identification (ADI), (b) DDoS Mitigation

Orchestrator (DMO), (c) Fine-Grained Monitoring (FGM), (d) Traffic Classification (TC) and (e) Anomaly Mitigation (AM). The ADI component is largely based on the work described in section 4. Some minor modifications were required to utilize P4 digests, i.e. alarms formatted as (*dst_network*, *ip_protocol*, *src_port*, *dst_port*), to discern a specific attack vector; indicative examples for reflection and amplification attacks may be found in [82].

The DMO coordinates the mitigation process for the identified anomaly, deploying countermeasures for the attack, based on predefined network policies. In case the attack traffic is not expected for the victim network e.g. Chargen traffic [82], [109], then this type of traffic is unilaterally blocked. These rules can be defined in a per-subnet/domain manner or even become part of standing policy. Alternatively, if benign traffic is expected for the same protocol as the attack e.g. DNS, then this type of traffic requires further analysis and refined scrubbing. To that end, DMO instantiates a sophisticated DDoS protection schema, realized by the Fine-Grained Monitoring (FGM), Traffic Classification (TC) and Anomaly Mitigation (AM) components. In turn, it notifies the P4-enabled device to redirect traffic related to the attack vector for the victim network to the FGM and AM components. Network traffic redirected (mirrored) to the FGM component is subjected to further packet analysis, while traffic redirected to the AM component is scrubbed and subsequently forwarded back to its destination.

Such workflows may be implemented by event-driven automation frameworks such as Saltstack (see sections 2.1.3.2 and 5.6). However, this work focuses on programmable traffic classification and attack mitigation delivered by FGM, TC and AM. Operations and related interactions are illustrated in Figure 7.2 below.

7.4.1 Fine-Grained Monitoring

FGM (i) receives the mirrored traffic, (ii) extracts appropriate packet fields (i.e. packet features) and (iii) conveys monitoring data to the TC component. Each FGM instance consists of the *Data Extractor* and the *Data Exporter* modules. The former is a kernel space XDP program that processes network packets to extract and store a set of preselected field values. As mentioned the exact packet fields heavily rely on the attack vector used; their number typically should be small for performance reasons but also to concisely isolate malicious characteristics. The combination of packet features can be represented by signature $X = [x_1, x_2, \ldots, x_n]$, where x_i represents packet field value i;

each unique signature X corresponds to a row in the Monitoring Data table of Figure 7.2. In the context of FGM, every observed packet signature corresponds to a counter stored within an appropriate BPF Map (i.e. hash table). The *Data Exporter* module is a user space program that periodically retrieves the contents (i.e. signatures) of the BPF map and compares them to the ones observed in the previous time window. Only new signatures are conveyed to the TC component.

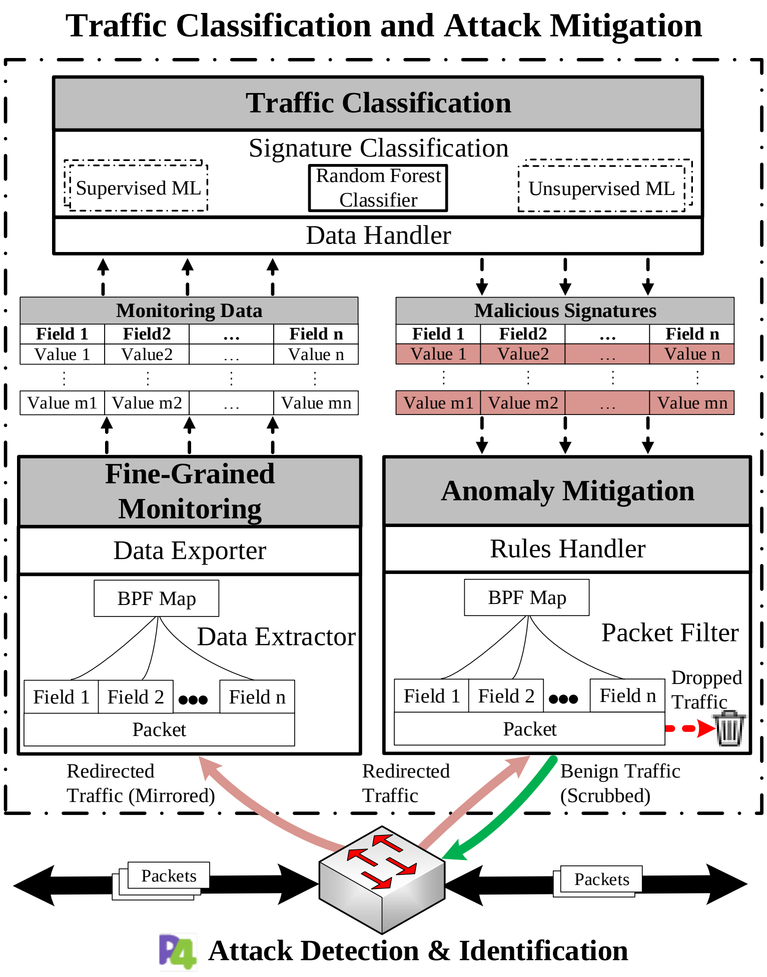


Figure 7.2 Fine-Grained Monitoring, Traffic Classification and Anomaly Mitigation interactions

Note that the FGM component could be implemented using any approach that allows access to packet fields (e.g. sFlow). We opted for XDP that provides high-performance monitoring capabilities and does not have potential limitations on the size of the extracted packet payload (e.g. sFlow) and by extent the available packet fields.

7.4.2 Traffic Classification

TC (i) collects monitoring data, (ii) feeds them as input to classification (supervised ML) methods and (iii) identifies malicious signatures. The *Data Handler* module collects the different signatures, X, relayed by the FGM component. In turn, the set of signatures is used as input to the *Signature Classification* module which characterizes them as benign or malicious. This module is preloaded with classification models, trained offline with malicious and benign traffic, both related to the specific attack vector (e.g. DNS attacks and benign DNS traffic). The trained models identify signatures that correspond to malicious traffic (see Figure 7.2) and convey them to the AM component via the *Data Handler* module. We experimented with the *Random Forest* classification algorithm [175], but our schema is decoupled from the classification method and can be extended with other algorithms. Moreover, for each attack vector we may employ the most appropriate classification mechanism for increased accuracy. Note that supervised classifiers rely on labeled training datasets. Alternatively, unsupervised learning e.g. outlier detection approaches [176], may also be employed when labeled datasets are not available.

Since packet features are used both to classify and filter packets, their selection requires specific understanding related to the protocol used by an attack vector. In addition, an arbitrary set of packet fields can be employed and gradually reduced using appropriate methods. Indicatively, we used such a method in an attempt to rank the most important features for classification of DNS packets. Note that the reduction of employed packet fields may enhance the performance of the XDP-enabled middleboxes and the ML models that reside in the TC component. Especially for the former, see related experiments in subsection 7.6.5.

7.4.3 Anomaly Mitigation

AM consists of the *Rule Handler* and the *Packet Filter* modules. The former receives a list of malicious signatures and installs them as filtering rules in a BPF map. The latter is an XDP kernel space program similar to the *Data Extractor* module of the FGM component. It parses and extracts the same set of predefined packet fields. These fields are subsequently compared to the filtering rules within the BPF Map. If the combination of packet fields (i.e. signature) of the received packet is contained in the BPF Map the packet is dropped (XDP_DROP). Otherwise, the packet is transmitted back (XDP_TX). This portion of traffic is considered benign (scrubbed) and thus normally forwarded to the victim network. While FGM can be implemented with various monitoring solutions (e.g. sFlow), the AM component is tightly coupled with programmable data planes solutions able to perform custom packet filtering based on any packet field (e.g. XDP).

7.5 Analysis of DNS-based Reflection and Amplification Attacks

We selected as a case study DNS reflection and amplification attacks, one of the most commonly used attack vectors for DDoS attacks. As in all similar attacks, malicious hosts spoof the IP address of the selected victim and send DNS requests mostly to vulnerable Open Resolvers or alternatively to authoritative servers. These requests are appropriately crafted to generate large DNS responses in an attempt to overwhelm the network capacity of the victims infrastructure. A common side effect of such attacks is the generation of fragmented packets, since large DNS responses exceed the Maximum Transmission Unit (MTU) of transit links. In our approach we consider that fragmented packets destined to the victim network should be blocked during a detected anomaly.

Packet fields (features) from the IP and DNS headers that may provide insightful information for classification are presented in Table 7.1 below; source IP addresses are excluded entirely, thus providing an IP agnostic classification and filtering mechanism.

Packet Fields	**Short Description**
IP_length	packet size in bytes
qdcount	number of entries in the question section
ancount	number of Resource Records (RRs) in the answer section
nscount	number of name server RRs in the authority records section
arcount	number of RRs in the additional records section
qr	specifies whether the message is a query (0) or a response (1)
qname	requested domain name; variable length field terminated by the zero length byte
qtype	type of the query (integer)

Table 7.1 Packet fields used in Traffic Classification of DNS volumetric attacks

The values of *IP_length*, *qdcount*, *ancount*, *nscount*, and *arcount* are numerical data and may be fed directly into ML algorithms. In contrast *qr*, *qtype* and *qname* may be considered as categorical variables, which can be handled via appropriate encoding techniques below.

As mentioned, both the FGM (*Data Extractor* module) and the AM (*Packet Filter* module) components need to parse and extract the aforementioned fields for each packet. All of the selected fields, except for *qname* and *qtype*, are fixed-placed and of fixed-length and thus can be parsed by XDP with relative ease. Specifically, the kernel space program receives each packet and extracts the *IP_length* from the IP header. It then extracts from the DNS header the *qdcount, ancount, nscount, arcount* and the *qr* values. Subsequently the *qname* value is required to be parsed; *qname* is a variable-length field (depends on the domain name size) with a maximum value of 255 characters (i.e. bytes) [177]. Hence, our program loops up to 255 times and breaks upon identifying the zero length byte (XDP supports only bounded loops). Note that *qname* is required to be parsed as it is used (i) both for classification purposes and (ii) *qtype* value extraction, that matches the two bytes following the *qname*.

After extraction, values are stored in or compared against the contents of BPF Maps. The memory space for storing each fixed-length field of a packet is specified in [177] (e.g. 2 bytes for *qdcount*). However, for storing the *qname* we would require 255 bytes, which would heavily increase the memory requirements of our program. Thus we opted to reduce it, using the *jhash* function implemented in Katran - Facebook's Load Balancer [2]. This implementation accepts up to 12 bytes as input (i.e. the first 12 characters of *qname*) and transforms it to a 4-byte integer. For the remainder of this section *qname*, refers to the hashed value of the DNS name. In addition to memory reduction, *qname* is encoded to a numerical value which can be directly fed in the *Signature Classification* module. This transformation affects domain names that share the first 12 characters or result in the same hash value (hash collision), as they are considered the same. However, packets with the same *qname* are not necessarily treated in the same manner by our schema, as additional features are employed both for classification and filtering purposes.

7.6 Evaluation

We evaluate our schema in an experimental testbed, employing real and synthetic network traces as detailed in subsection *7.6.1* below. In short, our experiments attempt to: (i) assess the detection accuracy of our mechanism, (ii) identify the most important features for attack classification, (iii) evaluate our signature-based filtering against source based mechanisms and (iv) demonstrate monitoring and filtering performance capabilities/limitations. These may be found accordingly in subsections *7.6.2*, *7.6.3*, *7.6.4* and *7.6.5*.

7.6.1 Experimental Setup and Datasets

We used a typical experimental setup as described in section 4; the setup was used to evaluate monitoring, classification as well as filtering capabilities. The FGM and AM components were implemented within the XDP framework. These were deployed on a physical machine (XDP-enabled node) equipped with a Netronome Agilio CX 2x10G SmartNIC [26]. For packet generation purposes, we used a VM equipped with an Intel X520 NIC 2x10G, able to generate packets at high rates using the PF_RING ZC framework [53]. The TC component was implemented using the *scikit-learn* Python library; it was deployed on a separate VM, with 12 vCPUs and 12GB RAM.

Real network traces were used to assess the detection accuracy of our schema, whereas synthesized traffic was used to stress test packet processing capabilities (monitoring and filtering). As benign traffic, we used traces: (i) from a 10G transit link between WIDE and DIX-IE (an experimental Internet Exchange), henceforth WIDE-G [108], (ii) from a 1G transit link between WIDE and an upstream provider, henceforth WIDE-F [108], and (iii) from Thapar University Campus Network, henceforth TU Campus [178]. As malicious traffic, we used seven of the *Booters* datasets (B_1, … B_7) [109]. These datasets contain different DNS-based reflection and amplification attacks generated by DDoS for hire services. All datasets apart from B_4 and B_5 contain *qtype ANY* DNS responses, a commonly used method for DNS reflection and amplification attacks that returns all RRs of all types for a given FQDN. In B_4 and B_5 attacks, the attackers attempted to use *type A* requests. B_4 contains multiple responses for a domain that contains a very large number of IP addresses. B_5 corresponds to a failed attack, where *type A* requests were used, in an unsuccessful attempt to generate large responses.

7.6.2 Accuracy of Signature-based Classification

In this subsection we evaluate the detection accuracy of the Traffic Classification (TC) component and specifically the *Signature Classification* module, using Random Forests (RF) with 10 decision trees. The training and testing were conducted separately for each of the following combinations:

- Each benign dataset (WIDE-G, WIDE-F, TU Campus)
- Each set A_i = {*Booters* - B_i}, where i = 1 … 7
  e.g. $A_4 = \{B_1, B_2, B_3, B_5, B_6, B_7\}$

The total different combinations are 21. Each trained model is evaluated against a mix of traffic (test dataset) based on the excluded attack dataset B_i and benign traffic from the same origin (e.g. WIDE-G) but from a different time period. In Table 7.2, we illustrate the *True Negative Rate* (*TNR*) of all combinations, which is the percentage of benign traffic that was classified as benign and the *True Positive Rate* (*TPR*), which is the percentage of attack traffic classified as malicious.

	Train: A1 Test: B1	Train: A2 Test: B2	Train: A3 Test: B3	Train: A4 Test: B4	Train: A5 Test: B5	Train: A6 Test: B6	Train: A7 Test: B7
WIDE-F (TNR)	99.99%	99.99%	99.99%	99.99%	100.00%	99.99%	99.99%
WIDE-F (TPR)	99.94%	100.00%	100.00%	**0.78%**	**0.14%**	99.99%	99.99%
WIDE-G (TNR)	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
WIDE-G (TPR)	100.00%	100.00%	100.00%	**0.31%**	**0.10%**	100.00%	99.99%
TUC (TNR)	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
TUC (TPR)	99.99%	100.00%	100.00%	**0.31%**	**0.05%**	100.00%	99.99%

Table 7.2 True Negative and True Positive Rates using *Booters* combined with benign datasets (WIDE-F, WIDE-G and TU Campus)

As illustrated in the table, RF is a consistent method to identify both benign (WIDE-F, WIDE-G, TU Campus) and attack traffic (*Booters*) patterns, provided it is trained with diverse attack data. However, RF is not able to recognize “unseen” (i.e. zero-day) attacks. This is clearly illustrated when the model is trained with A_4, which does not include B_4, while B_4 is used as the test dataset. Recall that B_4 contains large DNS responses with multiple *type A* RR for a *qname*; it differs from the training data (A_4), which contain attack traces with *type ANY* DNS responses. Similarly, *RF* classified all packets in B_5 as benign since it corresponds to a failed attack exhibiting similarities to benign traffic. Interestingly, during the experimentation process attack data were discovered within the benign datasets (WIDE-F, WIDE-G). A closer manual analysis of the original network traces revealed modest attack traffic, i.e. consecutive *type ANY* queries from specific IP sources to the same destination IP; these data were manually removed prior to the final experimentation process.

In summary, the proposed approach provides: (i) accurate detection of DNS volumetric attacks and (ii) robust identification of benign traffic for different, heterogeneous environments. We consider that the latter is attributed to the nature of DNS traffic, exhibiting specific traffic characteristics. This is also supported in the experimental results, which illustrate that RF classifiers achieve almost 100% *TNR* for traffic from different network environments, using the same features. To further emphasize this point, we also trained classifiers using data from one network environment (e.g. WIDE-

F) and used it to identify benign traffic in another network environment (e.g. WIDE-G). This was verified for all dataset combinations, thus demonstrating the generalization capability of our classifiers in common DNS traffic patterns. We expect similar behavior for other well-known protocols used for volumetric attacks e.g. NTP.

7.6.3 Feature Importance

As illustrated in the previous subsection, Random Forest (RF) classifiers yield accurate results (both *TPR* and *TNR* ~100%) provided they are trained with diverse attack traffic traces. To that end, we will showcase the most important features, as provided by the RF. This approach provides valuable insight into the classification process which can be used to eliminate inconsequential features. Ultimately, non-important features may be removed from the monitoring and filtering mechanism to significantly reduce packet processing time (XDP) and the required memory space (XDP and ML). The feature importance is quantified as a value between [0,1], where the highest values identify the most important features [175].

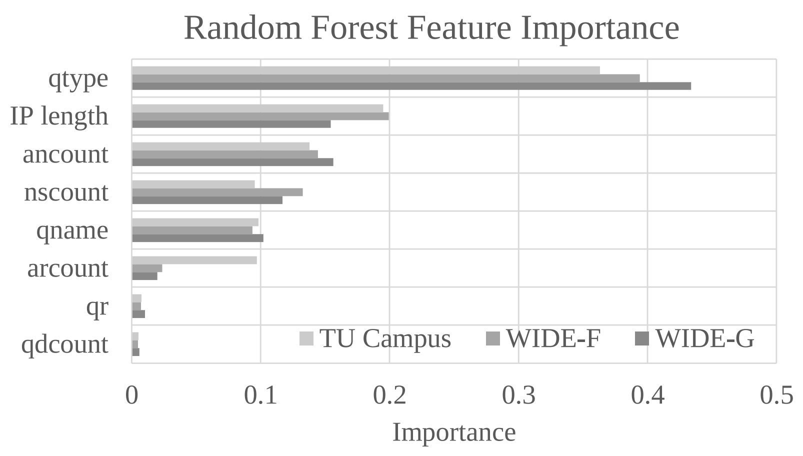


Figure 7.3 Feature Importance for DNS Traffic Classification provided by Random Forest

We trained 3 RF classifiers; each one uses all *Booters* datasets and one of the benign datasets (WIDE-F, WIDE-G, TU Campus) used in the previous subsection. In Figure 7.3 above, we depict the importance of each feature for the different combinations of datasets, as computed by the scikit-learn library. The reported values correspond to the average feature importance for multiple training iterations:

The dominant feature in all cases is the type of the query (*qtype*) since most attacks in the *Booters* dataset rely on DNS *type ANY* messages; as mentioned, this technique is commonly used to generate large volumes of malicious traffic. The length of the IP packet is the second most important feature; benign DNS traffic mainly consists of small packets while reflection and amplification attacks use large DNS responses. The latter are filled with a large number of (i) answers (*ancount*), (ii) authority (*nscount*) and (iii) additional (*arcount*) RRs. These numbers significantly increase in attack cases. Furthermore, attacks might use the same *qname* for generating large DNS packets, thus the *qname* hash may also impact the resulting classification (B_1, B_2, B_3: *root-servers.net*, B_6, B_7: *anonsc.com*). Low importance of the *qr* is expected since approximately half of the benign packets are DNS responses. Similarly, *qdcount* does not improve classification since all DNS packets in our datasets have *qdcount* =1.

7.6.4 IP-based vs Signature-based filtering

Most DDoS mitigation mechanisms reported in the literature use flow-based classification mechanisms, typically relying on IP sources to block attack traffic. In this subsection, we compare our IP agnostic, signature-based mechanism against a filtering approach that relies on IP sources. We extracted the total number of unique sources for each *Booter* dataset. Similarly, we extracted the signatures that characterize all the malicious traffic.

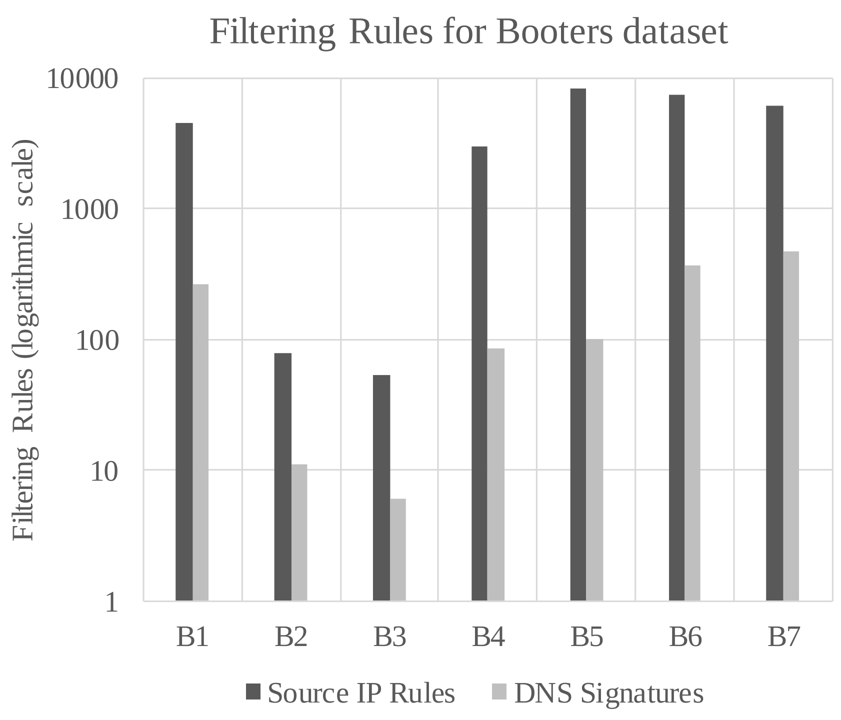


Figure 7.4 Comparison between Source IP and Signature-based filtering for *Booters* datasets

In Figure 7.4 above, we compare (in logarithmic scale) the number of the source IP filtering rules to the signatures that would be required to fully block the seven DNS attacks of the *Booters* datasets. Our approach significantly reduces the filtering rules required to mitigate the total attack traffic for every attack dataset. As illustrated, the number of the required rules is reduced considerably (exact values range between 86% and 99%). The benefits are twofold: (i) we do not rely on IP addresses that may be spoofed or change during an attack and (ii) we significantly reduce the memory consumed in the filtering process. Note that large memory utilization raises issues both in hardware (expensive TCAMs) and especially for our case in software data planes (lookup times are increased in large BPF maps).

7.6.5 Traffic Monitoring and Filtering Performance

In order to assess the monitoring and filtering performance of our schema, we isolated a window of reported network traffic and replayed it at high-speed rates (1 – 6 Mpps). Note that, a fully utilized 10G link typically corresponds to a packet rate of 2 Mpps. The traffic used for stress testing contains 100,000 unique IP sources and 10,000 unique combinations of packet fields (features). This proportion (10%) was selected based on the experiments of the previous subsection; the number of the DNS signatures for different datasets (i.e. *Booters*) is on average 10% of the number of source IPs.

Seven different XDP mechanisms were evaluated, four for monitoring and three for mitigation purposes: (i) COUNT_PKT that counts the number of received packets without extracting any fields; (ii) COUNT_IP counts packets per source-IP address; (iii) FGM (Fine-Grained Monitoring) maintains counters per combinations of all DNS features; (iv) FGM_LT, a streamlined version of FGM, that maintains counters per combinations of the five dominant DNS features; (v) DROP_IP that extracts the source IP address of each packet and compares against a blacklist of malicious sources (100,000 sources); (vi) AM (Anomaly Mitigation) that extracts DNS packet fields and uses a blacklist containing malicious signatures (all 10,000) to match and reject traffic; (vii) AM_LT that mitigates traffic using the 5 most dominant DNS features.

FGM_LT and AM_LT resulted from our experiments with Random Forest classifiers that demonstrated a clear distinction of DNS feature importance and led us to consider five features (*qtype*, *IP_length*, *ancount*, *nscount*, *qname*) out of eight total (see Figure 7.3 above). Mechanism (i) depicts XDP overhead to maintain monitoring statistics. Mechanism (ii) was implemented to compare our FGM (iii) and FGM_LT (iv) schemas to source-IP based anomaly detection alternatives. Accordingly, mechanism (v) was implemented for comparison against AM (vi) and AM_LT (vii).

In Table 7.3, we illustrate the percentage of packets processed by XDP compared to the total transmitted packets for various rates (in Million packets per second - Mpps). To avoid the additional overhead of counting the dropped packets in XDP, the processing performance of filtering mechanisms is calculated using the Netronome NIC counters based on the following:

$$dropped\_packets = dev\_rx\_pkts - dev\_rx\_discards$$

where dev_rx_pkts is the number of packets received by the NIC and $dev\_rx\_discards$ is the number of packets discarded without being handled by the XDP program.

Mpps	COUNT_PKT	COUNT_IP	FGM	FGM_LT	DROP_IP	AM	AM_LT
1	100.00%	100.00%	100%	100%	100.00%	100.00%	100.00%
2	100.00%	100.00%	100%	100%	100.00%	100.00%	100.00%
3	100.00%	99.99%	99.99%	100%	100.00%	100.00%	100.00%
4	100.00%	99.97%	99.96%	99.97%	99.98%	99.97%	99.98%
5	100.00%	91.29%	99.93%	99.95%	99.95%	99.93%	99.95%
6	100.00%	75.29%	85.73%	91.82%	85.56%	87.05%	92.86%

Table 7.3 Monitoring and Filtering Performance – Percentage of XDP Processed Packets

All monitoring approaches count almost all packets for values up to 4 Mpps. In higher rates, as was expected simplistic COUNT_PACKETS outperforms all other approaches. Although, FGM and FGM_LT process more packet fields than COUNT_IP, they have significantly better performance at 5 and 6 Mpps; similarly, AM and AM_LT outperform DROP_IP at 6 Mpps. This is attributed to the fact that FGM and FGM_LT have a decreased number of entries in the BPF Maps (i.e. 10,000), thus performing faster lookups. FGM_LT performs better than FGM as the former uses fewer packet fields than the latter and therefore executes fewer processing operations. An important observation is that the key size of a BPF Map does not impact the lookup time at all. Indicatively, COUNT_IP uses 32-bit keys whereas FGM and FGM_LT employ 160 and 96 bits respectively, however they all exhibit comparable lookup times (less than 200 ns). All observations above apply also for filtering experiments that in general perform better than their monitoring counterparts processing (i.e. dropping) packets for rates up to 5 Mpps nearly at line rate. We believe this occurs since retrieving values from the BPF Map is less intensive than updating them.

In summary, our monitoring and filtering mechanisms are suitable for high-speed modern network environments. Moreover, signature based approaches exhibit better or similar performance than source-IP based alternatives. Lighter versions of our mechanisms, FGM_LT and AM_LT, perform even better.

8 Conclusions and Future Research

8.1 Summary and Concluding Remarks

This dissertation investigated large-scale network attacks and specifically solutions related to various aspects of DDoS protection such as network data collection, anomaly detection and mitigation. Recent technological advances were combined to create automated services for network security, applicable to modern SDN environments and legacy network infrastructures.

Sections 3 and 4 addressed the collection and processing of monitoring data, primarily for anomaly detection purposes. **Section 3** focused on a monitoring schema for the collection and processing of network data exported from dispersed vantage points (i.e. devices) in an effort to enhance visibility into anomalous events. This approach is based on the observation that different types of attacks are aggregated at central nodes, while others exhibit localized characteristics better observed near the edge. The advent of programmable network hardware and related softwarized appliances can be used to further extend this concept in two ways: (i) middleboxes can be deployed on-demand along a network path as dynamic VNFs for various purposes (e.g. monitoring and anomaly detection) and more importantly (ii) processing tasks may be offloaded to network devices in a distributed manner.

Thus, **section 4** presented a DDoS detection schema implemented entirely in P4-enabled devices. In-network computing is an appealing concept, but potential solutions need to account for memory usage and processing limitations to keep up with requirements for high speed packet forwarding. To that end, we opted for an online algorithm that tracks typical DDoS metrics such as incoming flows and packet symmetry ratio for specific network subnets of interest. As the mechanism was implemented entirely in the data plane, no involvement of external controllers or systems was necessary for detection purposes, hence enabling rapid control loops. This approach was evaluated in a SmartNIC-based testbed yielding accurate detection results. Furthermore, high packet-rate tests were conducted to validate the processing performance of the mechanism P4-mechanism. The results were promising as the mechanism conforms to the packet processing requirements of modern environments despite being implemented on moderate-cost hardware. More powerful platforms (e.g.

hardware switches, FPGAs) are expected to perform even better; however, developers should always be wary of platform-specific limitations and intricacies despite P4 being a universal device-agnostic language.

After the detection of an anomaly, appropriate countermeasures should be deployed. Hence, **sections 5 and 6** discuss related mechanisms and schemas for the mitigation of distributed attacks. Specifically, **sections 5** considers the assignment of generic mitigation actions to counter multi-vector attacks in a heterogeneous network environment, comprised of devices with different capabilities and control/management APIs. The assignment was modeled as an integer problem that attempts to satisfy operational requirements (reward maximization) subject to specific constraints (hardware capacity). Assigned actions are translated to device-specific rules and appropriately deployed. Intelligent distribution of rules utilizing defense resources from various defense stages, yields noticeable improvements compared to rigid/standalone mitigation mechanisms.

With regards to rule distribution, techniques for automated and homogeneous deployment via abstraction layers were investigated, employing both SDN controllers and general purpose automation frameworks. As a generic observation, typical SDN solutions (e.g. OpenFlow, P4) and YANG models may be hindered by vendor-specific issues/bugs related to implementation details and platform limitations. Abstraction layers, be it multi-protocol controllers or generic automation frameworks, can bridge the gap between vendors and operators enabling generic events to be translated to device-specific instructions.

The sheer volume of present-day cyber threats comprised of multiple sources may overwhelm on-premise resources and/or saturate important links. As network anomalies are better pinpointed near the victim and more efficiently mitigated closer to their sources, provider collaborations seem well-suited to mitigate massive DDoS attacks early in the attack path. Thus, **section 6** extended the approach presented in **section 5**, investigating the establishment of trusted federations among adjacent and disjoint network domains that collectively mitigate malicious traffic assisted by digital (smart) contracts. Federated partners are able to (i) process all available mitigation offerings pertaining to a particular attack incident, (ii) identify the optimal sets of flows that should be mitigated by a collaborator, (iii) announce these sets by issuing Smart Contracts to the appropriate mitigation collaborators and (iv) collaboratively deploy mitigation solutions. As mentioned above, careful selection of malicious flows (e.g. heavy hitters) and subsequent assignment to reliable mitigators (i.e. reputation scores) should outperform other approaches.

Note that, distributing filtering rules within a domain (**sections 5**) or among collaborating domains (**sections 6)** alleviates the burden to an extent. However, commonly used source-centric approaches for monitoring and filtering may raise issues in terms of scalability (i.e. volume of source IPs) and effectiveness (e.g. spoofed IPs). Administrators (or even tenants) within network domains may profit from customized solutions for data collection and subsequently anomaly detection/mitigation tailored to specific anomalies and attacks.

These considerations were addressed in **section 7** via a two-level schema for Anomaly Detection and Mitigation. Specifically, the proposed approach incorporates the P4-based DDoS detection schema presented in **section 4** and further adopts XDP to create performant middleboxes, based on the identified attack. These middleboxes operate either (i) as programmable Deep Packet Inspectors (DPI), extracting monitoring data, or (ii) as flexible firewalls that block malicious traffic, using Machine Learning methods to create appropriate malicious signatures. While our initial proof-of-concept considers DNS volumetric attacks, this schema can extend to other attacks as well.

An important differentiation with previous efforts is that we do not rely on source IPs but use an IP-agnostic approach (i.e. packet signatures) to classify and ultimately mitigate attacks. This approach was based on the widely observed fact that volumetric DDoS attacks especially UDP-based may be characterized by a modest number of salient characteristics, thus enabling efficient Machine Learning algorithms. Note that we did not consider temporal correlations since they may hinder timely anomaly detection and mitigation. As a proof-of-concept, classification was based on the *Random Forests* supervised Machine Learning algorithm, trained with DNS datasets from past attacks and benign traffic from production networks. The experimental results were promising and drew interesting conclusions in terms of (i) accuracy, (ii) generalization capabilities of the detection mechanism and (iii) reduction of total number of filtering rules compared to source-based mechanisms. Moreover, XDP middleboxes also achieved high packet processing rates consistent with emerging network traffic profiles.

8.2 Areas for Future Research

This dissertation investigated different aspects of DDoS protection, each one associated with specific challenges and characteristics. Potential extensions may include an in-depth study of different detection/classification techniques for common attack vectors. Volumetric DNS attacks are summarized accurately by Random Forests, however each vector might require specific handling and in-depth tuning of detection and classification mechanisms, be it machine learning algorithms, time series analysis or statistical models. In a closely related topic, researchers usually have limited or no access to real network data, benign or malicious, primarily due to privacy concerns. Apart from privacy-aware solutions (e.g. data anonymization), these concerns may be also circumvented by utilizing statistical traffic attributes that can be used to artificially synthesize traces. Related efforts may leverage generative models (e.g. Generative Adversarial Networks [179]) to create synthetic yet realistic data, potentially used to further enhance network security schemas.

Another area for future research are data mining and processing techniques to generate appropriate signatures based on offline data from past incidents. These signatures will describe characteristics of attack vectors and can be further shared or even collaboratively created within trusted federations and defense coalitions [180]. Collaborative efforts may benefit from Federated Learning techniques that enable multiple entities to build a common machine learning model without sharing potentially sensitive data thus appeasing related concerns.

A common knowledge-base of malicious signatures per attack vector can embolden mitigation mechanisms or even enable preemptive filtering. As an example frequently abused domains could be used to filter malicious DNS responses used in volumetric attacks. After generation, these signatures should be examined for correlations and aggregated as much as possible. Thus, another area of potential interest are related summarization techniques that minimize signatures to preserve memory and computing resources. As exhibited, XDP-based processing might be delayed by memory lookups, a fact that may be correct for other systems as well.

Note that signature based mechanisms are ideal for DDoS mitigation but specific attacks might require additional effort (i.e. state information); a typical example are TCP SYN attacks. Though not impossible to create stateless signature profiles for malicious TCP traffic, typically some state information needs to be maintained. DDoS detection and mitigation services, stateless or stateful, can be offloaded within programmable middleboxes, protecting valuable resources in firewalls, routers and hosts. To that end, as another extension we envisage a multi-tier architecture for cyber defense that (i) rapidly identifies and localizes an anomaly, correlating data from multiple devices, (ii) on-demand inspects malicious traffic to identify the attack vector and the victim and (iii) creates filters tailored to the specific attack vector, preferably not reliant on easily spoofed source IPs. This approach can be integrated on top of interdomain federations powered by Distributed Ledger Technologies or within NFV environments. Federated members and users can provide XDP programs to be deployed across network paths or slices for monitoring and filtering purposes.

9 Publications

- M. Dimolianis, **A. Pavlidis**, V. Maglaris, "A Network Traffic Classification and Attack Mitigation Schema based on Programmable Data Planes", IEEE Transactions on Network and Service Management, 2020 **(under review)**
- **A. Pavlidis**, M. Dimolianis, K. Giotis, L. Anagnostou, N. Kostopoulos, T. Tsigkritis, I. Kotinas, D. Kalogeras, V. Maglaris "Orchestrating DDoS Mitigation via Blockchain-based Network Provider Collaborations", The Knowledge Engineering Review, 35, e16, 2020. doi:10.1017/S0269888920000259
- M. Dimolianis, **A. Pavlidis**, V. Maglaris, "A Multi-Feature DDoS Detection Schema on P4 Hardware", in Proc. of the 23rd IEEE Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), 2020
- N. Kostopoulos, **A. Pavlidis**, M. Dimolianis, D. Kalogeras, V. Maglaris, "A Privacy-Preserving Schema for the Detection and Collaborative Mitigation of DNS Water Torture Attacks in Cloud Infrastructures", in Proc. of the 8th IEEE International Conference on Cloud Networking (CloudNet), 2019
- **A. Pavlidis**, M. Dimolianis, D. Kalogeras, V. Maglaris, "Automated Distribution of Access Control Rules in Defense Layers of an Enterprise Network", in Proc. of the 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019
- M. Dimolianis, **A. Pavlidis**, D. Kalogeras, V. Maglaris, "Mitigation of Multi-vector Network Attacks via Orchestration of Distributed Rule Placement", in Proc. of the 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019
- K. Giotis, **A. Pavlidis**, L. Anagnostou, M. Dimolianis, T. Tsigkritis, D. Kalogeras, N. Kostopoulos, I. Kotinas, V. Maglaris, "Blockchain-based Federation of Network Providers for Collaborative DDoS Mitigation", 3rd Symposium on Distributed Ledger Technology, 2018
- **A. Pavlidis**, G. Sotiropoulos, K. Giotis, D. Kalogeras and V. Maglaris, "NFV-compliant Traffic Monitoring and Anomaly Detection based on Dispersed Vantage Points in Shared Network Infrastructures," in Proc. of the 4th IEEE Conference on Network Softwarization (NetSoft), 2018
- P. L. Ventre, J. Ortiz, A. Mendiola, C. Fernández, **A. Pavlidis**, P. Sharma, S. Buscaglione, K.Stamos, A. Sevasti, D. Whittaker, "Deploying SDN in GÉANT production network", in Proc. of the 3rd IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2017

10 Extended Abstract in Greek – Εκτεταμένη Περίληψη στα Ελληνικά

Η αλματώδης αύξηση της ταχύτητας ευρυζωνικών συνδέσεων και του αριθμού διασυνδεδεμένων συσκευών έχουν προκαλέσει ριζικές αλλαγές στη λειτουργία υπολογιστικών και δικτυακών υποδομών. Σχετικές τεχνολογίες που χρησιμοποιούνται αφορούν την αυτοματοποίηση εργασιών μέσω λογισμικού, τη δυναμική προσαρμογή στις εκάστοτε συνθήκες και εν γένει μηχανισμούς για τον ριζικό προγραμματισμό συσκευών και συστημάτων. Επιπρόσθετα, βασικό μέλημα είναι η συνεχής συλλογή και επεξεργασία δεδομένων με στόχο την εξαγωγή γνώσης (analytics) καθώς και την προστασία από κυβερνοεπιθέσεις.

Η έγκαιρη και αποτελεσματική ανίχνευση και αντιμετώπιση επιθέσεων είναι πεδίο ζωτικής σημασίας για την ομαλή λειτουργία των δικτύων. Ειδικότερα, σύγχρονες πρακτικές χρησιμοποιούν τις αρχιτεκτονικές Δικτύων Οριζόμενων από Λογισμικό (Software-Defined Networking - SDN) και Εικονικοποίησης Δικτυακών Λειτουργιών (Network Function Virtualization - NFV). Οι αρχιτεκτονικές αυτές παρέχουν τη δυνατότητα υλοποίησης ευέλικτων και κλιμακώσιμων μηχανισμών για την κατ' απαίτηση ανίχνευση και αντιμετώπιση μαζικών ετερογενών επιθέσεων.

Η άνθηση των δικτύων SDN ταυτίστηκε με το πρωτόκολλο OpenFlow. Κύριο χαρακτηριστικό τους είναι η διάκριση του επίπεδο ελέγχου (control plane) από το επίπεδο προώθησης δεδομένων (data/forwarding plane). Ως εξέλιξη των δικτύων SDN και του OpenFlow, η διεθνής ερευνητική κοινότητα προτείνει την ανάπτυξη προγραμματιζόμενων συσκευών επεξεργασίας και προώθησης πακέτων (Programmable Data Planes) με κύριο εκφραστή τη γλώσσα P4 (Programming Protocol-Independent Packet Processors).

Παράλληλα με τη γλώσσα P4, ιδιαίτερη άνθηση έχουν και άλλες σχετικές τεχνολογίες που προσφέρουν ευέλικτη προώθηση και επεξεργασία πακέτων σε υψηλές ταχύτητες. Αυτές οι λύσεις βασίζονται συνήθως στο λειτουργικό GNU/Linux και μεταφέρουν τα πακέτα απευθείας από την κάρτα δικτύου στην εφαρμογή, παρακάμπτοντας τις λειτουργίες του πυρήνα (kernel-bypass). Ενδεικτικά, αναφέρουμε το Data Plane Development Kit (DPDK) και το PF_RING. Μια εναλλακτική λύση είναι το XDP (eXpress Data Path), το οποίο επιτρέπει την εκτέλεση «προγραμμάτων» επεξεργασίας εντός του kernel, αλλά σε χαμηλό επίπεδο (οδηγούς δικτυακών καρτών). Το XDP ήδη χρησιμοποιείται σε εφαρμογές Εξισορρόπησης Φόρτου, Συστημάτων Ανίχνευσης/Αποτροπής Εισβολής, και Απόρριψης Κακόβουλης Κίνησης. Αυτές οι λύσεις αποκτούν ακόμα μεγαλύτερη σημασία υπό το πρίσμα του NFV, καθώς οι δικτυακές λειτουργίες βασίζονταν παραδοσιακά σε ειδικό εξοπλισμό υψηλού κόστους. Πλέον είναι ιδιαίτερα συνηθισμένο, οικονομικά προσιτές συσκευές γενικού σκοπού (Commercial off-the-shelf - COTS Hardware) να υποστηρίζουν διάφορες τεχνικές για γρήγορη προώθηση και επεξεργασία πακέτων.

Οι σύγχρονες τεχνολογίες έχουν δημιουργήσει νέες δυνατότητες για την παρακολούθηση, ανίχνευση και αντιμετώπιση επιθέσεων οι οποίες εξαπλώνονται ραγδαία κυρίως λόγω του πλήθους των διασυνδεδεμένων συσκευών και των κενών ασφαλείας που παρουσιάζουν. Χαρακτηριστικά παραδείγματα του παρελθόντος αφορούν δυσθεώρητες σε κλίμακα επιθέσεις άνω του 1 Terabit/δευτερόλεπτο (Github, Dyn). Ανεξαρτήτως αν πρόκειται για μεμονωμένες ή μαζικές ενέργειες, οι κυβερνοεπιθέσεις συνήθως αποσκοπούν σε: απώλεια δεδομένων, μόλυνση συσκευών για μελλοντική συμμετοχή τους σε επιθέσεις ή/και στην παρακώλυση παρεχόμενων υπηρεσιών (Κατανεμημένες Επιθέσεις Άρνησης Παροχής Υπηρεσίας - Distributed Denial of Service Attacks / DDoS). Ειδικότερα οι επιθέσεις DDoS προσπαθούν να πλημμυρίσουν δικτυακούς πόρους (π.χ. ζεύξεις) ή/και να κατασπαταλήσουν υπολογιστικούς πόρους σε συσκευές (π.χ. δρομολογητές, εξυπηρετητές) μέσω κατάλληλα κατασκευασμένων μηνυμάτων. Εν γένει, οι επιθέσεις DDoS ταξινομούνται σε (1) όγκου (volumetric), (2) πρωτοκόλλου (protocol) και (3) εφαρμογής (application) οι οποίες αντίστοιχα προσπαθούν να πλήξουν ζεύξεις, υπολογιστικούς πόρους και εφαρμογές.

Αποκορύφωμα των σύγχρονων επιθέσεων είναι οι ταυτόχρονες ετερογενείς επιθέσεις (multi-vector) που συνδυάζουν διάφορες επιθέσεις δημιουργώντας (1) δυσκολία στην ανίχνευση επιμέρους συνιστωσών ή/και (2) δημιουργώντας μεγάλο όγκο κίνησης. Ιδιαίτερο ενδιαφέρον παρουσιάζει και η άνθηση πλατφορμών (Booters & Stressers) που πραγματοποιούν επιθέσεις κατ' απαίτηση έναντι σημαντικά μικρής χρηματικής αμοιβής.

Σημαντικές προκλήσεις και πιθανά προβλήματα των μηχανισμών άμυνας είναι:

- Αποδοτική εξαγωγή και επεξεργασία δεδομένων κίνησης: Οι μηχανισμοί ανίχνευσης επιθέσεων και δικτυακών ανωμαλιών συνήθως βασίζονται σε δείγματα πακέτων και ομαδοποιημένες δικτυακές ροές (flow records). Αυτά εξάγονται από τις δικτυακές συσκευές και αποστέλλονται σε ξεχωριστά συστήματα επεξεργασίας. Οι σχετικοί αλγόριθμοι καθώς και τα υποκείμενα συστήματα πρέπει να μπορούν να ανταπεξέλθουν στην πυκνή και συνεχή ροή δεδομένων δικτυακής κίνησης, χωρίς να υστερούν σε ευκρίνεια και ακρίβεια.

- Έξυπνη χρήση αμυντικών πόρων για αντιμετώπιση επιθέσεων: Η αποτελεσματική αντιμετώπιση ετερογενών επιθέσεων απαιτεί κατάλληλη αξιοποίηση όλων των διαθέσιμων πόρων (π.χ. δρομολογητές, μεταγωγείς SDN και εξυπηρετητές). Ωστόσο οι επιθέσεις υπερβαίνουν της δυνατότητες του κάθε δικτύου προκαλώντας προβλήματα σε ζεύξεις και ενδιάμεσα συστήματα. Η πιο συνηθισμένη λύση είναι η ανακατεύθυνση της κίνησης στο «κενό» (Blackholing) μέσω διάδοσης κατάλληλων ανακοινώσεων BGP. Ωστόσο, η προσέγγιση αυτή οδηγεί στην απόρριψη και της καλόβουλης κίνησης που συνήθως είναι ο στόχος της επίθεσης. Εναλλακτικά, συνεργαζόμενα δίκτυα μπορούν να αιτηθούν και να λάβουν βοήθεια για την συλλογική αντιμετώπιση της επίθεσης. Βασικά προβλήματα της πιθανής συνεργασίας αφορούν την έλλειψη κατάλληλων κινήτρων και μηχανισμών επικοινωνίας μεταξύ εταίρων.
- Συνήθεις περιορισμοί αμυντικών μηχανισμών: Οι υπάρχοντες μηχανισμοί ανίχνευσης και αντιμετώπισης κυβερνοεπιθέσεων βασίζονται είτε σε κλειστές λύσεις προσφερόμενες ως φυσική/εικονική συσκευή είτε σε υποδομές/παρόχους υπηρεσιών αντιμετώπισης επιθέσεων (DDoS Protection Service Providers). Στην πρώτη περίπτωση χρησιμοποιείται εξειδικευμένος εξοπλισμός με τις αντίστοιχες εμπορικές άδειες. Ταυτόχρονα, οι δυνατότητες προγραμματισμού για αυτές τις συσκευές είναι παραδοσιακά περιορισμένες και εξαρτώμενες από τον εκάστοτε κατασκευαστή. Συνεπώς δημιουργούνται μονολιθικές λύσεις, υψηλού κόστους και περιορισμένης ευελιξίας, με τελικό αποτέλεσμα την προσκόλληση σε συγκεκριμένου τύπου εξοπλισμό και κατασκευαστή (vendor lock-in). Οι πάροχοι υπηρεσιών από την άλλη πλευρά ανακατευθύνουν όλη την κίνηση προς το θύμα στην υποδομή τους, όπου εντοπίζεται και φιλτράρεται η επίθεση ενώ η καλόβουλη κίνηση επιστρέφει στο επιχειρησιακό δίκτυο. Οι μηχανισμοί αυτοί συνήθως εισαγάγουν καθυστέρηση προς τον τελικό αποδέκτη και επίσης εγείρουν προβληματισμούς ως προς την ιδιωτικότητα και το απόρρητο των επικοινωνιών.

Η παρούσα διδακτορική διατριβή μελετά μεθόδους συλλογής δεδομένων, ανίχνευσης και (συνεργατικής) αντιμετώπισης κυβερνοεπιθέσεων με έμφαση σε κατανεμημένες επιθέσεις άρνησης παροχής υπηρεσίας. Επιπρόσθετα, διερευνώνται τεχνολογίες για τον προγραμματισμό του επιπέδου δεδομένων των δικτυακών συσκευών καθώς και τον έλεγχο τους μέσω λογισμικού, με στόχο την δημιουργία αυτοματοποιημένων υπηρεσιών ασφάλειας και συλλογής δεδομένων.

Η κύρια συνεισφορά της εργασίας μπορεί να χωριστεί σε τέσσερις διακριτούς αλλά όχι ανεξάρτητους άξονες, οι οποίοι αντιστοιχούν στα **κεφάλαια 3, 4, 5, 6 και 7**. Συνοπτικά:

- Τα **κεφάλαια 3, 4** ασχολούνται με μηχανισμούς συλλογής και επεξεργασίας δεδομένων κυρίως για ανίχνευση δικτυακών ανωμαλιών και επιθέσεων. Αρχικά προσεγγίζεται το θέμα μέσω συνηθισμένων μηχανισμών συλλογής και επεξεργασίας δεδομένων. Στην συνέχεια μελετώνται δυνατότητες των δικτυακών συσκευών για κατανεμημένη επεξεργασία και ανίχνευση επιθέσεων.
- Το **κεφάλαιο 5** εστιάζει στην αυτοματοποιημένη αντιμετώπιση κυβερνοεπιθέσεων με βάση διαχειριστικές πολιτικές και τους εκάστοτε περιορισμούς (υλικό/λογισμικό) των συσκευών. Επίσης διερευνώνται τεχνικές για την μετάφραση αφαιρετικών οδηγιών σε ειδικούς κανόνες καθώς και την αυτοματοποιημένη διανομή τους μέσω κατάλληλων διεπαφών και πρωτοκόλλων.
- Το **κεφάλαιο 6** επεκτείνει την προσέγγιση του προηγούμενου κεφαλαίου σε πολλαπλές διαχειριστικές περιοχές υπό την μορφή ενός ομόσπονδου περιβάλλοντος για παρόχους δικτυακών υπηρεσιών. Σημαντικό πλεονέκτημα είναι η προστασία των ζεύξεων καθώς και των σχετικών αμυντικών μηχανισμών.
- Το **κεφάλαιο 7** πηγάζει από ζητήματα που προέκυψαν σε προηγούμενα κεφάλαια και συνδυάζει διαφορετικές τεχνολογίες σε έναν ολοκληρωμένο και ευέλικτο μηχανισμό δυο επιπέδων για την αποδοτική ανίχνευση και αντιμετώπιση επιθέσεων.

Αναλύοντας με μεγαλύτερη λεπτομέρεια, το κεφάλαιο 3 εστιάζει σε υπηρεσίες συλλογής και επεξεργασίας δεδομένων δικτυακής κίνησης, οι οποίες προσφέρονται σε χρήστες (ενοίκους και διαχειριστές) κοινών δικτυακών υποδομών. Κάθε δικτυακή συσκευή αποτελεί εν δυνάμει ένα διακριτό σημείο εποπτείας (vantage point) με δυνατότητα παρατήρησης και εξαγωγής δεδομένων διαφορετικής ευκρίνειας. Η ανάλυση των δεδομένων, είτε αυτά αφορούν συγκεκριμένους χρήστες είτε τους διαχειριστές της υποδομής, μπορεί να ενισχύσει την ικανότητα ανίχνευσης κεντρικών ή τοπικά εστιασμένων ανωμαλιών και επιθέσεων.

Αυτό γίνεται εμφανέστερο αν αναλογιστεί κανείς τη διαστρωμάτωση που παρουσιάζουν τόσο παραδοσιακά επιχειρησιακά δίκτυα όσο και μοντέρνες υποδομές υπολογιστικών νεφών. Αναλόγως τον ρόλο της, η κάθε συσκευή έχει διαφορετική οπτική ως προς την δικτυακή κίνηση. Αυτό ενισχύεται με τις συνήθεις πρακτικές δειγματοληψίας της κίνησης κατά την συλλογή δεδομένων (δείγματα πακέτων και ομαδοποιημένες ροές) που εφαρμόζονται συνήθως με στόχο την αποφόρτιση των κεντρικά τοποθετημένων δικτυακών συσκευών.

Γενικώς, οι τεχνολογίες για την επεξεργασία μεγάλου όγκου δεδομένων όπως ουρές μηνυμάτων (π.χ. Apache Kafka) αλλά και κατάλληλες αποθηκευτικές δομές χρησιμοποιούνται εκτενώς σε παραγωγικά δίκτυα. Ωστόσο, οι σχετικές λύσεις βασίζονται εγγενώς σε παραδοσιακά πρωτόκολλα και τεχνικές για την αποστολή δεδομένων στα αντίστοιχα συστήματα. Πιθανοί προβληματισμοί είναι η μειωμένη ακρίβεια λόγω δειγματοληψίας, οι απαιτούμενοι πόροι για την επεξεργασία μεγάλου όγκου δεδομένων και οι ανάγκες για συνεχή πληροφορία (π.χ. συχνά ερωτήματα σε αποθηκευτικές δομές). Εναλλακτική προσέγγιση όσον αφορά τις επιθέσεις, μπορούν να αποτελέσουν μηχανισμοί για τον προγραμματισμό των δικτυακών συσκευών που θα μειώσουν τον χρόνο ανίχνευσης και κατ’ επέκταση αντίδρασης.

Σε αυτή τη λογική, παρουσιάζεται στο κεφάλαιο 4 ένας μηχανισμός για την ανίχνευση επιθέσεων σε προγραμματιζόμενες δικτυακές συσκευές. Ο προτεινόμενος μηχανισμός διενεργεί απλούς αλλά αποτελεσματικούς στατιστικούς υπολογισμούς με χρήση της γλώσσας P4 για την ανίχνευση επιθέσεων DDoS. Σημαντικά πλεονεκτήματα είναι η άμεση ανίχνευση επιθέσεων χωρίς την παρέμβαση εξωτερικών συστημάτων και η άμεση αποστολή ειδοποιήσεων για περεταίρω ενέργειες. Συνοπτικά, ο μηχανισμός: (1) παρακολουθεί την κίνηση και διατηρεί μετρικές ομαδοποιημένες σε διαφορετικά επίπεδα ευκρίνειας (π.χ. ανά προστατευόμενο υποδίκτυο) – οι μετρικές αφορούν μοναδικές ροές και ασυμμετρία κίνησης, (2) συγκρίνει τις μετρικές αυτές με τιμές αναφοράς (κατώφλια) και (3) πυροδοτεί μηνύματα κινδύνου σε περίπτωση που όλες οι μετρικές παραβιάσουν τα αντίστοιχα κατώφλια. Αυτή η διαδικασία σχετίζεται άμεσα με την προσέγγιση του κεφαλαίου 3, καθώς επιτρέπει την αυτόνομη εκτέλεση αλγορίθμων σε διάφορες συσκευές – σημεία εποπτείας – με στόχο την κατανεμημένη και συλλογική ανίχνευση της επίθεσης.

Ο μηχανισμός αξιολογήθηκε σε υλικό (κάρτες Netronome SmartNIC) ως προς την ακρίβεια και την ικανότητα προώθησης πακέτων. Συγκεκριμένα, αφενός συγκρίνουμε την ακρίβεια του μηχανισμού για διαφορετικά μεγέθη επιθέσεων καθώς και για δυο διαφορετικούς τρόπους λειτουργίας – με ή χωρίς χρήση συμμετρίας κίνησης σαν μετρική. Τα αποτελέσματα δείχνουν πως ο μηχανισμός παρουσιάζει βελτίωση ως προς την ακρίβεια με την χρήση της συμμετρίας, ιδίως για τυπικές και μεγάλες επιθέσεις. Αφετέρου, παρουσιάζεται η επίπτωση που έχει η χρήση της εκάστοτε μετρικής στην ικανότητα του μηχανισμού να προωθεί και να μετράει πακέτα. Παρατηρούμε πως η διαδικασία συλλογής δεδομένων υστερεί της προώθησης, δηλαδή οι μετρήσεις παρουσιάζουν χαμηλότερες τιμές από τις πραγματικές. Αυτό οφείλεται σε θέματα υλοποίησης των «καταχωρητών» (registers) του P4 στην συγκεκριμένη συσκευή και ειδικότερα, στον βαθμό που μπορέσαμε να το διασταυρώσουμε, στην ταυτόχρονη προσπέλαση θέσεων μνήμης από παράλληλες διεργασίες. Αυτό οδηγεί και σε μια γενικότερη παρατήρηση πως η υποκείμενη υλοποίηση παίζει σημαντικό ρόλο και εξαρτάται σχεδόν αποκλειστικά από τον κατασκευαστή παρότι το P4 αποτελεί μια ενοποιημένη γλώσσα προγραμματισμού ανεξαρτήτως υλικού («στόχου» – target).

Συνολικά, αξιοσημείωτο είναι πως ακόμα και με κάρτες σχετικά μικρού κόστους ήταν εφικτό να υλοποιήσουμε έναν μηχανισμό που να μπορεί να ανταποκριθεί σε τυπικές ταχύτητες μιας πλήρως χρησιμοποιούμενης γραμμής (2 εκατομμύρια πακέτα/δευτερόλεπτο στα 10 Gigabit). Είναι αναμενόμενο συσκευές με μεγαλύτερες δυνατότητες π.χ. μεταγωγείς και FPGAs, να μπορούν να ανταπεξέλθουν σε ακόμα υψηλότερες ταχύτητες.

Όπως έχει αναφερθεί, η ανίχνευση είναι ένα από τα πρώτα στάδια των αμυντικών μηχανισμών. Στην συνέχεια, η διατριβή ασχολείται με τεχνικές για την αποτελεσματική αντιμετώπιση επιθέσεων. Στο κεφάλαιο 5, παρουσιάζεται ένας μηχανισμός για την αποκοπή ετερογενών επιθέσεων μεγάλης κλίμακας (multi-vector attacks). Ο στόχος είναι η ευέλικτη και συνολικά αποδοτικότερη αντιμετώπιση της κακόβουλης κίνησης αναθέτοντας κανόνες σε διάφορες συσκευές κατά μήκος του ίχνους της επίθεσης. Η ανάθεση μοντελοποιείται σαν ένα συνδυαστικό πρόβλημα βελτιστοποίησης ακέραιου προγραμματισμού. Ειδικότερα, αντικείμενα (γενικοί κανόνες) ανατίθενται σε κάδους (συσκευές ομαδοποιημένες σε στάδια άμυνας) με στόχο την βελτιστοποίηση κάποιας συνάρτησης κέρδους υπό συγκεκριμένους περιορισμούς. Το κέρδος προκύπτει άμεσα από διαχειριστικές πολιτικές που υποδεικνύουν την αποκοπή συγκεκριμένων επιθέσεων και των αντίστοιχων κακόβουλων ροών, σε κάποιο στάδιο άμυνας. Οι περιορισμοί επηρεάζονται άμεσα από την χωρητικότητα των δικτυακών συσκευών π.χ. πόσες εγγραφές υποστηρίζει το hardware. Γενικώς, οι αλγόριθμοι ακέραιου προγραμματισμού χαρακτηρίζονται από αυξημένη υπολογιστική πολυπλοκότητα. Παρόλα αυτά στην περίπτωση μας το πρόβλημα προσεγγίζεται με τεχνικές συμπύκνωσης των αντικειμένων προς ανάθεση (π.χ. μαζική ανάθεση ροών του ίδιου τύπου) για την μείωση της εισόδου και την εύρεση λύσης σε λογικά χρονικά πλαίσια.

Βασικό κομμάτι της προτεινόμενης αρχιτεκτονικής είναι επίσης η μετάφραση και τελική διανομή των κανόνων στις συσκευές. Για αυτό το σκοπό εξετάζονται τεχνικές μέσω ελεγκτών SDN αλλά και σχετικών μηχανισμών για δικτυακή αυτοματοποίηση. Αρχικά χρησιμοποιήθηκε ο ελεγκτής SDN Ryu, ο οποίος υποστηρίζει διάφορα πρωτόκολλα διαχείρισης όπως BGP και OpenFlow. Ως εναλλακτική διερευνήθηκε το Salt, μια δημοφιλής πλατφόρμα σχεδιασμένη για ταυτόχρονη διαχείριση πολλαπλών συσκευών μέσω κατάλληλων διεπαφών, πρωτοκόλλων αλλά και εγκατεστημένου λογισμικού εντός των συσκευών (σπανίως). Τέτοιες λύσεις παρουσιάζουν ιδιαίτερο ενδιαφέρον καθώς υλοποιούν ένα επίπεδο αφαίρεσης προς τις εφαρμογές ενώ ταυτόχρονα προσαρμόζονται στις απαιτήσεις και ιδιαιτερότητες των εκάστοτε δικτυακών συσκευών. Εν γένει, αποτελούν τον συνδετικό κρίκο ανάμεσα σε διαχειριστές δικτύων και κατασκευαστές συσκευών προσφέροντας ευελιξία και πολλές δυνατότητες για αυτοματοποίηση.

Σε περιπτώσεις επιθέσεων μεγάλης κλίμακας, ενδεχομένως δεν επαρκούν οι παραπάνω μηχανισμοί οι οποίοι λειτουργούν στο πλαίσιο μιας δικτυακής περιοχής. Πιθανά προβλήματα και περιορισμοί σχετίζονται με την έλλειψη αμυντικών πόρων καθώς και την υπερφόρτωση/κορεσμό των εξωτερικών ζεύξεων. Συνεπώς η αποκοπή δεν είναι εφικτή εντός του δικτύου που δέχεται την επίθεση και αποκτά νόημα η αντιμετώπιση των επιθέσεων σε συνεργασία με τρίτες διαχειριστικές οντότητες.

Αποτρεπτικοί παράγοντες για την συνεργασία μεταξύ οργανισμών συνήθως είναι: (1) η έλλειψη εμπιστοσύνης, (2) κατάλληλοι μηχανισμοί ανταλλαγής πληροφοριών και (3) τα περιορισμένα κίνητρα. Καινοτόμες τεχνολογίες βασισμένες σε αρχιτεκτονική Αλυσίδων Επιβεβαιωμένων Συναλλαγών (Blockchain), οι οποίες άνθισαν με την εμφάνιση των κρυπτονομισμάτων, προσφέρουν διαφάνεια των συναλλαγών - ανακοινώσεων, αποκεντρωμένη φύση καθώς και μη δυνατότητα άρνησης συναλλαγής (non repudiation). Τα παραπάνω χαρακτηριστικά καθιστούν τέτοιες τεχνολογίες κατάλληλες για τη διαμόρφωση αυτοματοποιημένων συνεργατικών σχημάτων μεταξύ έμπιστων ομόσπονδων φορέων περιορίζοντας τυχόν αποτρεπτικούς παράγοντες.

Κατ’ αυτόν τον τρόπο, η προσέγγιση του κεφαλαίου 5 επεκτείνεται στο κεφάλαιο 6, το οποίο εστιάζει σε ένα συνεργατικό σχήμα για την συλλογική αντιμετώπιση μαζικών κυβερνοεπιθέσεων. Η αρχιτεκτονική βασίζεται σε ένα ομόσπονδο σχήμα εμπιστοσύνης μεταξύ παρόχων δικτυακών υπηρεσιών και ενσωματώνει έξυπνα ψηφιακά συμβόλαια (smart contracts) αποτυπωμένα σε αλυσιδωτές δομές συναλλαγών. Σημειώνεται πως τεχνολογίες Blockchain δεν αποτελούν αυτοσκοπό για την συγκεκριμένη προσέγγιση, αλλά έναν «επίσημο» και κοινώς αποδεκτό δίαυλο επικοινωνίας μεταξύ εταίρων, για την σηματοδοσία, τον συντονισμό και την ενορχήστρωση του συνεργατικού μηχανισμού άμυνας. Τα συμβόλαια καταγράφονται, επιβεβαιώνονται από τους αντισυμβαλλόμενους και εφαρμόζονται αυτόματα. Το όφελος δεν συνεπάγεται κατ’ ανάγκη την ύπαρξη κάποιου κρυπτονομίσματος αλλά ένα ευρύτερο ανταλλακτικό πλαίσιο επιβράβευσης (π.χ. μονάδες εμπιστοσύνης) για την από κοινού αντιμετώπιση επιθέσεων.

Η ανάθεση κανόνων αποκοπής στους ομόσπονδους εταίρους πραγματοποιείται από τον αμυνόμενο με μια παραλλαγή του αλγορίθμου που παρουσιάστηκε στο κεφάλαιο 5. Σε αυτή την περίπτωση, βασικό κριτήριο είναι η σημασία της εκάστοτε κακόβουλης ροής που δεσμεύεται να αποκόψει ένας ομόσπονδος εταίρος καθώς και η αξιοπιστία του σε προηγούμενα περιστατικά. Οι ακριβείς μηχανισμοί για την αποκοπή κακόβουλης κίνησης εξαρτώνται αποκλειστικά από την δυνατότητα και τεχνογνωσία του κάθε εταίρου. Χωρίς να είναι δεσμευτικό, υλοποιήθηκε πιλοτικά ένας μηχανισμός αποκοπής πηγών (blacklist) στο περιβάλλον XDP.

Σημειώνεται πως, οι περισσότεροι μηχανισμοί ανίχνευσης και αντιμετώπισης επιθέσεων χρησιμοποιούν σε κάποιο βαθμό τις διευθύνσεις IP των επιτιθέμενων (κακόβουλες πηγές). Αφενός, το μεγάλο πλήθος πηγών δημιουργεί πρόβλημα κλίμακας τόσο κατά την αντιμετώπιση αλλά και κατά την διαδικασία παρακολούθησης και διατήρησης δεδομένων για όλες τις πηγές (πιθανές λύσεις: κατανομή πόρων, συνεργασία, ομαδοποίηση πηγών). Αφετέρου, οι πηγές είναι αμφιβόλου γνησιότητας καθώς κακόβουλοι κόμβοι πλαστογραφούν την διεύθυνση τους. Σε αντίθετη περίπτωση, η ιχνηλάτιση και αποκοπή τους θα ήταν απλούστερη διαδικασία. Ωστόσο, παρά τις σχετικές προσπάθειες, το πρόβλημα των ψευδεπίγραφων διευθύνσεων IP εξακολουθεί να υφίσταται.

Σαν λύση στα παραπάνω, μπορούν να χρησιμοποιηθούν καινοτόμες προσεγγίσεις για τον προγραμματισμό του επιπέδου δεδομένων με στόχο την δημιουργία ευέλικτων μηχανισμών, ειδικά προσαρμοσμένων στους διάφορους τύπους επιθέσεων. Έτσι φτάνουμε και στον τελευταίο θεματικό άξονα της διατριβής, ο οποίος αποτυπώνεται στο κεφάλαιο 7. Το κεφάλαιο αυτό πηγάζει από εμπειρίες και προβλήματα προηγούμενων κεφαλαίων και προτείνει έναν ευέλικτο μηχανισμό δυο επιπέδων για την ταξινόμηση κίνησης σε καλόβουλη ή κακόβουλη και την τελική αποκοπή της.

Το πρώτο επίπεδο προσφέρει έναν γρήγορο μηχανισμό ανίχνευσης/αναγνώρισης της επίθεσης υλοποιείται μέσω του P4 μηχανισμού που παρουσιάζεται στο κεφάλαιο 4. Η αναγνώριση βασίζεται στις θύρες που χρησιμοποιούν συγκεκριμένοι τύποι επιθέσεων. Παραδείγματος χάριν, δικτυακή κίνηση που προέρχεται (πηγή) από την θύρα 53, αφορά επιθέσεις μεγάλου όγκου σχετικές με το Σύστημα Ονοματοδοσίας Τομέων (Domain Name System – DNS). Κατόπιν της αναγνώρισης, ενεργοποιείται το δεύτερο επίπεδο για την λεπτομερή συλλογή δεδομένων και αποκοπή της κακόβουλης κίνησης. Αυτό βασίζεται σε XDP προγράμματα υλοποιημένα σε γενικού σκοπού δικτυακές διεπαφές.

Η συλλογή δεδομένων εστιάζεται κυρίως σε χαρακτηριστικές τιμές των πεδίων των πακέτων, κατάλληλα επιλεγμένες ανάλογα με τον τύπο της επίθεσης. Μετά από την συλλογή, τα δεδομένα αποστέλλονται ομαδοποιημένα (με βάση τις άνω τιμές) προς ταξινόμηση σε συστήματα που χρησιμοποιούν μεθόδους επιβλεπόμενης μηχανικής μάθησης (supervised Machine Learning). Τα ομαδοποιημένα στοιχεία που ταξινομούνται ως κακόβουλα, αποτελούν συνοπτικές περιγραφές κακόβουλης κίνησης και εγκαθίστανται δυναμικά σε προγραμματιζόμενα φίλτρα XDP με στόχο την άμεση αποκοπή επιθέσεων.

Η ειδοποιός διαφορά με συναφή συστήματα είναι ότι δε βασιζόμαστε σε διευθύνσεις IP για ταξινόμηση και αποκοπή επιθέσεων. Για την ακρίβεια, προσπαθούμε να μην χρησιμοποιούμε καθόλου τέτοια πληροφορία, παρακάμπτοντας πιθανώς ψευδεπίγραφες διευθύνσεις. Η προσέγγιση πηγάζει από την υπόθεση πως επιθέσεις όγκου (volumetric) που βασίζονται στο πρωτόκολλο UDP, εκμεταλλεύονται ευαίσθητα συστήματα με προβλέψιμο τρόπο. Έτσι, η κίνηση μπορεί να αξιολογηθεί από ένα σχετικά μικρό πλήθος χαρακτηριστικών γνωρισμάτων. Η αρχική υλοποίηση εστιάζει σε επιθέσεις με βάση το DNS ωστόσο ο ίδιος μηχανισμός μπορεί να επεκταθεί και σε άλλους τύπους επιθέσεων.

Ειδικότερα για την ταξινόμηση, χρησιμοποιούμε τον αλγόριθμο Random Forest. Παρά ταύτα η διαδικασία δεν είναι άμεσα συνδεδεμένη με κάποια συγκεκριμένη μέθοδο. Για τις ανάγκες της πειραματικής αξιολόγησης χρησιμοποιήθηκαν δεδομένα πραγματικής κίνησης από παρελθοντικές επιθέσεις αλλά και καλόβουλης κίνηση από πραγματικά περιβάλλοντα.

Ως αποτέλεσμα της αξιολόγησης, προκύπτουν ενδιαφέροντα αποτελέσματα και συμπεράσματα. Γενικώς, η ταξινόμηση μέσω Random Forest αποδεικνύεται εξαιρετικά ακριβής, με την προϋπόθεση ότι τα δέντρα έχουν εκπαιδευτεί με αντιπροσωπευτικό δείγμα δεδομένων. Σε διαφορετική περίπτωση, π.χ. νέες επιθέσεις, είναι αρκετά πιθανό να μην ανιχνευτεί σωστά η επίθεση. Ωστόσο, οι διαχειριστές μπορούν να εκπαιδεύσουν σε μεγάλο βαθμό τέτοια μοντέλα ως προς την καλόβουλη κίνηση. Έτσι πιθανότατα θα μετριαστούν οι ψευδώς θετικές (False Positives) ταξινομήσεις που θα οδηγήσουν σε απώλεια καλόβουλης κίνησης. Ιδιαίτερο ενδιαφέρον παρουσιάζει επίσης η ανακύκλωση των περιγραφών/υπογραφών από τους επιτιθέμενους. Χαρακτηριστικά αναφέρεται πως το πλήθος των μοναδικών περιγράφων σε σύγκριση με τις αντίστοιχες πηγές (διευθύνσεις IP) παρουσιάζει μείωση από 86% μέχρι 99%.

Αυτό αποκτά μεγαλύτερη βαρύτητα στα πειράματα επίδοσης (throughput), τα οποία δείχνουν πως οι υλοποιήσεις σε XDP επηρεάζονται περισσότερο από το πλήθος των στοιχείων που αποθηκεύουν σε δομές μνήμης, παρά από επιπρόσθετες ενέργειες σχετικά με ανάγνωση επικεφαλίδων. Έτσι, οι προτεινόμενοι μηχανισμοί είχαν ισοδύναμη και καλύτερη επίδοση από αντίστοιχους που βασίζονται στις διευθύνσεις IP των επιτιθέμενων. Η επίδοση πραγματοποιήθηκε τόσο την εξαγωγή δεδομένων όσο και την αποκοπή επιθέσεων και τα αποτελέσματα αποδίδονται στις σημαντικά λιγότερες μοναδικές περιγραφές πακέτων σε σχέση με το πλήθος των πηγών.

Τα προηγούμενα κεφάλαια εξερευνούν το φαινόμενο των κυβερνοεπιθέσεων από διάφορες οπτικές γωνίες, όπου κάθε μια παρουσιάζει ενδιαφέρουσες ιδιαιτερότητες και δυσκολίες. Συνεπώς, στη συνέχεια αναφέρονται πιθανές μελλοντικές επεκτάσεις.

Ξεκινώντας από το τελευταίο κεφάλαιο, ενδιαφέρον θα παρουσίαζε μια εκτενής μελέτη μηχανισμών ανίχνευσης επιθέσεων και ταξινόμησης της κίνησης, εστιάζοντας σε σύγχρονους τύπους επιθέσεων. Κατά πάσα πιθανότητα ο κάθε τύπος μπορεί να χρειάζεται εξειδικευμένη μελέτη και βελτιστοποίηση των παραμέτρων για τους σχετικούς αλγόριθμους, π.χ. μηχανική μάθηση, ανάλυση χρονοσειρών ή/και στατιστικά μοντέλα. Ένα σχετικό πρόβλημα είναι η σημαντικά περιορισμένη πρόσβαση σε πραγματικά δεδομένα κυρίως για λόγους προστασίας προσωπικών δεδομένων και απορρήτου επικοινωνίας. Παραδοσιακά, τέτοιες καταστάσεις αντιμετωπίζονται με τεχνικές ανωνυμοποίησης των διευθύνσεων IP και περικοπής των πακέτων ώστε να μην περιέχουν τυχόν ευαίσθητες πληροφορίες. Εναλλακτικά, ο διαμοιρασμός στατιστικών στοιχείων της κίνησης δεν παρουσιάζει τέτοια προβλήματα. Αυτές οι πληροφορίες θα μπορούσαν να χρησιμοποιηθούν για την δημιουργία συνθετικών αλλά αληθοφανών δεδομένων δικτυακής κίνησης που θα ανταποκρίνονται στην πραγματικότητα. Πιθανά οφέλη είναι βεβαίως η χρήση για την αξιολόγηση και βελτίωση μηχανισμών άμυνας.

Επιπρόσθετα, ενδιαφέρον πεδίο είναι η μελέτη τεχνικών εξόρυξης και επεξεργασίας δεδομένων για την δημιουργία περιγραφών επίθεσης. Αυτές μπορούν να βασίζονται σε παρελθοντικά δεδομένα και να διαμοιράζονται μεταξύ συνεργαζόμενων εταίρων (π.χ. ομοσπονδίες ακαδημαϊκών φορέων και παρόχων δικτυακών υπηρεσιών). Συνεργατικά σχήματα μπορούν να υπάρξουν και κατά τη δημιουργία των περιγράφων π.χ. με τεχνικές Ομόσπονδης Μάθησης (Federated Learning), οι οποίες επιτρέπουν την εκπαίδευση ενός κοινού μοντέλου, αμβλύνοντας προβληματισμούς περί προσωπικών δεδομένων και ιδιωτικότητας.

Κοινά μοντέλα και περιγραφές για κάθε τύπο επίθεσης μπορούν να ενισχύσουν σημαντικά μηχανισμούς αντιμετώπισης, ακόμα και να οδηγήσουν στην προληπτική αποκοπή επιθέσεων με κοινά χαρακτηριστικά. Χαρακτηριστικό παράδειγμα είναι ονόματα DNS (domains) τα οποία συχνά χρησιμοποιούνται για επιθέσεις. Ωστόσο, σημαντική είναι και η ομαδοποίηση των περιγραφών με στόχο την βέλτιστη υλοποίηση μηχανισμών αποκοπής στο επίπεδο δεδομένων είτε πρόκειται για λύσεις τύπου P4 (υλικό) είτε για XDP (λογισμικό και υλικό). Συνεπώς, ιδιαίτερο ενδιαφέρον παρουσιάζουν σχετικές τεχνικές επεξεργασίας και ομαδοποίησης.

Σημειώνεται πως, παρότι οι προσεγγίσεις που βασίζονται σε διακριτές περιγραφές φαίνονται ιδανικές, υπάρχουν επιθέσεις που απαιτούν εναλλακτικές προσεγγίσεις. Χαρακτηριστικό παράδειγμα είναι οι επιθέσεις TCP SYN, που σχεδόν πάντοτε βασίζονται σε ψευδεπίγραφες διευθύνσεις IP. Σχετικές λύσεις αρχικά επιχειρούν να επιβεβαιώσουν τη γνησιότητα κάθε πηγής (SYN Cookie) και στη συνέχεια επιτρέπουν την επικοινωνία.

Συνοψίζοντας, υπηρεσίες που αφορούν συλλογή δεδομένων, ανίχνευση και αντιμετώπιση επιθέσεων, μπορούν να μεταφερθούν με ικανοποιητικές επιδόσεις σε προγραμματιζόμενες ενδιάμεσες συσκευές προστατεύοντας πολύτιμους πόρους σε δρομολογητές, κόμβους και firewalls. Οραματιζόμαστε μια πλατφόρμα υπηρεσιών κυβερνοασφάλειας πολλαπλών επιπέδων η οποία (1) ανιχνεύει έγκαιρα τυχόν επιθέσεις συνδυάζοντας πληροφορία από πολλαπλές πηγές (2) εξάγει επιπλέον δεδομένα δικτυακής κίνησης, (3) εντοπίζει το θύμα καθώς τον τύπο της επίθεσης και (4) δημιουργεί με δυναμικό τρόπο μηχανισμούς αποκοπής επιθέσεων, ειδικά κατασκευασμένους για την κάθε περίπτωση. Ιδιαίτερο ενδιαφέρον, θα παρουσίαζε η ενσωμάτωση μιας τέτοιας πλατφόρμας πάνω σε ομόσπονδα σχήματα συνεργασίας ή/και υποδομές NFV όπου εταίροι και χρήστες διαλέγουν/παρέχουν το μηχανισμό αποκοπής προς εγκατάσταση (π.χ. XDP πρόγραμμα).

11 References

[1] S. Jain *et al.*, “B4: Experience with a globally-deployed software defined WAN,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013, doi: 10.1145/2534169.2486019.

[2] N. Shirokov and R. Dasineni, “Open-sourcing Katran, a scalable network load balancer - Facebook Engineering.” available at: https://engineering.fb.com/open-source/open-sourcing-katran-a-scalable-network-load-balancer/ [accessed 28 April 2020].

[3] “SONiC: Software for Open Networking in the Cloud.” available at: https://azure.github.io/SONiC/ [accessed 13 May 2020].

[4] T. Høiland-Jørgensen *et al.*, “The eXpress data path: Fast programmable packet processing in the operating system kernel,” in *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT 2018)*, pp. 54–66, 2018.

[5] P. Bosshart *et al.*, “P4: Programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014, doi: 10.1145/2656877.2656890.

[6] M. Majkowski, “How to drop 10 million packets per second - Cloudflare.” available at: https://blog.cloudflare.com/how-to-drop-10-million-packets/ [accessed 28 April 2020].

[7] G. Bertin, “XDP in practice: integrating XDP in our DDoS mitigation pipeline,” in *NetDev 2.1 - The Technical Conference on Linux Networking*.

[8] “Stratum - Open Networking Foundation.” available at: https://www.opennetworking.org/stratum/ [accessed 13 May 2020].

[9] N. McKeown *et al.*, “OpenFlow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008, doi: 10.1145/1355734.1355746.

[10] M. Jonker, A. Sperotto, R. Van Rijswijk-Deij, R. Sadre, and A. Pras, “Measuring the adoption of DDoS protection services,” in *Proceedings of the 2016 ACM Internet Measurement Conference (IMC 2016)*, pp. 279–285, 2016.

[11] S. Shirali-Shahreza and Y. Ganjali, “Efficient implementation of security applications in OpenFlow controller with FleXam,” in *Proceedings of the 21st IEEE Annual Symposium on High-Performance Interconnects (HOTI 2013)*, pp. 49–54, 2013.

[12] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, “FlowSense: Monitoring Network Utilization with Zero Measurement Cost,” in *Proceedings of the 14th International conference on Passive and Active Measurement (PAM 2013)*, pp. 31–41, 2013.

[13] K. Giotis, G. Androulidakis, and V. Maglaris, “A scalable anomaly detection and

mitigation architecture for legacy networks via an OpenFlow middlebox,” *Security and Communication Networks*, vol. 9, no. 13, pp. 1958–1970, 2016, doi: 10.1002/sec.1368.

[14] “SaltStack.” available at: https://www.saltstack.com/ [accessed 28 April 2020].

[15] A. Doria *et al.*, “RFC 5810 - Forwarding and Control Element Separation (ForCES) Protocol Specification.” available at: https://tools.ietf.org/html/rfc5810 [accessed 15 May 2020].

[16] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, “Ethane,” *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, pp. 1–12, 2007, doi: 10.1145/1282427.1282382.

[17] “OpenDaylight.” available at: https://www.opendaylight.org/ [accessed 28 April 2020].

[18] “ONOS.” available at: https://onosproject.org/ [accessed 28 April 2020].

[19] “Ryu.” available at: https://osrg.github.io/ryu/ [accessed 28 April 2020].

[20] “NAPALM (Network Automation and Programmability Abstraction Layer with Multivendor support).” available at: https://github.com/napalm-automation/napalm [accessed 28 April 2020].

[21] “OpenConfig.” available at: http://openconfig.net/ [accessed 28 April 2020].

[22] ONF, “OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06),” 2015.

[23] C. Y. Hong *et al.*, “B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google’s Software-Defined WAN,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2018)*, pp. 74–87, 2018.

[24] P. L. Ventre *et al.*, “Deploying SDN in GÉANT production network,” in *Proceeding of the 2017 IEEE Conference on Network Function Virtualization and Software Defined Networking (NFV-SDN 2017)*, pp. 1–2, 2017.

[25] “Tofino 2 - Barefoot.” available at: https://www.barefootnetworks.com/products/brief-tofino-2/ [accessed 28 April 2020].

[26] “Agilio CX SmartNICs - Netronome.” available at: https://www.netronome.com/products/agilio-cx/ [accessed 28 April 2020].

[27] “NetFPGA.” available at: https://netfpga.org/ [accessed 02 May 2020].

[28] “In-band Network Telemetry.” available at: https://p4.org/assets/INT-current-spec.pdf [accessed 15 May 2020].

[29] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé, “NetPaxos: Consensus at network speed,” in *Proceedings of the 2015 ACM Symposium on SDN Research (SOSR 2015)*, pp. 1–7, 2015.

[30] A. Sapio, I. Abdelaziz, A. Aldilaijan, M. Canini, and P. Kalnis, “In-network computation is a dumb idea whose time has come,” in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets 2017)*, pp. 150–156, 2017.

[31] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, “Heavy-hitter detection entirely in the data plane,” in *Proceedings of the 2017 ACM Symposium on SDN Research (SOSR 2017),* pp. 164–176, 2017.

[32] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, “Network-Wide Heavy Hitter Detection with Commodity Switches,” in *Proceedings of the 2018 ACM Symposium on SDN Research (SOSR 2018),* pp. 1–7, 2018.

[33] A. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, “Offloading real-time DDoS attack detection to programmable data planes,” in *Proceeding of the 2019 IFIP/IEEE Symposium on Integrated Network Management (IM 2019),* pp. 19–27, 2019.

[34] “P4 16 Language Specification version 1.2.0.” available at: http://p4.org [accessed 28 April 2020].

[35] “P4 Language Specification version 1.0.5,” 2018. available at: http://p4.org [accessed 28 April 2020].

[36] C. Kim, “SLIDES: P4 Language Tutorial,” 2017. available at: http://p4.org [accessed 28 April 2020].

[37] “Ansible.” available at: https://www.ansible.com/ [accessed 28 April 2020].

[38] J. Case, M. Fedor, M. Schoffstall, and J. Davin, “RFC 1157 - Simple Network Management Protocol (SNMP).” available at: https://tools.ietf.org/html/rfc1157 [accessed 30 May 2020].

[39] M. Rose and K. McCloghrie, “RFC 1155 - Structure and identification of management information for TCP/IP-based internets.” available at: https://tools.ietf.org/html/rfc1155 [accessed 30 May 2020].

[40] M. Bjorklund, “RFC 7950 - The YANG 1.1 Data Modeling Language.” available at: https://tools.ietf.org/html/rfc7950 [accessed 30 May 2020].

[41] “Openstack - Build the future of Open Infrastructure.” available at: https://www.openstack.org/ [accessed 28 April 2020].

[42] “Puppet.” available at: https://puppet.com/ [accessed 28 April 2020].

[43] “Network Functions Virtualisation (NFV); Architectural Framework Group Specification - ETSI GS NFV 002 V1.1.1,” ETSI, 2013.

[44] “OPNFV.” available at: https://www.opnfv.org/ [accessed 28 April 2020].

[45] “ONAP.” available at: https://www.onap.org/ [accessed 28 April 2020].

[46] “OSM.” available at: https://osm.etsi.org/ [accessed 28 April 2020].

[47] “CORD - ONF.” available at: https://www.opennetworking.org/cord/ [accessed 31 May 2020].

[48] M. Majkowski, “Why we use the Linux kernel’s TCP stack - Cloudflare.” available at: https://blog.cloudflare.com/why-we-use-the-linux-kernels-tcp-stack/ [accessed 16 May 2020].

[49] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, “Extending Networking into the Virtualization Layer,” in *8th ACM Workshop on Hot Topics in Networks (HotNets 2009),* 2009.

[50] “Contrail Virtual Router - Juniper.” available at: https://github.com/Juniper/contrail-vrouter [accessed 28 April 2020].

[51] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, “The click modular router,” *ACM Transactions on Computer Systems (TOCS),* vol. 18, no. 3, pp. 263–297, 2000, doi: 10.1145/319344.319166.

[52] L. Rizzo, “NetMap: A novel framework for fast packet I/O,” in *Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC 2012),* pp. 101–112, 2012.

[53] “PF_RING – ntop.” available at: https://www.ntop.org/products/packet-capture/pf_ring/ [accessed 28 April 2020].

[54] “DPDK.” available at: https://www.dpdk.org/ [accessed 28 April 2020].

[55] J. Martins *et al.*, “ClickOS and the Art of Network Function Virtualization,” in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014),* pp. 459–473, 2014.

[56] J. Hwang, K. K. Ramakrishnan, and T. Wood, “NetVM: High performance and flexible networking using virtualization on commodity platforms,” in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014),* pp. 445–458, 2014.

[57] P. Phaal and M. Lavine, “sFlow Version 5.” available at: https://sflow.org/sflow_version_5.txt [accessed 29 April 2020].

[58] A. Douitsis and V. Maglaris, “Towards a scalable management collector,” in *Proceedings of the 2016 Global Information Infrastructure and Networking Symposium (GIIS 2016),* pp. 1–6, 2016.

[59] “Model Driven Telemetry - Cisco.” available at: https://www.cisco.com/c/en/us/solutions/service-provider/cloud-scale-networking-solutions/model-driven-telemetry.html [accessed 29 April 2020].

[60] “Overview of the Junos Telemetry Interface - TechLibrary - Juniper Networks.” available at: https://www.juniper.net/documentation/en_US/junos/topics/concept/junos-telemetry-interface-oveview.html [accessed 29 April 2020].

[61] M. Hira and L. Wobker, “Improving Network Monitoring and Management with Programmable Data Planes.” available at: https://p4.org/p4/inband-network-telemetry/ [accessed 29 April 2020].

[62] J. Vestin, A. Kassler, D. Bhamare, K.-J. Grinnemo, J.-O. Andersson, and G. Pongracz, “Programmable Event Detection for In-Band Network Telemetry,” 2019. available at: http://arxiv.org/abs/1909.12101 [accessed 29 April 2020].

[63] E. B. Claise, “RFC 3954 - Cisco Systems NetFlow Services Export Version 9,” Oct. 2004. available at: https://tools.ietf.org/html/rfc3882 [accessed 29 April 2020].

[64] A. Santos Da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, “ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN,” in *Proceedings of 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS 2016),* pp. 27–35, 2016.

[65] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: Scaling Flow Management forHigh-Performance Networks,” in *Proceedings of the 2011 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2011),* pp. 254–265, 2011.

[66] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments,” *Computer Networks*, vol. 62, pp. 122–136, 2014, doi: 10.1016/j.bjp.2013.10.014.

[67] T. Yang *et al.*, “Elastic sketch: Adaptive and fast network-wide measurements,” in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2018)*, pp. 561–575, 2018.

[68] “nProbe – ntop.” available at: https://www.ntop.org/products/netflow/nprobe/ [accessed 29 April 2020].

[69] G. Gardikis *et al.*, “An integrating framework for efficient NFV monitoring,” in *Proceedings of the 2016 IEEE Conference on Network Softwarization (NetSoft 2016)*, pp. 1–5, 2016.

[70] D. Palmisano *et al.*, “D-STREAMON - NFV-Capable Distributed Framework for Network Monitoring,” in *Proceedings of the 29th International Teletraffic Congress (ITC 2017)*, pp. 30–35, 2017.

[71] M. A. Kourtis *et al.*, “Enhancing VNF performance by exploiting SR-IOV and DPDK packet processing acceleration,” in *Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Networking (NFV-SDN 2015)*, pp. 74–78, 2015.

[72] A. TaheriMonfared and C. Rong, “Multi-tenant network monitoring based on software defined networking,” in *Proceedings of OTM Confederated International Conferences “On the Move to Meaningful Internet Systems,”* pp. 327–341, 2013.

[73] “Amazon CloudWatch - Application and Infrastructure Monitoring.” available at: https://aws.amazon.com/cloudwatch/ [accessed 29 April 2020].

[74] “Azure Monitor overview | Microsoft Docs.” available at: https://docs.microsoft.com/en-us/azure/azure-monitor/overview [accessed 29 April 2020].

[75] “Stackdriver | Google Cloud.” available at: https://cloud.google.com/stackdriver/

[accessed 29 April 2020].

[76] "Cloud Monitoring as a Service | Datadog." available at: https://www.datadoghq.com/ [accessed 29 April 2020].

[77] "2019 Data Breaches: 4 Billion Records Breached So Far | Norton." available at: https://us.norton.com/internetsecurity-emerging-threats-2019-data-breaches.html [accessed 29 April 2020].

[78] M. Kührer, C. Rossow, and T. Holz, "Paint it black: Evaluating the effectiveness of malware blacklists," in *Proceedings of the 2014 International Workshop on Recent Advances in Intrusion Detection,* pp. 1–21, 2014.

[79] F. J. Ryba, M. Orlinski, M. Wählisch, C. Rossow, and T. C. Schmidt, "Amplification and DRDoS Attack Defense - A Survey and New Perspectives," 2015. available at: http://arxiv.org/abs/1505.07892 [accessed 29 April 2020].

[80] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *2014 Symposium on Network and Distributed System Security (NDSS 2014),* 2014.

[81] M. Kuhrer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a Handshake : Abusing TCP for Reflective Amplification DDoS Attacks," in *8th USENIX Workshop on Offensive Technologies (WOOT 2014),* 2014.

[82] M. Majkowski, "Reflections on reflection (attacks) - Cloudflare." available at: https://blog.cloudflare.com/reflections-on-reflections/ [accessed 29 April 2020].

[83] M. Majkowski, "Memcrashed - Major amplification attacks from UDP port 11211 - Cloudflare." available at: https://blog.cloudflare.com/memcrashed-major-amplification-attacks-from-port-11211/ [accessed 29 April 2020].

[84] C. Morales, "NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack ; The Terabit Attack Era Is Upon Us." available at: https://www.netscout.com/blog/asert/netscout-arbor-confirms-17-tbps-ddos-attack-terabit-attack-era [accessed 29 April 2020].

[85] "KrebsOnSecurity Hit With Record DDoS — Krebs on Security." available at: https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/ [accessed 29 April 2020].

[86] "Cyber attack hits Danish rail network - The Local." available at: https://www.thelocal.dk/20180514/cyber-attack-hits-danish-rail-network [accessed 29 April 2020].

[87] M. Prince, "The DDoS That Knocked Spamhaus Offline (And How We Mitigated It) - Cloudflare." available at: https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho/ [accessed 15 May 2020].

[88] Arbor NETSCOUT, "Threat Intelligence Report -Powered by ATLAS Findings from First Half 2019," 2019. available at: https://www.netscout.com/sites/default/files/2019-07/SECR_010_EN-1901 – NETSCOUT Threat Report 1H 2019 – Web.pdf [accessed 25

May 2020].

[89] E. Osterweil, A. Stavrou, and L. Zhang, “20 Years of DDoS: a Call to Action,” 2019. available at: http://arxiv.org/abs/1904.02739 [accessed 25 May 2020].

[90] J. Mirkovic and P. Reiher, “A taxonomy of DDoS attack and DDoS defense mechanisms,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004, doi: 10.1145/997150.997156.

[91] S. M. Specht and R. B. Lee, “Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures,” *International Workshop on Security in Parallel and Distributed Systems*, no. 9, pp. 543–550, 2004, doi: 10.1.1.133.4566.

[92] S. T. Zargar, J. Joshi, and D. Tipper, “A survey of defense mechanisms against distributed denial of service (DDOS) flooding attacks,” *IEEE Communications Surveys and Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013, doi: 10.1109/SURV.2013.031413.00127.

[93] Y. Gu, A. McCallum, and D. Towsley, “Detecting anomalies in network traffic using maximum entropy estimation,” in *Proceedings of the 2005 ACM Internet Measurement Conference (IMC 2005)*, pp. 345–350, 2005.

[94] G. Androulidakis, V. Chatzigiannakis, and S. Papavassiliou, “Network Anomaly Detection and Classification via Opportunistic Sampling,” *IEEE Network*, vol. 23, no. 1, pp. 6–12, 2009, doi: 10.1109/MNET.2009.4804318.

[95] “Snort - Network Intrusion Detection & Prevention System.” available at: https://snort.org/ [accessed 15 May 2020].

[96] “Suricata | Open Source IDS / IPS / NSM engine.” available at: https://suricata-ids.org/ [accessed 15 May 2020].

[97] “The Zeek Network Security Monitor.” available at: https://www.zeek.org/ [accessed 15 May 2020].

[98] V. Paxson, “Bro: A system for detecting network intruders in real-time,” *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, 1999, doi: 10.1016/S1389-1286(99)00112-7.

[99] T. T. T. Nguyen and G. Armitage, “A survey of techniques for internet traffic classification using machine learning,” *IEEE Communications Surveys and Tutorials*, vol. 10, no. 4, pp. 56–76, 2008, doi: 10.1109/SURV.2008.080406.

[100] D. Berman, A. Buczak, J. Chavis, and C. Corbett, “A Survey of Deep Learning Methods for Cyber Security,” *Information*, vol. 10, no. 4, p. 122, 2019, doi: 10.3390/info10040122.

[101] C. Siaterlis and V. Maglaris, “Detecting DDoS attacks using a multilayer Perceptron classifier,” in *Proceedings of the 2004 International conference on Artificial neural networks (ICANN 2004)*, pp. 1–14, 2004.

[102] R. Braga, E. Mota, and A. Passito, “Lightweight DDoS flooding attack detection using NOX/OpenFlow,” in *Proceedings of the 2010 IEEE Local Computer Network Conference (LCN 2010)*, pp. 408–415, 2010.

[103] Y. Cui *et al.*, “SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks,” *Journal of Network and Computer Applications*, vol. 68, pp. 65–79, 2016, [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804516300480.

[104] Q. Niyaz, W. Sun, and A. Y. Javaid, “A Deep Learning Based DDoS Detection System in Software-Defined Networking (SDN).” available at: https://arxiv.org/pdf/1611.07400.pdf [accessed 23 May 2020].

[105] X. Yuan, C. Li, and X. Li, “DeepDefense: Identifying DDoS Attack via Deep Learning,” in *Proceedings of the 2017 International Conference on Smart Computing (SMARTCOMP 2017)*, pp. 1–8, 2017.

[106] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del-Rincon, and D. Siracusa, “LUCID: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection,” *IEEE Transactions on Network and Service Management (Early Access)*, pp. 1–14, 2020, [Online]. Available: https://ieeexplore.ieee.org/document/8984222/.

[107] “The CAIDA Anonymized Internet Traces 2016 Dataset.” available at: http://www.caida.org/data/passive/passive_2016_dataset.xml [accessed 15 May 2020].

[108] K. Cho, K. Mitsuya, and A. Kato, “Traffic data repository at the WIDE project,” in *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC 2000)*, 2000.

[109] J. J. Santanna *et al.*, “Booters - An analysis of DDoS-as-a-service attacks,” in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, pp. 243–251, 2015.

[110] D. Turk, “RFC 3882 - Configuring BGP to Block Denial-of-Service Attacks.” available at: https://tools.ietf.org/html/rfc3882 [accessed 15 May 2020].

[111] C. Dietzel, A. Feldmann, and T. King, “Blackholing at IXPs: On the Effectiveness of DDoS Mitigation in the Wild,” in *Proceedings of the 17th International Conference on Passive and Active Network Measurement (PAM 2016)*, pp. 319–332, 2016.

[112] V. Giotsas, G. Smaragdakis, C. Dietzel, P. Richter, A. Feldmann, and A. Berger, “Inferring BGP blackholing activity in the internet,” in *Proceedings of the ACM Internet Measurement Conference (IMC 2017)*, pp. 1–14, 2017.

[113] W. Kumari and D. McPherson, “RFC 5635 - Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF).” available at: https://tools.ietf.org/html/rfc5635 [accessed 15 May 2020].

[114] “RFC 3704 - Ingress Filtering for Multihomed Networks.” available at: https://tools.ietf.org/html/rfc3704 [accessed 15 May 2020].

[115] “RFC 5575 - Dissemination of Flow Specification Rules.” available at: https://tools.ietf.org/html/rfc5575 [accessed 15 May 2020].

[116] “Firewall on Demand - GRNET.” available at: https://grnet.gr/en/services/internet-services/firewall-on-demand/ [accessed 15 May 2020].

[117] “MANRS – Mutually Agreed Norms for Routing Security.” available at: https://www.manrs.org/ [accessed 15 May 2020].

[118] B. Rashidi, C. Fung, and E. Bertino, “A Collaborative DDoS Defence Framework Using Network Function Virtualization,” *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, pp. 2483–2497, 2017, doi: 10.1109/TIFS.2017.2708693.

[119] K. Giotis, M. Apostolaki, and V. Maglaris, “A reputation-based collaborative schema for the mitigation of distributed attacks in SDN domains,” in *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)*, pp. 495–501, 2016.

[120] “RFC 7970 - The Incident Object Description Exchange Format Version 2.” available at: https://tools.ietf.org/html/rfc7970 [accessed 15 May 2020].

[121] A. Jøsang and R. Ismail, “The Beta Reputation System,” in *Proceedings of the 15th Bled Electronic Commerce Conference*, pp. 2502–2511, 2002.

[122] “3DCoP: DDoS Defense for a Community of Peers - Galois, Inc.” available at: https://galois.com/project/3dcop-ddos-defense/ [accessed 15 May 2020].

[123] “DDoS Open Threat Signaling (dots).” available at: https://datatracker.ietf.org/wg/dots/about/ [accessed 15 May 2020].

[124] O. O. Malomo, D. B. Rawat, and M. Garuba, “Next-generation cybersecurity through a blockchain-enabled federated cloud framework,” *Journal of Supercomputing*, vol. 74, no. 10, pp. 5099–5126, 2018, doi: 10.1007/s11227-018-2385-7.

[125] K. Kim, Y. You, M. Park, and K. Lee, “DDoS Mitigation: Decentralized CDN Using Private Blockchain,” in *Proceedings of the 2018 International Conference on Ubiquitous and Future Networks (ICUFN 2018)*, pp. 693–696, 2018.

[126] “Decentralized CDN, WAF, and DDoS protection.” available at: https://gladius.io/ [accessed 15 May 2020].

[127] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, and B. Stiller, “A Blockchain-Based Architecture for Collaborative DDoS Mitigation with Smart Contracts,” in *Proceedings of the 11th IFIP International Conference on Autonomous Infrastructure, Management and Security (AIMS 2017)*, pp. 16–29, 2017.

[128] A. Gruhler, B. Rodrigues, and B. Stiller, “A Reputation Scheme for a Blockchain-based Network Cooperative Defense,” in *Proceedings of the 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, pp. 71–79, 2019.

[129] S. Mannhart, B. Rodrigues, E. Scheid, S. S. Kanhere, and B. Stiller, “Toward Mitigation-

as-a-Service in Cooperative Network Defenses," in *Proceedings of the 16th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2018)*, pp. 362–367, 2018.

[130] V. Maglaris *et al.*, "Toward a holistic federated future internet experimentation environment: The experience of NOVI research and experimentation," *IEEE Communications Magazine*, vol. 53, no. 7, pp. 136–144, 2015, doi: 10.1109/MCOM.2015.7158277.

[131] J. M. Alcaraz Calero and J. G. Aguado, "MonPaaS: An adaptive monitoring platform as a service for cloud computing infrastructures and services," *IEEE Transactions on Services Computing*, vol. 8, no. 1, pp. 65–78, 2015, doi: 10.1109/TSC.2014.2302810.

[132] V. Bajpai and J. Schonwalder, "A survey on internet performance measurement platforms and related standardization efforts," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 3, pp. 1313–1341, 2015, doi: 10.1109/COMST.2015.2418435.

[133] Y. Shavitt and U. Weinsberg, "Quantifying the importance of vantage points distribution in internet topology measurements," in *Proceedings of the 2009 IEEE International Conference on Computer Communications (INFOCOM 2009)*, pp. 792–800, 2009.

[134] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *Proceedings of the 2002 IEEE International Conference on Computer Communications (INFOCOM 2002)*, pp. 618–627, 2002.

[135] N. Chatzis, G. Smaragdakis, J. Böttger, T. Krenc, and A. Feldmann, "On the benefits of using a large IXP as an Internet vantage point," in *Proceedings of the 2013 ACM Internet Measurement Conference (IMC 2013)*, pp. 333–346, 2013.

[136] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM SIGMOD Record*, vol. 26, no. 1, pp. 65–74, 1997, doi: 10.1145/248603.248616.

[137] "Kubernetes." available at: https://kubernetes.io/ [accessed 02 December 2019].

[138] V. Koukis, C. Venetsanopoulos, and N. Koziris, "~Okeanos: Building a cloud, cluster by cluster," *IEEE Internet Computing*, vol. 17, no. 3, pp. 67–71, 2013, doi: 10.1109/MIC.2013.43.

[139] A. TaheriMonfared, T. W. Wlodarczyk, and C. Rong, "Real-time handling of network monitoring data using a data-intensive framework," in *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2013)*, pp. 258–265, 2013.

[140] D. Sarlis, N. Papailiou, I. Konstantinou, G. Smaragdakis, and N. Koziris, "Datix: A system for scalable network analytics," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 21–28, 2015, doi: 10.1145/2831347.2831351.

[141] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster

computing with working sets.," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud 2010)*, 2010.

[142] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," in *Proceedings of the 2011 ACM SIGMOD Workshop on Networking Meets Databases*, pp. 1–7, 2011.

[143] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 217–228, 2005, doi: 10.1145/1090191.1080118.

[144] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards Understanding the Performance of P4 Programmable Hardware," in *Proceedings of the 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2019)*, pp. 1–6, 2019.

[145] O. Hohlfeld, J. Krude, J. H. Reelfs, J. Ruth, and K. Wehrle, "Demystifying the Performance of XDP BPF," in *Proceddings of the 2019 IEEE Conference on Network Softwarization (NetSoft 2019)*, pp. 208–212, 2019.

[146] J. Hill, M. Aloserij, and P. Grosso, "Tracking Network Flows with P4," in *Proceedings of the 2018 IEEE/ACM Innovating the Network for Data-Intensive Science Conference (INDIS 2018)*, pp. 23–32, 2018.

[147] R. Sadre, A. Sperotto, and A. Pras, "The effects of DDoS attacks on flow monitoring applications," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pp. 269–277, 2012.

[148] H. Liu, Y. Sun, and M. S. Kim, "Fine-grained DDoS detection scheme based on bidirectional count sketch," in *Proceedings of the 20th IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*, pp. 1–6, 2011.

[149] "DDoS Attack Glossary: Top 12 Attack Vectors - CPO Magazine." available at: https://www.cpomagazine.com/cyber-security/ddos-attack-glossary-top-12-attack-vectors/ [accessed 16 May 2020].

[150] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970, doi: 10.1145/362686.362692.

[151] "Monitoring Tools - GRNET." available at: https://mon.grnet.gr/ [accessed 16 May 2020].

[152] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, and M. Bailey, "Bohatei: Flexible and Elastic DDoS Defense," in *Proceedings of the 24th USENIX Conference on Security Symposium (USENIX Security 2015)*, pp. 817–832, 2015.

[153] J. Deng *et al.*, "VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls," in *Proceedings of the 2015 IEEE Conference on Network

Function Virtualization and Software Defined Networking (NFV-SDN 2015), pp. 107–114, 2015.

[154] C. J. Fung and B. McCormick, “VGuard: A distributed denial of service attack mitigation method using network function virtualization,” in *Proceedings of the 11th International Conference on Network and Service Management, CNSM 2015*, pp. 64–70, 2015.

[155] M. Majkowski, “Meet Gatebot - a bot that allows us to sleep.” available at: https://blog.cloudflare.com/meet-gatebot-a-bot-that-allows-us-to-sleep/ [accessed 16 May 2020].

[156] G. T. Ross and R. M. Soland, “A branch and bound algorithm for the generalized assignment problem,” *Mathematical Programming*, vol. 8, no. 1, pp. 91–103, 1975, doi: 10.1007/BF01580430.

[157] F. Soldo, K. Argyraki, and A. Markopoulou, “Optimal source-based filtering of malicious traffic,” *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 381–395, 2012, doi: 10.1109/TNET.2011.2161615.

[158] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, “Branch-and-price: Column generation for solving huge integer programs,” *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998, doi: 10.1287/opre.46.3.316.

[159] M. O’sullivan, Q.-S. Lim, C. Walker, I. Dunning, and S. Mitchell, “Dippy-a simplified interface for advanced mixed-integer programming,” 2011.

[160] A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010, doi: 10.1016/j.patrec.2009.09.011.

[161] K. Lee, J. Kim, K. H. Kwon, Y. Han, and S. Kim, “DDoS attack detection method using cluster analysis,” *Expert Systems with Applications*, vol. 34, no. 3, pp. 1659–1665, 2008, doi: 10.1016/j.eswa.2007.01.040.

[162] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System,” 2014. available at: http://arxiv.org/abs/1407.3561 [accessed 15 May 2020].

[163] “On Public and Private Blockchains - Ethereum.” available at: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ [accessed 15 May 2020].

[164] “Proof-of-Authority Chains · Parity Tech Documentation.” available at: https://wiki.parity.io/Proof-of-Authority-Chains [accessed 15 May 2020].

[165] A. Back *et al.*, “Enabling Blockchain Innovations with Pegged Sidechains,” 2014. available at: https://blockstream.com/sidechains.pdf [accessed 15 May 2020].

[166] “Ethereum.” available at: https://github.com/ethereum/ [accessed 15 May 2020].

[167] R. Van Rijswijk-Deij, G. Rijnders, M. Bomhoff, and L. Allodi, “Privacy-conscious threat intelligence using DNSBLoom,” in *Proceedings of the 2019 IFIP/IEEE*

Symposium on Integrated Network and Service Management (IM 2019), pp. 98–106, 2019.

[168] “Memcached DDoS Attacks: 95,000 Servers Vulnerable to Abuse.” available at: https://www.bankinfosecurity.com/memcached-ddos-attacks-95000-servers-vulnerable-to-abuse-a-10705 [accessed 15 May 2020].

[169] A. Broder and M. Mitzenmacher, “Network applications of bloom filters: A survey,” *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004, doi: 10.1080/15427951.2004.10129096.

[170] “Ethereum Go Implementation.” available at: https://github.com/ethereum/go-ethereum [accessed 15 May 2020].

[171] “Ethereum Network Intelligence API.” available at: https://github.com/cubedro/eth-net-intelligence-api [accessed 15 May 2020].

[172] “Ethereum Network Stats.” available at: https://github.com/cubedro/eth-netstats [accessed 15 May 2020].

[173] “nfdump: Netflow processing tools.” available at: https://github.com/phaag/nfdump [accessed 29 April 2020].

[174] M. Majkowski, “The real cause of large DDoS: IP Spoofing - CloudFlare.” available at: https://blog.cloudflare.com/the-root-cause-of-large-ddos-ip-spoofing/ [accessed 25 May 2020].

[175] L. Breiman, “Random Forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[176] “Novelty and Outlier Detection.” available at: https://scikit-learn.org/stable/modules/outlier_detection.html [accessed 25 May 2020].

[177] P. Mockapetris, “RFC 1035 - Domain Names Implementation and Specification.” available at: https://www.ietf.org/rfc/rfc1035.txt.

[178] M. Singh, M. Singh, and S. Kaur, “10 Days DNS Network Traffic from April-May, 2016,” *Mendeley Data, v1*, 2018, doi: http://dx.doi.org/10.17632/zh3wnddzxy.1.

[179] Q. Yan, M. Wang, W. Huang, X. Luo, and F. R. Yu, “Automatically synthesizing DoS attack traces using generative adversarial networks,” *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 12, pp. 3387–3396, 2019, doi: 10.1007/s13042-019-00925-6.

[180] “National Anti-DDoS-coalition - No More DDoS.” available at: https://www.nomoreddos.org/en/ [accessed 31 May 2020].