ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 37

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

EVOLVING INTRUSION DETECTION AND PREVENTION WITH PROGRAMMABLE DATA PLANES

Presentata da: Amir Al Sadi

Coordinatore Dottorato

Ilaria Bartolini

Supervisore

Marco Prandini

Esame finale anno 2025

Borsa di dottorato del Programma Operativo Nazionale Ricerca e Innovazione 2014-2020 (CCI 2014IT16M2OP005), risorse FSE REACT-EU, Azione IV.4 "Dottorati e contratti di ricerca su tematiche dell'innovazione" e Azione IV.5 "Dottorati su tematiche Green." Codice CUP: J35F21003070006

Abstract

Network monitoring has traditionally been constrained by the infrastructure's inherent limitations, including diverse device capabilities, legacy systems, and the physical separation between data generation and monitoring. These constraints delay threat detection and response, providing adversaries opportunities to exploit network vulnerabilities and exhaust resources. While Software-Defined Networking (SDN) has introduced tools for enriched network visibility, its centralized nature imposes latency, limiting real-time threat mitigation.

This thesis explores the potential of fully programmable P4 data planes to transform network threat detection. By enabling line-rate packet processing and custom pipelines, P4 empowers network engineers to proactively detect threats and react with unprecedented speed. Our work demonstrates the versatility of P4 through frameworks addressing diverse challenges. In Part II, we introduce P4RTHENON, which combines a simple data plane anomaly detection algorithm with control plane refinement to detect and mitigate DDoS attacks. Building on this, we propose an adaptive anomaly detection framework leveraging active learning to iteratively refine machine learning models, seamlessly integrating them into P4 pipelines. Part III investigates Distributed Ledger Technologies (DLTs) for tamper-proof alert dissemination, presenting P-IOTA, a framework linking P4-enabled switches directly to IOTA's ledger, bypassing intermediaries and enhancing trust in alert systems. Part IV explores P4's applications in IIoT networks, presenting use cases such as edge-assisted in-network computing with data integrity and industrial tunneling mechanisms, showcasing P4's ability to secure legacy and resource-constrained systems.

Concluding, the thesis outlines open challenges, such as balancing real-time detection with resource efficiency and extending P4's capabilities for broader system integration, emphasizing its role as a cornerstone for future secure networks.

Contents

Abstract **3**

Preamble **1**

I Background **5**

1 Threat Detection in modern networks **7**
1.1 The evolution of network monitoring 7
1.2 Challenges in Modern Network Monitoring 8
1.3 Pinpointing threats with Software Defined Networking 9
1.4 Open Challenges and Future Directions in SDN Security 10
1.5 Bridging the gap with adversaries using P4 12
1.6 In-network computing to protect vulnerable devices 12
1.7 Tackling relevant security threats with P4 13
1.8 Always keep in mind packet forwarding 14

II Finding threats in the Internet **15**

2 A framework for in-line DDoS detection **17**
2.1 Related Work 19
2.1.1 Monitoring solutions with programmable data planes 20
2.1.2 ML-based monitoring with SDN control planes 22
2.1.3 Data and ML-based control planes interaction 24
2.1.4 Programmable data planes pipeline reconfiguration 25
2.2 P4RTHENON: Monitoring Architecture 26
2.2.1 Coarse-Grained Monitoring vs. Fine-Grained Monitoring . . 27
2.2.2 Main design principles of P4RTHENON 29
2.2.3 Programming and interaction of the architectural elements . 29
2.3 P4RTHENON use case: DDoS detection 30

2.3.1 Asymmetric Count-min Sketch (ACMS) . . . 30
2.3.2 Strategies Description . . . 32
2.3.3 Transition between CGM_DDoS and FGM_DDoS . . . 33
2.3.4 Implementation . . . 34
2.4 Performance Evaluation . . . 38
2.4.1 Evaluation metrics and methodology . . . 39
2.4.2 Description of the testbed environment and parameters . . . 40
2.4.3 CGM_DDoS evaluation . . . 41
2.4.4 FGM_DDoS evaluation . . . 44
2.4.5 Overall evaluation . . . 48
2.4.6 Data plane pipeline reconfiguration evaluation . . . 50
2.5 Discussion . . . 51

3 Continual detection upgrade 53
3.1 The case for continual learning . . . 55
3.2 Our approach . . . 55
3.3 Challenges . . . 57

III Safe distributed alerting with DLTs 59

4 IOTA to safely propagate data plane alerts 61
4.1 Background . . . 63
4.1.1 Distributed Ledger Technology . . . 63
4.1.2 IOTA . . . 64
4.2 Related Work . . . 66
4.2.1 SDN and Blockchain . . . 66
4.2.2 Blockchain Interaction with P4-enabled Switches . . . 68
4.2.3 P4 for Thwarting SYN Flooding Attacks . . . 69
4.2.4 Considerations about P4 Employment in Blockchain Solutions 71
4.3 P-IOTA Architecture . . . 71
4.3.1 IOTA Layer . . . 73
4.3.2 Control Plane . . . 74
4.3.3 Data Plane . . . 78
4.4 Case Study . . . 78
4.4.1 Experimental Setup . . . 79
4.4.2 Experiments . . . 81
4.4.3 Experiment 1: Notify a Detected Attack . . . 82
4.4.4 Experiment 2: Update a Wrong Detection . . . 82
4.4.5 Experiment 3: Collect Alerts . . . 83
4.4.6 Experiment 4: Publish . . . 84

4.4.7 Experiment 5: Subscribe . . . 84
4.4.8 Experiment 6: Packet Filter Installation . . . 84
4.4.9 Time and Computational Analysis . . . 84
4.4.10 Discussion . . . 86
4.5 Discussion . . . 87

IV Shielding IIoT devices from intruders 89

5 In-network computing for enhanced data integrity 91
5.1 Related Work . . . 93
5.1.1 Data Plane Programmability for in-network offloading . . . 94
5.1.2 Service orchestration at the Edge . . . 95
5.2 Orchestrating Services using Programmable Data Planes . . . 97
5.3 Industrial security: a focus on legacy devices . . . 101
5.3.1 Threats on legacy devices: the OPC UA protocol . . . 103
5.3.2 Use case: Remote Maintenance of Industrial Plants . . . 104
5.4 Proof of Concept: Emulation of an Industrial Environment . . . 106
5.4.1 Experimental setup . . . 108
5.4.2 Hashing performance comparison . . . 109
5.5 Evaluation Process and Results . . . 113
5.6 Discussion . . . 121

6 Switch-to-switch IIoT tunneling 123
6.1 Use case scenario . . . 124
6.2 Testbed set-up . . . 126
6.3 Results . . . 129
6.3.1 Overall Performance . . . 130
6.3.2 Encryption and Decryption Overhead . . . 130
6.3.3 Key Exchange Overhead . . . 131
6.4 Discussion . . . 133

V Conclusions and bibliography 135

Challenges and open problems 137

Conclusions 141

Bibliography 145

Preamble

Network monitoring has traditionally been tailored to the specific infrastructure, including its hardware and communication protocols. The diversity of devices in modern networks – ranging from their computational capabilities and supported security or encryption levels to the presence of legacy software – compounds the complexity of creating effective monitoring systems. Furthermore, the physical and logical distance between the networking devices and monitoring control plane has relegated network engineers to late observers of already-in-place attacks. Delayed reaction threats can be a pivotal advantage to adversaries, which can cut off defenders by exhausting the control channels and consequently the networking resources [AVOW18].

The advancements in network softwarization, particularly through Software-Defined Networking [McK09](SDNs), have provided network engineers with new tools to enrich threat monitoring and network visibility. SDNs decouple the control plane from the data plane, enabling more dynamic management of network resources and better situational awareness. However, while powerful, they operate with latency inherent to centralized processing, which limits their effectiveness in real-time threat detection and mitigation. As a result, SDN systems, though transformative, are not immune to the challenges of scaling, precision, and timeliness in threat detection.

The emergence of fully programmable P4 [Bos+14] devices represents a paradigm shift in network security. For the first time, network engineers are empowered to take charge of detection processes within the data plane itself. With line-rate custom processing of packets, P4 implies three significant advantages: (i) the ability to create ad hoc pipelines specifically tailored to detect threats, (ii) the capability to detect threats in real-time as they traverse the network, (iii) complete control over

the pace of monitoring, enabling timely reaction to threats. This thesis explores the potential of P4 programmability to enhance intrusion and threat detection. It addresses critical challenges in network monitoring and proposes novel P4-centered solutions to detect threats or improve the overall system's security.

Part II of this work focuses on using ad hoc data structures to detect network attacks. In Chapter 2, we introduce P4RTHENON, a framework that integrates data plane detection mechanisms with control plane refinement to address DDoS attacks. The data plane employs count-min sketching and sampling/thresholding techniques to identify when an attack begins, while the control plane agent isolates and identifies the subset of traffic that constitutes the attack. In Chapter 3, we build on this foundation by advocating for a dynamic and adaptive approach to anomaly detection. Using active learning, we propose selecting critical data on the data plane and transmitting it to the control plane, where it continually updates a machine learning (ML) model. This refined model is then compiled into a P4 pipeline and redeployed on the data plane. This step is repeated at each substantial model update. While this solution demonstrates significant potential, challenges such as implementing floating-point arithmetic on the data plane and ensuring lossless runtime datapath reconfiguration remain open for further exploration.

Part III of the thesis investigates the role of Distributed Ledger Technologies (DLTs) in supporting network threat detection. Chapter 4 presents P-IOTA, a framework that leverages the IOTA([PL19]) DLT to distribute tamper-proof alerts generated by P4-enabled switches. P-IOTA eliminates the need for complex intermediaries between the monitoring level and the tangle, by directly integrating the data plane with the IOTA layer.

Part IV of the thesis explores security applications of P4 in Industrial Internet of Things (IIoT) networks, presenting practical implementations of in-network computation. Chapter 5 investigates the potential of P4-enabled in-network computing in edge-assisted IIoT environments. We present a framework that deploys pipelines ensuring data integrity through symmetric encryption techniques. This approach highlights the feasibility of moving edge computations closer to the data plane and demonstrates how this can improve security and reduce latency. However, significant challenges remain, such as extending this solution to more diverse IIoT networks and pushing the complexity boundaries of the data plane logic.

Chapter 6 extends this exploration by implementing a switch-to-switch tunneling mechanism specifically designed for industrial environments. This practical demonstration underscores the versatility and effectiveness of P4 in securing IIoT networks without upgrading or replacing outdated devices.

The thesis concludes by discussing the challenges and interesting open problems to address in the future (Part V).

Part I

Background

Chapter 1

Threat Detection in modern networks

The primary objective of network monitoring is to ensure the availability, performance, and security of networks by observing, analyzing, and diagnosing their behavior. Over the decades, network monitoring technologies have evolved to address the increasing complexity and dynamic nature of modern networks. This chapter examines the evolution of network monitoring to detect network threats and the pivotal role of data plane programmability in detecting modern attacks.

1.1 The evolution of network monitoring

Network monitoring has significantly evolved since its inception, adapting to the changing demands of increasingly complex networks. Early approaches [Muu83; Jac88] were basic, providing minimal insights into network behavior and lacking the scalability required for expanding infrastructures. These methods were primarily manual and focused on monitoring connectivity and performance.

With technological advancement came more structured monitoring systems that enabled the collection and analysis of network metrics. These systems [Cis96] improved operational efficiency by providing visibility into the state of devices and traffic flows. However, they were predominantly designed for performance management and often lacked robust mechanisms for identifying and responding

to security threats.

As networks grew in size and speed, the limitations of early methods became apparent, prompting the development of more advanced monitoring techniques. Aggregating and analyzing network data provides valuable insights into traffic patterns and helps detect anomalies indicative of potential threats.

Despite their improvements, these systems faced challenges in addressing the evolving threat landscape. Reliance on predefined rules and static analysis made them less effective in pinpointing threats inside sophisticated crafted attack traffic. This gap highlighted the need for holistic monitoring systems, to detect anomalies at different granularities.

1.2 Challenges in Modern Network Monitoring

The modern network landscape is characterized by:

- **Increased Complexity:** The proliferation of devices, virtualization, and multi-cloud environments has made networks more intricate.

- **Dynamic Traffic Patterns:** Applications like video streaming, Internet of Things (IoT), and real-time communications generate traffic with highly dynamic and unpredictable behaviors.

- **Timely deployment of countermeasures:** Modern attacks (like pulse-wave DDoS [Kie23]) are becoming quicker and hence harder to detect and even so to be mitigated, especially in polling-based monitoring systems.

Traditional monitoring solutions lack the flexibility and programmability to adapt to changing network conditions and emerging threats. Moreover, if the attacker knows the deployed monitoring system (especially if it is static), they can easily infer how to avoid it and craft malicious ad-hoc traffic.

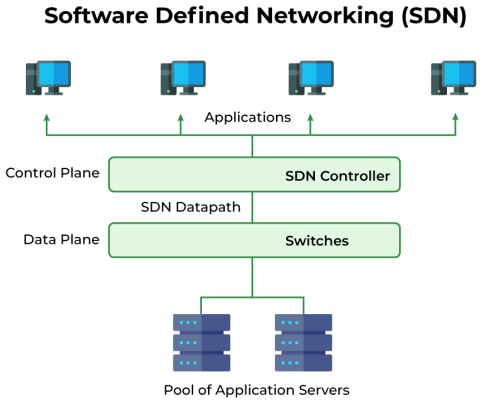


Figure 1.1: Software Defined Networking.

1.3 Pinpointing threats with Software Defined Networking

Software-defined networking [McK09] (SDN) provides means to tackle these challenges. It is an approach to network management that separates the control plane (decision-making) from the data plane (traffic forwarding), as shown in Figure 1.1. This decoupling provides centralized control and programmability of the network.

In the context of threat detection, the data plane can monitor traffic at line rate, allowing quick identification of anomalous attacking patterns. However, its computational capabilities are limited, restricting it to simple tasks like packet counting or basic filtering. Data plane inspection enables real-time processing of packet-level information and in-line monitoring, ideal for fine-grain analysis.

The control plane has greater computational resources and can perform more complex threat analysis, such as correlation across flows or deep packet inspection. Monitoring at this level enables the detection of high-level anomalies, such as abnormal traffic volumes or routing instabilities. However, polling networking data delays threat detection and reaction.

This trade-off highlights the complementary roles of the control and data planes in SDN-based threat detection, balancing speed and analysis depth. Depending on the threat, moving the computation closer to the data plane may be beneficial for prompt detection and reaction.

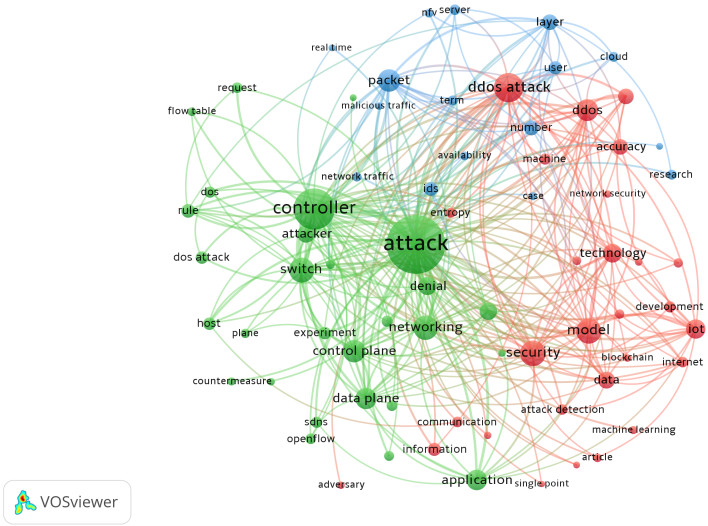


Figure 1.2: Keywords distribution in [Mel+23] articles dataset (image generated with VOS viewer).

1.4 Open Challenges and Future Directions in SDN Security

This section outlines key challenges and future research directions in Software Defined Networking (SDN) security, derived from a systematic study [Mel+23] we carried out. Figure 1.2 provides a visual outlook of the most popular keywords in the article abstracts' dataset: the smaller dots roughly coincide with the less studied topics at the time ($\sim 2015 - 2021$).

SDN's architectural approach introduces inherent vulnerabilities, particularly due to its centralized design. The SDN controller and the control channel are primary targets for security threats. Although current technologies offer potential solutions, a comprehensive analysis of the security challenges posed by SDN is essential for improving defensive measures. Future efforts should aim to address these challenges while leveraging SDN's strengths.

The centralized SDN controller remains a single point of failure. Research should focus on tools that enforce security policies across SDN systems. Data plane programmability can provide mechanisms to monitor the system's health

and enhance the prevention, detection, and mitigation of attacks, or offload some intelligence from the control plane to the data plane.

A lack of standardized threat models tailored to SDN has also been identified. Developing such models would help categorize and address SDN-specific threats, including denial of service (DoS) attacks, facilitating the creation of secure applications and architectural designs. Current research, however, primarily emphasizes prevention and detection, with limited focus on mitigation. Building active defense systems capable of autonomously counteracting attacks in real time represents a critical area for future work.

Moreover, the limited and under-documented range of technologies supporting SDN architecture hinders progress. Expanding the variety of technologies and thoroughly documenting them could enable the design of more secure systems. New technologies may directly address well-known SDN vulnerabilities, enhancing security by design.

Machine Learning (ML) plays a pivotal role in advancing SDN security, particularly in prevention and detection mechanisms. Promising approaches such as Reinforcement Learning (RL) can enable real-time network reconfiguration in response to detected attacks. Adversarial ML is also gaining traction for simulating and stress-testing intrusion detection systems. Future ML applications should expand their scope to include mitigation techniques.

Denial of Service (DoS) attacks remain the most prevalent and damaging threat to SDN environments. Recent threats such as Low-rate DoS (LDoS) and Economic DoS (EDoS) attacks pose significant challenges. From this rises the urge for effective lightweight detection and mitigation strategies.

Finally, distributed ledger technologies (DLTs), including blockchain, offer substantial potential to enhance SDN security. These technologies provide certified models and monitoring layers, ensuring integrity in programmable data plane ecosystems. Further exploration of performance optimization in blockchain-enabled SDN systems is needed.

These challenges emphasize the necessity for continued innovation in SDN security, focusing on architecture resilience, advanced mitigation strategies, and the integration of emerging technologies. This study clearly indicated that programmable data planes provide more scalability and expressiveness than tradi-

tional SDN architectures, allowing for more effective threat detection and prevention.

1.5 Bridging the gap with adversaries using P4

While traditional SDN architectures enable a degree of programmability (due to the use of Openflow [Ope]), emerging technologies like P4 [Bos+14] (Programming Protocol-Independent Packet Processors) take this a step further [But17]. P4 allows developers to define how packets are processed within network devices, providing unprecedented flexibility.

P4 offers significant advantages in network monitoring and threat detection by enabling custom forwarding behaviors and advanced stateful analysis directly in the data plane. It allows programmable control over packet processing, enabling tailored rules to tag suspicious traffic or implement unique detection protocols. By leveraging stateful elements such as registers and counters, P4 can maintain traffic states (e.g., packet rates or byte counts) to quickly detect anomalies without relying on the slower control plane. Techniques like `thresholding`, where metrics like packet rates trigger alerts upon exceeding defined limits, `sketching`, which summarizes traffic using compact data structures to detect patterns or deviations, and `sampling`, which efficiently monitors subsets of traffic, empower P4 to identify threats like DDoS attacks or abnormal flows in real-time. For example, a P4 program could monitor packet rates per source IP and block traffic immediately if it exceeds a threshold, mitigating threats swiftly while minimizing overhead. These capabilities make P4 a powerful tool for timely and efficient threat interception in modern networks.

1.6 In-network computing to protect vulnerable devices

A powerful side-effect brought by data plane programmability and P4 is in-network computing [KT22], where network devices, traditionally limited to forwarding packets, can perform simple computations directly in the data plane. This includes

tasks like simple encryption, hashing, and basic arithmetic operations, which can be particularly beneficial for offloading computation from resource-constrained devices, such as IoT devices or legacy systems in Operational Technology (OT) environments.

These devices often lack the computational power to implement robust security features, leaving them vulnerable to attacks. For instance, many IoT devices natively communicate using lightweight or outdated protocols that do not support encryption or other modern security measures. With P4, network switches or routers can symmetrically encrypt sensitive data fields, verifying packet integrity, or adding authentication tags before forwarding packets.

This reduces the computational burden of IoT devices, enabling them to operate securely without expensive hardware upgrades. By shielding vulnerable devices and enabling real-time deployment of security measures, P4 extends its utility beyond anomaly detection to actively supporting and securing the broader network ecosystem. Moreover, in-network computation in edge-assisted networks [Kon+22] can move the detection closer to the target.

1.7 Tackling relevant security threats with P4

P4 has enabled innovative applications in the data plane, spanning multiple domains of network security and monitoring. In machine learning-based threat detection, P4 can be used to implement lightweight `decision trees` or `random forests` [CSF22] directly in the data plane, enabling fast classification of traffic patterns to identify anomalies or potential threats without involving the control plane. These models can be simply encoded on a P4 pipeline with the use of MATs and ad-hoc processing stages. P4 allows in-network encryption via the implementation of `cryptographic algorithms`, such as AES [Che20], directly on network devices. This facilitates secure data transmission by encrypting sensitive traffic flows in real-time, protecting vulnerable endpoints. Lastly, in stateful data plane threat detection, P4 enables the deployment of `advanced data structures` like packet distributions, count-min sketches, and Bloom filters. These structures allow efficient detection of DDoS attacks or unusual traffic behaviors by detecting significant deviations in traffic volumes [Din+21; GHV21]. These applications

demonstrate the versatility of P4 in enhancing real-time network security through data plane programmability.

1.8 Always keep in mind packet forwarding

While P4 offers complete programmability in the data plane, its primary goal remains high-speed packet forwarding. Indeed, the P4 language operates with limited memory resources, which restricts the size and complexity of data structures that can be maintained. In addition to memory constraints, P4 employs strict limitations on control flow and data types. Constructs like loops, which could lead to variable execution times, are not permitted. Similarly, P4 does not support floating-point operations.

However, adding complex data structures and multiple forwarding stages can still increase the processing overhead, potentially slowing down packet forwarding. For instance, implementing advanced stateful mechanisms involves additional memory lookups and computations, which can accumulate latency, especially under high traffic loads.

These language constraints limit P4 programming to simple and efficient pipelines, with the end goal of ensuring that added functionalities do not slow down packet forwarding. However, trade-offs between programmability and performance should still be part of the design of our P4 monitoring solution.

Part II

Finding threats in the Internet

Chapter 2

A framework for in-line DDoS detection *

In this Part, we are going to analyze how programmable data planes (PDP) can help the control-plane spotting threats by running in-line detection algorithms. Networks are becoming by the day more pervasive in the processes of our daily lives, from work to leisure. This creates unimaginable opportunities but also opens the floor to new threats. For this reason, modern networks should promptly respond to unexpected events to safeguard the running services and avoid service disruption. Thus, to cope with legacy network technologies, modern infrastructures require in-depth and responsive network monitoring, management, and control.

Two key points, also relevant in the 5G world [Gro16], are *(i)* Control and User Plane Separation (CUPS) and *(ii)* network programmability. CUPS improves flexibility and scalability by de-coupling the logic problems from the pure data forwarding issues, while programmability allows networks to react to unwanted situations [Cal+16]. This envisions a *closed-loop approach* where the control plane collects real-time information about the status of the underlying network and reacts, by issuing suitable directives to the data plane, to modify its behavior [Bor+23; Mel+20].

In this chapter, we present P4RTHENON, a viable approach to implement

*Part of the material presented in this chapter is based on the following publications: Amir Al Sadi et al. "Unleashing Dynamic Pipeline Reconfiguration of P4 Switches for Efficient Network Monitoring". In: *IEEE Transactions on Network and Service Management* (2024)

a closed-loop monitoring system, which can intercept network behaviors and take real-time actions. P4RTHENON stems from the idea of minimizing the congestion of the control channel between data and control planes [Zha+18], especially in the occurrence of abnormal behaviors.

We validate our solution by devising a volumetric Distributed Denial of Service (DDoS) attack detection over P4RTHENON, to showcase how we can minimize the impact on the control channel and keep high detection rates. P4RTHENON follows the Software Defined Networking (SDN) paradigm: the detection logic is split between a simple data plane logic and a more complex control plane strategy.

In this latter landscape, the goal of our approach is to achieve the best possible trade-off between monitoring performance, computational complexity, and control channel utilization.

To this aim, P4RTHENON splits the monitoring task into two phases called *coarse-grained* and *fine-grained* monitoring, i.e., *Coarse Grained Monitoring (CGM)* and *Fine Grained Monitoring (FGM)*, respectively. The peculiarity of our approach is that FGM comes into place only when needed after an appropriate runtime *data plane pipeline reconfiguration*. For the sake of the proposed use case we implemented *(i) CGM_DDoS* strategy to detect the traffic portion suspected to belong to a DDoS attack, and *(ii) FGM_DDoS* strategy to deeply analyze the suspect traffic in the control plane and classify it in the right DDoS class if proven to be malicious.

We implemented CGM_DDoS as a simple in-network P4-based solution that calculates the degree of traffic asymmetry between two end hosts A and B in the two directions ($A \rightarrow B$ and $B \rightarrow A$), assuming that traffic is strongly asymmetric when a DDoS attack is in place, and flagging as suspect all the flows characterized by a high asymmetry degree. This strategy, based on a Count-min Sketch [CM05], over-estimates the number of DDoS attack flows, leading to some false positives while keeping the number of false negatives low.

Once suspect flows are identified by CGM_DDoS, P4RTHENON triggers FGM_DDoS *(i)* to extract relevant features from their packets in the data plane and *(ii)* to mirror this data to the control plane in the form of P4 digests. The collected features are then given as input to a trained Convolutional Neural Network (CNN), i.e., LUCID [DC+20], performing ML inference and classifying any suspect flow

as belonging to a DDoS attack class or as benign. According to our results, FGM_DDoS can substantially reduce false positives with respect to CGM_DDoS, thus achieving high Precision while keeping the control channel utilization low.

To summarize, the main contributions of this chapter are:

- A new architecture, i.e., P4RTHENON, to dynamically reconfigure the data plane pipeline at runtime and remodel the control plane accordingly, to minimize the control channel utilization.

- A lightweight P4 strategy to early detect Volumetric DDoS flows, i.e., Asymmetric Count-Min Sketch (ACMS).

- A validation of P4RTHENON to detect volumetric DDoS attacks, which analyzes the tradeoff between memory consumption/control channel occupation and the detection performance, leveraging a state-of-the-art ML agent [DC+20] in the control plane.

The chapter is organized as follows. We start by summarizing the state of the art on existing P4-based (i.e., in-network) and ML-based monitoring solutions (Section 2.1), outlining their limitations. Section 2.2 details the principles of our proposal, P4RTHENON, and its architectural components. Section 2.3 presents the DDoS detection use case, from the scenario to the implementation of a testbed, and the performance of the proposed solution is evaluated in Section 2.4. Finally, we discuss the results in Section 2.5.

2.1 Related Work

In this Section, we analyze the state of the art regarding SDN monitoring solutions. Table 2.1 summarizes them. First, we sum up the existing PDP monitoring solutions. Then, we investigate how ML-based monitoring is exploited in the SDN control plane. We proceed to give an overview of works that integrate data plane and ML-centered control plane solutions for monitoring. We conclude the Section by analyzing the pipeline reconfiguration methods proposed in the literature, to better position our architectural choice.

Category	Works
PDP monitoring	[Din+22], [Tan+21], [Jey+14], [Li+19], [Kat+17], [BB+16], [BB+18], [THL20], [BG+19], [CSF22], [Swa+22], [Qin+20], [Sir+22], [Raz+22]
ML control-plane	[DC+20], [Ddo], [Shi+12]
PDP + ML control-plane	[Zha+20], [Zhe+22], [MGB21], [Mus+22], [Bar+21]
Pipeline reconf.	[Xin+22], [Fen+21]

Table 2.1: Summary of the Related Work.

2.1.1 Monitoring solutions with programmable data planes

Offloading part of the control plane intelligence to the data plane has become increasingly popular in SDN [Lia+23] thanks to the rise of data plane programmability (DPP). DPP opened the field to greater monitoring expressiveness on network devices since it can be leveraged to describe arbitrary, albeit simple packet manipulation strategies on top of regular forwarding. In recent years, programmable data planes have proven to be effective in supporting complex monitoring strategies by coding part of them directly on the data plane [Din+22], most commonly exploiting the P4 language [Con20].

The most straightforward approach showing how DPP can be exploited to support network monitoring is *In-band Network Telemetry* (INT) [Tan+21], a framework proposed by some of the biggest networking companies in conjunction with the P4 Working Group. INT allows gathering monitoring information by transparently adding custom headers to users' packets, which are then extrapolated and forwarded to a centralized collector. INT has been extensively used to support traffic engineering [Jey+14], congestion control [Li+19], and routing [Kat+17].

Another possibility to take advantage of DPP for network monitoring consists in exploiting the stateful memory made available by P4-based programmable data planes (i.e., P4 registers), to implement customized data structures (i.e., sketches [Han+22]) for advanced *in-network monitoring* [Dar+17]. Thanks to these data structures, it is possible to support complex tasks, such as intrusion or anomaly detection, by keeping track of flows' state and aggregate statistics directly in the data plane. For instance, many strategies have been proposed to detect heavy

flows (or *heavy hitters*) [BB+16], using different data structures such as hash tables [BB+18] or invertible sketches [THL20].

Work	ML model	Limitation
[Swa+22]	CNN, DNN, SVM	Model accuracy for DNN below 70%
[BG+19]	RF	Evaluated using an arbitrary F1Score threshold
[CSF22]	RF	Requires a minimum of 63 rules to work
[Qin+20]	BNN	Low Recall in certain datasets
[Sir+22]	DT	Low detection performance in certain datasets
[Raz+22]	DNN	8-bit floating point arithmetic

Table 2.2: Limitations of current P4-based ML data plane solutions.

Recently, some works have also proposed to offload ML inference to the data plane, meaning that the whole ML model is made executable in the data plane pipeline in support of widely different monitoring tasks. For example, pForest [BG+19] and BACKORDERS [CSF22] have proposed to offload a random forest (RF) [Bre01] on the data plane for in-network inference. These works either require a large amount of Match-Action Tables (MAT) installed or use arbitrary F1Score thresholds to rate the detection quality. A further step in this domain has been made by Taurus [Swa+22], which offloads CNN [ON15], Deep Neural Networks (DNN) [Sam+21], and Support Vector Machine (SVM) [SS16] models to the data plane exploiting MapReduce [DG08]. However, Taurus shows limitations already at model-level accuracy, which is below 70% for the DNN, while we could not find accuracy benchmarks for the other two models. Other works, such as [Qin+20; Sir+22; Raz+22], offload different ML models, among which Decision Trees (DT) and Binary Neural Networks (BNN), to the data plane. Especially, DT and BNN exhibit low inference performance. This is because the DT can only have limited depths and BNN adopts simple binary weights to overcome data plane operational limitations. Razavi et al. [Raz+22] implement a DNN, but the authors encoded weights' floating point numbers with 8-bit integers, leading to similar performance degradation as [Qin+20; Sir+22]: this is a major limitation that they also clearly highlight in the paper.

Table 2.2 provides a brief summary of the aforementioned ML-based data plane solutions, all exploiting the P4 language technology, outlining their limitations. To

summarize, trying to fit ML models to the programmable data plane is not simple, requires nontrivial operations to optimize the code and memory consumption, and high inference performance is hard to achieve. These are the reasons why P4RTHENON relies on simple sketch-based strategies in the data plane for CGM, while a ML-based in-depth analysis, as that performed by FGM (see Section 2.1.2), leverages the more powerful control plane computational capabilities.

DDoS attack detection: As specified in the Introduction, in this chapter we focus on DDoS attack detection as a use case. To address this problem, many solutions based on DPP have been proposed [AlS+22]. ML-based methods in the data plane have also been proposed, as BACKORDERS [CSF22] already discussed above. Other works, e.g. [DSS21; SI+20], adopt a coarse-grained strategy by definition, where the data plane is employed as a valid support to roughly detect anomalies. Ding et al. [Din+21] propose INDDoS, a pragmatic way to detect victims targeted by a DDoS attack using Direct Bitmap combined with a Count-min Sketch. P4RTHENON's DDoS detection implementation indeed relates to other P4-based solutions mentioned in the AlSabeh et al. survey [AlS+22]. However, none of the considered schemes provides an in-depth analysis of the memory and control channel utilization when data and control planes interact while ensuring acceptable DDoS detection performance. Moreover, our P4 implementation of asymmetric flow detection is a novel contribution. We argue that our analysis not only validates the scalability of P4RTHENON, but also demonstrates how it is possible to match the detection performance of state-of-the-art solutions while drastically reducing the management overhead. In fact, our proposed CGM_DDoS strategy takes inspiration from the cited works, but it simplifies the strategy even further at the expense of increasing the false positive rate, which is then corrected by FGM_DDoS.

2.1.2 ML-based monitoring with SDN control planes

The effectiveness of ML-based solutions involving the SDN centralized control plane for monitoring has been thoroughly demonstrated [Zha+19]. The most important factors contributing to their success are the following [Xie+18]: *(i)* a single ML model can be deployed on top of the centralized controller to monitor network-

wide scenarios; *(ii)* centralizing data collection is key for precise prediction; *(iii)* relevant data can be retrieved in real-time by the controller. In the following, we will specifically focus on monitoring tasks related to ML-based DDoS attack detection, from which we took inspiration to design FGM_DDoS.

Work	**Model**	**Dataset**	**Features**	**True Pos. (%)**	**Training Time**
[GK18]	SVM	CAIDA [Ddo]	60	87.35%	N/A
[YLL17]	CNN RNN	ISCX [Shi+12]	20	∼98%, unclear	N/A, [DC+20] says $> 25\ h$
[DC+20]	CNN	CAIDA [Ddo], ISCX [Shi+12], and more	12	∼99%	4500 s

Table 2.3: Comparison between ML-based control plane DDoS detection solutions.

DDoS attack detection: A thorough high-level analysis of ML techniques to detect DDoS attacks is proposed by He et al. [HZL17], which outlines detection performance differences when selecting various features and models. This work also suggests that classic ML approaches are usually highly dependent on feature choice and datasets. To better generalize the model and loosen the constraint of selecting a fixed set of features, Deep Learning-based schemes have become extremely popular in detecting DDoS attacks. Among them, Ghanbari et al. [GK18] propose a solution that leverages a CNN, achieving high detection rates on a very well-known dataset, i.e., UNB ISCX intrusion detection evaluation dataset [Shi+12]. DeepDefense [YLL17] is another example that combines a CNN and a Recurrent Neural Network (RNN), evaluated with good performance in the CAIDA DDoS 2007 attack dataset [Ddo]. However, both solutions require a high number of features and do not suit real-time scenarios given their complexity. LUCID, proposed by Doriguzzi et al. [DC+20], adopts similar concepts but in a way that makes the trained ML model suitable for online scenarios. LUCID is a lightweight CNN that classifies each traffic flow as belonging to a known DDoS class or as benign. With a rather fast training phase and a limited number of needed features, LUCID is capable of high detection rates.

Table 2.3 reports a comparison summary of the three works discussed above.

LUCID ensures limited training time while keeping detection performance high, and thus it is a suitable solution to be adopted on top of an SDN control plane. However, it needs to inspect all the network traffic to provide good prediction rates.

Mirroring packets to the control plane during a volumetric attack could congest the control channel [Son+16; DC+24], in fact propagating it. Our proposed FGM_DDoS strategy exploits LUCID to classify network traffic, but it adds a data plane data aggregation logic to relieve the control channel, by delivering to the control plane, via P4 digests, only features extracted from suspect traffic in the data plane.

2.1.3 Data and ML-based control planes interaction

Some works in literature have proposed monitoring architectures envisioning a tight interaction between programmable data and control planes in an SDN environment, with the goal of implementing refined strategies to optimize such an interaction: this is to some extent also the main objective of P4RTHENON. It is important to note that all the previous work on this topic focuses on anomaly/attack detection tasks.

Zhang et al. propose POSEIDON [Zha+20], a framework to map attack countermeasures to the programmable data plane and to servers. They propose a language for hardware abstraction and a runtime environment to orchestrate the real-time reaction to attacks. However, this solution requires multiple technologies and components and is bound to the language proposed by the authors, making it hardly replicable. A general solution that jointly maps ML-assisted detection in a programmable data plane and in a control plane is IIsy [Zhe+22]. Two models are proposed: a lighter one, fully deployed in the data plane, and a heavier one, in the control plane. They intensively tested the deployment of different models in the data plane, but do not consider the possibility of swapping between different configurations at runtime.

ORACLE [MGB21] and the work proposed by Musumeci et al. [Mus+22] focus on two architectural approaches that envision a collaboration between control and programmable data plane to detect DDoS attacks. In both cases, aggregated

statistics from packets' flows are computed by the programmable data plane and forwarded to the control plane, where they are processed by an ML engine to detect attacks through ML inference. Although simple and effective, these solutions require intense and constant communication between the data and the control plane, even when the attack is not happening, as they require the data plane to constantly send the aggregated statistics to the control plane. While taking inspiration from these proposals, our solution is the first attempt known to us to design a two-phase system for the optimization of control channel usage: in-network monitoring is autonomously performed by the programmable data plane to detect suspect traffic, and a more refined ML-based analysis takes place in the control plane. Differently from [MGB21; Mus+22], the latter is performed on categorical features extracted from packets belonging to suspect flows. Extracting packets' categorical features (e.g. IP flags) instead of computing flows aggregated statistics is another peculiarity of our proposal. Thanks to this, multi-class classification can be better performed (e.g. to which type of DDoS attack the packet belongs, if malicious) instead of only performing binary classification (DDoS/benign) as done in [MGB21; Mus+22].

FlowLens [Bar+21] is another work that resembles our solution regarding purpose and scope. Its authors propose an SDN architecture that leverages programmable switches to efficiently support multi-purpose ML-based security applications. FlowLens collects features related to packet distribution at line speed and classifies flows directly in the switches, using their CPU. However, though highly flexible and reliable, FlowLens cannot benefit from the network-wide view provided by a centralized SDN control plane. In addition, it does not envision any data plane pipeline reconfiguration at runtime, as supported by P4RTHENON. Reconfiguring the data plane pipeline makes it possible to install specialized pipelines, instead of using a general-purpose one, and to optimize the amount of data exchanged between data and control plane.

2.1.4 Programmable data planes pipeline reconfiguration

The potential advantages of runtime data plane pipeline reconfiguration have already attracted the attention of the research community. We argue that the work

presented by Xing et al. [Xin+22] is the most convincing attempt to (re)program a switch at runtime. In this work, the authors propose an extension of the P4 language that enables partial reconfiguration of the data plane with minimum resource overhead, without service disruption, and guaranteeing consistent packet processing. By allowing developers to load new features at runtime on a reserved memory area, the authors propose a solution to the notorious problems of repopulating all the existing tables and of the introduced delay when a switch firmware is replaced. This work does not consider any specific application domain and it is not clear what the impact would be if the whole pipeline had to be reconfigured. A similar proposal was advanced by Feng et al. [Fen+21]. The authors designed a specific real-time upgradable architecture called In-situ Programmable Switch Architecture (IPSA). This approach allows to implement more efficient reconfiguration, but at the cost of having to upgrade the whole network to adopt switches whose architecture follows the IPSA one, which may not be feasible in large-scale scenarios.

In contrast to the existing works, P4RTHENON leverages the native feature made available by the P4Runtime specification [Con20] that allows a P4 pipeline reconfiguration at runtime. We chose this approach because P4Runtime is a well-established Application Programming Interface (API) for controlling the data plane elements of a device whose behavior is specified by a P4 program, and thus no architectural change is needed as long as a P4-enabled device is adopted. To the best of our knowledge, the few experiments that we were able to find about partial or total pipeline reconfiguration have never focused on the optimization of the burden on the control channel. Conversely, P4RTHENON is specifically designed to minimize the amount of data exchanged between the involved planes, while ensuring high monitoring performance.

2.2 P4RTHENON: Monitoring Architecture

In this Section, we describe the main concepts behind P4RTHENON, our general-purpose real-time solution to describe and implement monitoring policies. The main goal of P4RTHENON is minimizing the amount of data exchanged between data and control planes while achieving high performance of monitoring engines

running in the control plane that require detailed features extracted from packets, such as ML-based ones.

P4RTHENON executes monitoring tasks in two phases: *(i)* in the first phase, an approximate traffic analysis is performed, which identifies the flows that should be monitored more in depth (i.e., *Coarse-Grained Monitoring*); *(ii)* in the second phase, an accurate analysis is done on the flows selected by CGM, with the aim of further discriminating what flows meet the behavior specified by the monitoring policies (i.e., *Fine-Grained Monitoring*). Each phase is associated with a specific strategy deployed by the control plane, which requires a runtime reconfiguration of the data plane pipeline.

Specifically, CGM is meant to run completely in the data plane, meaning that just a few data points need to be forwarded to the control plane in this phase. Based on the information gathered during CGM, when meeting some pre-defined condition, the control plane triggers FGM, with a consequent reconfiguration of the data plane pipeline. FGM is data-intensive as it requires traffic features to be mirrored from the data plane to the control plane. However, the features mirrored during FGM are only those extracted from flows selected by CGM: this significantly reduces the amount of data to be forwarded and analyzed, enhancing Precision by lowering the input detection noise and reducing the burden on the control channel. Whenever some other pre-defined condition is met (e.g. after a user-defined timeout expiration), P4RTHENON triggers the return to the execution of CGM and the pipeline is reconfigured to the previous status accordingly. Figure 2.1 illustrates the architecture of the system, also showing the specific strategies adopted in our DDoS detection use case, whose design and implementation will be detailed in Section 2.3. In the following we will provide further details on CGM and FGM, on design principles, and on the main enabling technologies.

2.2.1 Coarse-Grained Monitoring vs. Fine-Grained Monitoring

CGM is the default strategy installed in the data plane. It is designed to be executed on top of the regular forwarding with a low-added overhead. It relies on a very simple monitoring strategy that *(i)* continuously monitors the traffic; *(ii)*

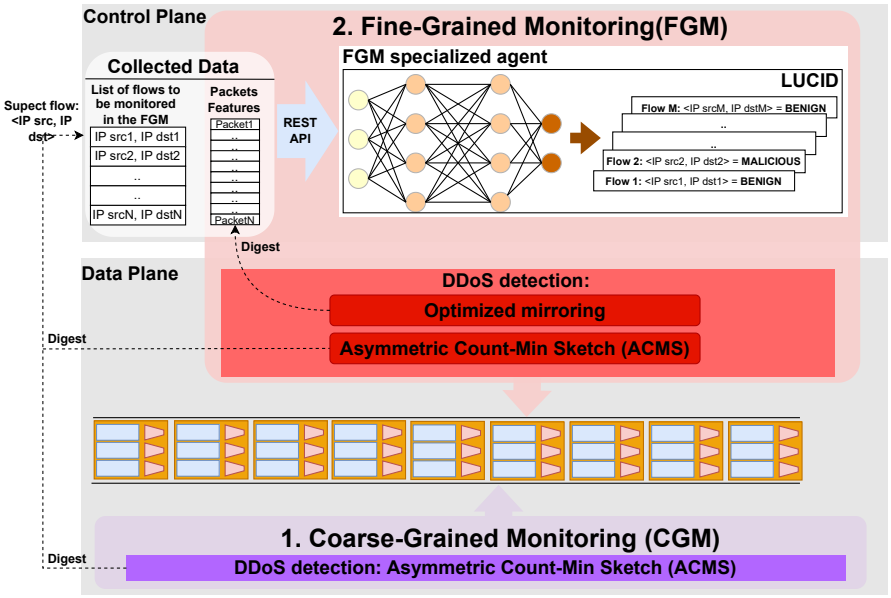


Figure 2.1: P4RTHENON architecture with a focus on the DDoS detection use case (Section 2.3).

identifies the set of flows that meet some monitoring requirements. CGM is fully executed by the programmable data plane, and exchanges minimal data with the control plane. In fact, the data plane occasionally sends management messages to the control plane updating it with a summary of the current network status. Then, the control plane inspects the collected data and, if a condition is met, the execution of FGM is triggered.

FGM is instead deployed by the control plane whenever some traffic needs to be inspected with higher Precision. CGM is responsible for identifying the traffic flows worth being monitored by FGM. Unlike CGM, FGM's logic is evenly split between data and control planes. In the data plane, FGM extracts relevant features (e.g. IP flag, TCP ports, etc.) from packets of selected flows and mirrors them to the control plane. In the control plane, a specialized agent (e.g. a trained ML model) takes as input the extracted features and performs a deeper monitoring (e.g. flow classification).

2.2.2 Main design principles of P4RTHENON

CGM and FGM should be designed to ensure that CGM is able to recognize *all* (ideally) the flows that may need attention, but it could include in such a set also flows that are wrongly selected as interesting. Instead, FGM should be capable of further discriminating, from the set of flows selected by CGM, the flows that are truly relevant. The use case presented later in this chapter (see Section 2.3), which refers to DDoS detection, will show that CGM_DDoS is effective in identifying a superset of flows that belong to DDoS attacks, hence finding all the true positives with a certain degree of false positives, while FGM_DDoS is very efficient at trimming out all the false positives. As far as these design principles are met, P4RTHENON could be adopted for widely different monitoring tasks other than DDoS detection.

2.2.3 Programming and interaction of the architectural elements

The data plane pipeline's behavior is specified by a program written in the *P4 language* [Con21]. Each P4-programmable pipeline consists of a set of processing blocks, which can modify the packet headers and gather packet-related data (e.g. the features required by FGM). As Southbound Interface (SBI) we adopt the well-known *P4Runtime* [Con20], which is exploited to *(i)* install match-action rules (enabling the selective per-flow features' mirroring in FGM) and *(ii)* send data to the control plane (e.g. extracted features) by means of *digest* messages.

The digest is a type of message specified in the P4Runtime specification [Con20] that can be adapted to send one-way data recovered by the data plane to the control plane. As the documentation explains, it differs from *packet-in* messages [Ope] as it is optimized to only send some packet's header fields and metadata, while packet-in is generally used to also send the payload. Multiple digests can be aggregated by P4Runtime into larger messages to reduce their number.

The control plane retrieves the digest data as a *JSON collection*, where each JSON encapsulates a digest associated with a packet. The FGM specialized agent (see Fig. 2.1), which is implemented as a *Python script*, is continuously fed by the

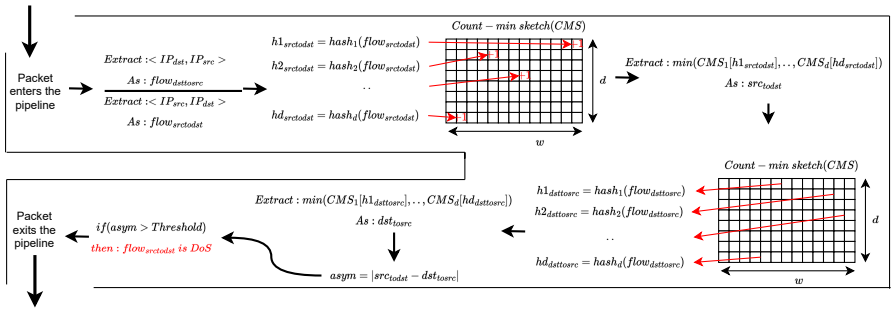


Figure 2.2: Asymmetric Count-min Sketch description. A detailed explanation can be found in Section 2.3.2.

JSON collection, relying on a *RESTful* communication.

2.3 P4RTHENON use case: DDoS detection

This section illustrates the use case we chose to validate our approach. The produced code has been open-sourced [Uni]. We considered volumetric DDoS detection as an example to showcase P4RTHENON peculiarities. We will refer to the specialized versions of CGM and FGM as CGM_DDoS and FGM_DDoS, respectively. A preliminary investigation on the considered use case can be found in [Sad+23].

2.3.1 Asymmetric Count-min Sketch (ACMS)

To detect suspect DDoS attacks in the data plane, we devised a simple sketch-based algorithm implemented in P4 called *Asymmetric Count-min Sketch* (*ACMS*, see Fig. 2.2). ACMS was designed by observing the behavior of volumetric DDoS attacks, which usually generate a large number of packets toward the victim by means of a large number of compromised clients belonging to a botnet. In particular, ACMS is designed to detect flows with an unexpected asymmetry rate. In this condition, the traffic volume between the compromised client and the victim is expected to be much larger than the traffic volume in the opposite direction. It should be noted however that P4RTHENON could be configured to support

different types of attacks and multiple flavors of DDoS attacks, e.g. DDoS attacks that target a specific destination (as analyzed in [Din+21]).

ACMS incorporates two algorithms, i.e., *Count-min Sketch* and *asymmetric flow detection*:

Count-min Sketch (CMS) [CM05]

It exploits a probabilistic, low-memory data structure (i.e., sketch) that can be used to estimate *flows' packet count*, i.e., the number of packets carried by any network flow in a time window. It relies on two operations carried out on the sketch: *(i) Update*, to keep the count of incoming packets updated in the sketch; *(ii) Query*, to estimate the number of counted packets for a given flow. CMS relies on d different pairwise-independent hash functions, each with an output size w. The data structure is composed of a matrix of $d \cdot w$ counters: the packet-count estimation accuracy increases as the two dimensions increase, and vice versa, with theoretical bounds that have been proven [CM05].

Asymmetric flow detection

It is a simple in-network algorithm (proposed in P-SCOR [Mel+20]) that calculates whether a flow is part of a potential DDoS attack. It uses a fixed $Threshold$, a data structure called R that includes w counters, and a hashing function h that returns a number between 0 and $w-1$. Every time a packet crosses the switch, k is calculated as the hash of the $s =< IP_{src}, IP_{dst} >$ string, i.e., $h(s) = k$. The counter of R in the k-th position, i.e., $R(k)$, is then incremented ($R(k) = R(k) + 1$). The algorithm then calculates $h(s') = j$, where $s' =< IP_{dst}, IP_{src} >$, and the asymmetry rate $asym = |R(k) - R(j)|$: if $asym > Threshold$, the flow is marked as a potential DDoS attack, as the difference of the traffic volume in the two directions is abnormal. The choice of identifying and tracking network flows considering the $< IP_{src}, IP_{dst} >$ couple rather than the more typical 5-tuple flow definition ($< IP_{src}, IP_{dst}, port_{src}, port_{dst}, protocol >$) has been made to ensure a slim approach in the data plane. Distinguishing 5-tuple malicious flows within $< IP_{src}, IP_{dst} >$ is a duty left to the control plane.

2.3.2 Strategies Description

CGM_DDoS

The strategy leverages ACMS as follows (Fig. 2.2). When a packet enters the data plane pipeline, the algorithm updates a CMS to increase the packets' counter for the considered flow. Then, the CMS is queried to retrieve the packet count estimation for the flow in the forward direction represented by the key $< IP_{src}, IP_{dst} >$, i.e., src_{todst}. The CMS is then queried again using the key $< IP_{dst}, IP_{src} >$ to retrieve the estimated packet count in the backward direction, i.e., dst_{tosrc}. The asymmetry rate is then computed as $asym = |src_{todst} - dst_{tosrc}|$: if $asym$ is higher than a value $Threshold$ the flow is labeled as *suspect*, and an alert is sent to the control plane in the form of a digest. This alert will be registered by the control plane, which will install a mirroring MAT during FGM_DDoS (we explain this thoroughly in Section 2.3.2). The CMS is reset by the control plane every time a fixed *time window* expires.

It must be noted that setting the most appropriate $Threshold$ is not trivial and could affect the detection performance in both CGM_DDoS and FGM_DDoS. In Section 2.4 we will report the results of a sensitivity analysis aimed at determining what $Threshold$ best suits our scenario.

FGM_DDoS

It comes into place, through a data plane pipeline reconfiguration, following an alert that is sent to the control plane during CGM_DDoS. It includes both a *data plane* and a *control plane* logic.

The *data plane* logic in the P4-programmable pipeline combines two sub-strategies, namely *(i) ACMS* and *(ii) optimized mirroring*. ACMS is the same as that deployed in CGM_DDoS, and it is needed to keep monitoring any new suspect flow once the data plane pipeline has been reconfigured. Optimized mirroring is instead deployed to extract relevant features from packets and forward them to the control plane through digests. We call it *optimized* mirroring because it is meant to minimize the amount of data flowing on the control channel. It only mirrors features from packets belonging to flows deemed suspect by ACMS, both the ones marked as such during CGM_DDoS and, if any, those detected during

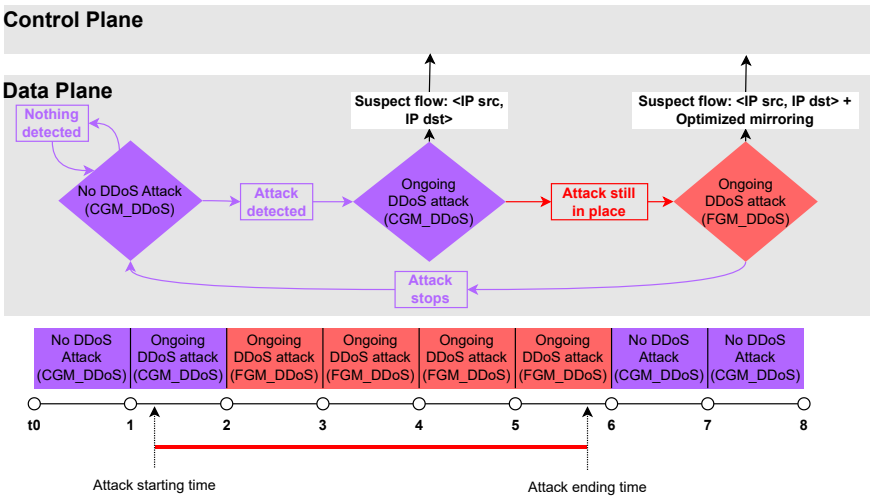


Figure 2.3: Flow diagram of transitions between CGM_DDoS and FGM_DDoS (top) and timeline (bottom).

FGM_DDoS. To further reduce the burden on the control channel, it also employs *packet sampling*, meaning that features from only 1 *out of* N (i.e., *sampling rate* of $1/N$) suspect packets, flowing through the pipeline, are forwarded. N is a parameter that needs to be carefully set to strike the best balance between detection performance and control channel utilization, as we will show in Section 2.4.

The *control plane* collects and stores the features extracted from the network traffic that are mirrored through the control channel. This data is then fed to a specialized online ML algorithm based on a pre-trained CNN model, i.e., LUCID [DC+20], which pre-process it and performs a classification task to determine what suspect flows truly belong to a DDoS attack and what are instead benign. LUCID enriches CGM_DDoS analysis on $< IP_{src}, IP_{dst} >$ to further discriminate the 5-tuple malicious flows between source and destination.

2.3.3 Transition between CGM_DDoS and FGM_DDoS

The time is slotted in *time windows*, which starts at integer time reference $t_s = \bar{t}$ and lasts until $t_e = \bar{t} + 1$. At the beginning of each time window, it is possible to switch from CGM_DDoS to FGM_DDoS or vice-versa. Figure 2.3 shows an example of how the transition between the two strategies occurs. The top part of

the figure reports a flow diagram showing state transitions in the face of a DDoS attack, while the bottom part focuses on a time perspective. Let's assume, as shown in the bottom part of the figure, that a DDoS attack starts during the second time window (in between $t_s = 1$ and $t_e = 2$) and expires in the sixth time window (in between $t_s = 5$ and $t_e = 6$). No other DDoS attack is in place in our time horizon, meaning that at the beginning of the first time window, CGM_DDoS is installed for coarse-grained traffic analysis.

During the first time window, nothing is detected by ACMS and no interaction between data and control plane occurs. During the second time window, as soon as the DDoS attack begins, CGM_DDoS starts sending alerts to the control plane every time a flow is considered to be suspect, as its asymmetry rate computed by ACMS overcomes the pre-defined threshold. After being notified of a possible attack, the control plane waits until the end of the current time window and then switches to FGM_DDoS, which requires a data plane pipeline reconfiguration: this happens at the beginning of the third time window, i.e., at $t_s = 2$. The reconfigured data plane starts extracting and mirroring features from (sampled) packets of the suspect flows identified by ACMS during CGM_DDoS, and at the same time monitors the rest of the traffic for potential new suspect flows. In the meantime, the control plane feeds the ML-based agent with packets' extracted features to identify malicious flows with high confidence. This condition holds until the DDoS attack ends, in this case during the sixth time window. As soon as this happens, the asymmetry rate of all flows falls behind the specified threshold, and at the beginning of the seventh time windows, CGM_DDoS can replace FGM_DDoS again.

2.3.4 Implementation

CGM_DDoS

To develop CGM_DDoS, we wrote ∼250 lines of P4 code. Our implementation of *Asymmetric Count-min Sketch* is summarized in Algorithm 1, including a description of the developed functions in P4.

The P4 program specifies a CMS data structure as an array of P4 registers, which is used to summarize the number of packets per flow (i.e., packet count)

in any direction. CMS is updated and queried leveraging a set of CRC32 hash functions (H), and the asymmetry $Threshold$ used to evaluate abnormal packet count differences in forward and backward flow directions is hard-coded in the program. Every time a *packet* enters the P4 pipeline, the following operations are sequentially performed:

- `updateCMS`: the CMS is updated. The packet count for the $< ip_{src}, ip_{dst} >$ flow is increased by one unit. This is done by accessing, for each row i of the data structure, the cell with index equivalent to the hash value of $< ip_{src}, ip_{dst} >$, obtained by considering the i-th hash function from the set H, and increasing its value accordingly (see [CM05]).

- `queryCMS`: the operation is similar to the one illustrated in `updateCMS` but, instead of updating the value from the cell in each row i, the minimum among the stored values in the cells are kept to estimate the packet count for the corresponding flow (see [CM05]). `queryCMS` is executed twice, first to estimate the packet count for the forward flow $< ip_{src}, ip_{dst} >$, and then for the backward flow $< ip_{dst}, ip_{src} >$. Those values are called min_{fwd} and min_{bwd} respectively.

- The asymmetry rate ($asym$) is finally computed as $asym = |min_{fwd} - min_{bwd}|$ and if it exceeds the value $Threshold$, the $< ip_{src}, ip_{dst} >$ flow is considered suspect of belonging to a DDoS attack. When this happens, an alert is sent to the control plane in the form of a digest, which wraps 64 bits containing ip_{src} and ip_{dst} of the flow. To reduce the burden on the control channel, such an alert is generated only once per time window, at the first time that $< ip_{src}, ip_{dst} >$ leads to an $asym$ value greater than the $Threshold$. Note that multiple alerts can be sent within the same time window, but the duplicates are ignored by the control plane.

Every Δt (time window size; in this chapter we consider a $\Delta t = 30$ s) the switch sends a digest notifying the expiration of the window, which can result in two different outcomes: *(i)* if no flow is deemed suspect during the time slot, no action is required apart from resetting the counters of CMS; *(ii)* if at least one

alert has been sent to the control plane during the window, the controller triggers FGM_DDoS.

Algorithm 1: Asymmetric Count-min Sketch

Input: $packet, H, CMS, Threshold$
Output: $sendAlert$

```
1  src ← packet.ip_src,   dst ← packet.ip_dst
2  CMS ← updateCMS(CMS, H, src, dst)
3  min_fwd ← queryCMS(CMS, H, src, dst),
     min_bwd ← queryCMS(CMS, H, dst, src)
4  asym ← |min_fwd − min_bwd|
5  if asym > Threshold and < src, dst > ∉ suspect_flows then
6      suspect_flows ←< src, dst >,   sendAlert(src, dst)
7  return
8  Function updateCMS(CMS, H, src, dst):
9      for i ← 0 to |H| − 1 do
10         h ← H_i(src, dst),   CMS_i[h] ← CMS_i[h] + 1
11     return CMS
12 Function queryCMS(CMS, H, src, dst):
13     min ← ∞
14     for i ← 0 to |H| − 1 do
15         h ← H_i(src, dst),   if CMS_i[h] < min then
16             min ← CMS_i[h]
17     return min
```

FGM_DDoS

The P4-based *data plane* pipeline logic of FGM_DDoS is a superset of the logic of CGM_DDoS.

In fact it includes ACMS (see Alg. 1) in its whole, with in addition:

1. A *feature extraction* logic to retrieve relevant features from packets flowing through the pipeline;

2. A *feature forwarding* logic to forward to the control plane only features *(i)* extracted from packets pertaining to suspect flows through ACMS and *(ii)* meeting the sampling requirements.

Algorithm 2: Optimized Mirroring: Feature Extraction

Input: *packet*
Output: *metadata*
`Parser()` **if** *packet* is IPv4 **then**
  $metadata.ip_{(*)} \leftarrow packet.ip_{(*)}$
if *packet* is ICMP **then**
  $metadata.icmp_{type} \leftarrow packet.icmp_{type}$
if *packet* is UDP **then**
  $metadata.udp_{len} \leftarrow packet.udp_{len}$
if *packet* is TCP **then**
  $metadata.tcp_{(*)} \leftarrow packet.tcp_{(*)}$
return *metadata*
`Ingress()` $metadata.timestamp \leftarrow ingress_{timestamp}$
return *metadata*

Together, 1) and 2) define the *optimized mirroring* strategy as described in Section 2.3.2. The feature extraction logic is detailed in Alg. 2 (it must be noted that we only consider IPv4 packets for compatibility with the control plane CNN), while the feature forwarding logic encapsulates the extracted metadata in a digest with a total size of 281 bits, which is sent to the control plane through the control channel using P4Runtime [Con20]. The procedure is shown in Alg. 3. The control plane then decodes the digest's data and saves the features in a JSON list. Simply put, Alg. 1 and Alg. 3 define the main logic of the ingress control block of the V1Model P4 pipeline, while Alg. 2 defines the parser logic.

Algorithm 3: Optimized Mirroring: Feature Forwarding

Input: $metadata, N_{sampling}, suspect_{flows}$
Output: *sendDigest*
$counter \leftarrow 0$
if $< metadata.ip_{src}, metadata.ip_{dst} > \in suspect_{flows}$ **then**
  $counter \leftarrow counter + 1$
  if $counter == N_{sampling}$ **then**
    $sendDigest(metadata), \quad counter \leftarrow 0$

The control plane exploits LUCID [DC+20] for a finer-grained detection of DDoS attacks. LUCID includes a trained ML model (i.e., CNN) and a prepro-

cessing algorithm, needed to reorganize retrieved features as required by the ML model (i.e., on a per-flow basis).

Feature	Description	Collected in	Protocol
ip_{src}	Source IP address of the packet	Parser	IPv4
ip_{dst}	Destination IP address of the packet	Parser	IPv4
ip_{flags}	IP flags used for fragmentation of the packet	Parser	IPv4
$ip_{protocol}$	Higher-layer protocol header encapsulated in the packet	Parser	IPv4
$ip_{totalLength}$	Size of the entire IP packet in bytes	Parser	IPv4
$icmp_{type}$	ICMP type of the ICMP packet	Parser	ICMP
udp_{len}	Length of the UDP segment in byte	Parser	UDP
tcp_{len}	Length of the TCP segment in byte	Parser	TCP
tcp_{ack}	Acknowledgement number of the TCP segment	Parser	TCP
tcp_{flags}	TCP flags of the segment (URG, ACK, PSH, RST, SYN, FIN)	Parser	TCP
$tcp_{srcPort}$	Source port number of the TCP connection	Parser	TCP
$tcp_{dstPort}$	Destination port number of the TCP connection	Parser	TCP
$tcp_{winSize}$	Window size of the TCP connection in bytes	Parser	TCP
$ingress_{timestamp}$	Timestamp of when packet is received in the ingress queue	Ingress Control Block	-

Table 2.4: Packet features encapsulated in the digest sent to the control plane by optimized mirroring. The features in red are used by LUCID [DC+20] in the preprocessing stage, but not for detection.

For an online detection (i.e., classification) of malicious flows, the JSON list including the features is continuously sent to LUCID via RESTful communication. LUCID then aggregates and splits the traffic into flows, marking them as malicious or as benign by means of ML inference (Table 2.4 lists the features used for detection). The JSON list is emptied every Δt seconds, i.e., every time window expiration. This is done to reduce the amount of data stored in the control plane and to keep it updated on the current shape of the underlying traffic. If LUCID is fed with the most recent traffic, it is possible to spot whether a flow previously deemed as malicious starts behaving legitimately. In this case, the flow can be ruled out from the list of malicious flows.

2.4 Performance Evaluation

This Section presents a performance evaluation of P4RTHENON, with respect to the considered use case of DDoS detection, both from a resource consumption and detection capability point of view.

2.4.1 Evaluation metrics and methodology

The tests here presented are based on a labeled PCAP dataset (details in Section 2.4.2), containing both true positives (TP, i.e., flows that belong to DDoS classes) and true negatives (TN, i.e., flows of benign traffic). The total number of flows is $T = TP + TN$. In each experiment, we obtain both false positives (FP, i.e., all the flows wrongly deemed belonging to a DDoS attack) and false negatives (FN, i.e., all those flows wrongly deemed benign). The *detection performance* is thus analysed by means of three metrics:

- $Precision = \frac{TP}{TP+FP}$. It measures how many of the positive predictions are correct. The higher the value, the lower the noise from false positives.
- $Recall = \frac{TP}{TP+FN}$. It measures how many positive cases are recognized. The higher the value, the lower the number of attacks escaping detection.
- $F1Score = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$. It is computed as the harmonic mean of Precision and Recall, indicating an overall quality of the detection.

We also measure the average *Control Channel Utilization (CCU)*, which is defined as the amount of data transmitted on the control channel, which we call $collectedData_{size}$, in an observation time window Δt, i.e., $CCU = \frac{collectedData_{size}}{\Delta t}$. The higher CCU, the less efficient the strategy in terms of data-control plane interaction.

We divided our evaluation into four parts:

- *CGM_DDoS evaluation*, which presents a performance evaluation of our solution if only in-network data plane detection is performed. We compare it to an effective state-of-the-art in-network solution.
- *FGM_DDoS evaluation*, which analyses and validates our solution when ML-based control plane logic is installed. We evaluate the effectiveness and efficiency of the strategy over multiple combinations of ACMS thresholds and sampling rates.
- *Overall evaluation*, which summarises the results of CGM_DDoS and FGM_DDoS when combined, clearly pointing out the benefits of P4RTHENON with respect to other approaches.

- *Data plane pipeline reconfiguration evaluation*, which reports a brief discussion on the time needed by P4RTHENON to reconfigure the data plane pipeline while swapping between CGM_DDoS and FGM_DDoS.

Before delving into the obtained results, in the following, we give a concise description of the testbed and its settings.

2.4.2 Description of the testbed environment and parameters

Our experiments were carried out on in a virtual environment that consists of:

- An emulated single-switch network running on Mininet [KSG14] with bmv2 [Con+19] as P4 virtual switch target;
- A controller, developed in Go [Goo23], responsible of *(i)* the information exchange with the data plane and *(ii)* reconfiguring the pipeline using the P4Runtime APIs.
- A process running LUCID, interacting with the controller via RESTful communication. LUCID was pre-trained using a dataset provided in its official repository [dor], and model hyperparameters were set as the default ones specified in the paper [DC+20]. For further details on LUCID's configuration the reader should refer to [DC+20].
- A process simulating the DDoS attack by means of tcpreplay [Lin23], which replays network traffic at 50 Mbps speed for a 6-minute long attack. We used a pre-generated PCAP sample dataset containing roughly 2 GB of traffic. It is composed of 10% of benign traffic (taken from the CIC-IDS2017 dataset [Cic]) and 90% of DDoS traffic (generated with the hping3 [Lin22] Linux utility). The attack speed is designed to saturate the switch, while the 6-minute duration allows replaying the dataset ~ 2 times. We generated traffic datasets with different numbers of malicious $< IP_{src}, IP_{dst} >$ flows (from 30 to 120) to stress the CMS with various traffic volumes: this aspect will be analyzed later in this section.

All the components were executed on an Ubuntu 20.04 LTS Server with 14GB of RAM and 3 CPU cores KVM machine.

The in-network P4-based ACMS strategy uses a $(d = 2) \times (w = 1024)$ CMS, where every 48 bits are allocated to each cell, resulting in $2 \cdot 1024 \cdot 48 \sim 9.8Kb$ memory occupation. The two adopted hash functions are available by default in the bmv2's v1model.p4 [Conb], i.e., $crc16$ and $crc32$.

2.4.3 CGM_DDoS evaluation

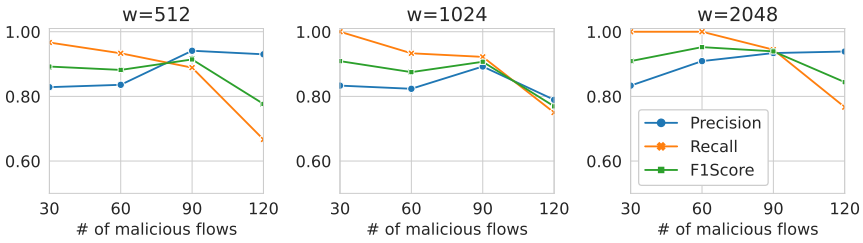
Sensitivity analysis of ACMS

To choose the right values d and w for CMS we conducted a detailed sensitivity analysis. Table 2.5 shows a comparison between different values of d for $w = 1024$. It reports how the F1Score improves for $d = 2$ with respect to $d = 1$, and does not significantly improve further for $d = 3$ or more, meaning that $d = 2$ is a valid compromise between good detection performance and acceptable memory consumption.

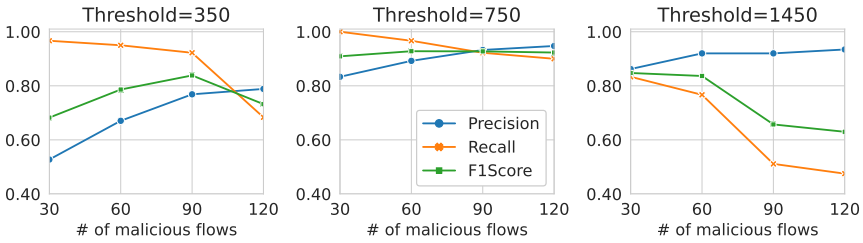
Depth	Precision	Recall	F1Score	Memory in switch (Kb)
$d = 1$	0.54	0.95	0.69	4.9
$d = 2$	0.69	0.97	0.81	9.8
$d = 3$	0.71	0.97	0.82	14.7

Table 2.5: Detection comparison of ACMS while varying d (with $w = 1024$ fixed).

Figure 2.4 shows the orthogonal analysis, for fixed $d = 2$. Here, the Precision, Recall, and F1Score are collected over a variable *number of malicious flows*, with fixed w (Figure 2.4a) and fixed ACMS *thresholds* (Figure 2.4b). This analysis suggests that for $w = 1024$, the F1Score is significantly higher for any number of malicious flows compared with $w = 512$, and almost matches $w = 2048$. On the other hand, for $Threshold = 750$, the F1Score outperforms every other configuration. The threshold analysis anticipates the result we will further discuss in Section 2.4.4. This investigation suggests that the best parameters for CGM_DDoS, given our settings, should be $d = 2$ and $w = 1024$. Moreover, in all the following experiments we will focus on a number of malicious flows equal to 60.



(a) ACMS variable width



(b) ACMS variable threshold

Figure 2.4: CGM_DDoS: Detection performance for different ACMS widths (with $d = 2$ and $Threshold = 750$) and thresholds (with $d = 2$ and $w = 1024$) while varying the number of malicious flows.

Comparison with the state of the art

Strategy	Precision	Recall	F1Score	CCU (Kbps)	Memory(Kb)
[Din+21]	0.86	1	0.93	0.001	3145.7
CGM_DDoS	0.80	0.97	0.88	0.03	9.8

Table 2.6: Detection comparison between an in-network state-of-the-art strategy [Din+21] and CGM_DDoS.

We compare CGM_DDoS with an open-source [Din], state-of-the-art solution called INDDoS [Din+21]. As CGM_DDoS, INDDoS is an in-network P4-based solution that detects hosts targeted by volumetric DDoS attacks. It is threshold-based, like ACMS, i.e., the core strategy of CGM_DDoS: it estimates the *per-destination flow cardinality* (number of sources contacting a specific destination) and, if it is above a threshold value, the destination is considered under attack. To estimate it, *BACON sketch* is used: a data structure that combines a CMS and a Bitmap to *Update* and *Query* the per-destination flow cardinality once a packet

enters the P4 pipeline. When the queried value crosses the specified threshold, a digest wrapping the *IP destination* of the victim is sent to the control plane.

The main difference between INDDoS and ACMS is that they focus on two different properties of volumetric DDoS attacks to detect them: the former on per-destination flow cardinality (which is expected to be high for destinations under attack), the latter on flows' asymmetry rate (which is expected to be high for malicious flows). We want to stress that INDDoS could replace ACMS as the core in-network algorithm of CGM_DDoS. However, if we look at Table 2.6, some aspects can be highlighted.

We decided to test CGM_DDoS and INDDoS considering their best configuration in terms of detection performance (F1Score) which are:

- INDDoS: $Threshold = 60$, BACON sketch of size $(d = 3) \times (w = 1024) \times (m = 1024)$ [Din+21].
- CGM_DDoS: $Threshold = 750$, CMS of size $(d = 2) \times (w = 1024)$ (as shown in the previous subsection);

From Table 2.6 it can be seen that much more memory is used by INDDoS with respect to CGM_DDoS. In fact, the memory occupied by BACON sketch, considering that 1 bit is allocated to each cell [Din+21], is $3 \cdot 1024 \cdot 1024 = 3145.7Kb$, which is more than 300 times higher than the memory occupied by the CMS adopted by CGM_DDoS (i.e., $9.8Kb$). However, INDDoS outperforms CGM_DDoS in terms of Precision, Recall, and F1Score. This is explained by the higher complexity (and required memory) of INDDoS compared with CGM_DDoS, which makes it a more performing stand-alone solution. However, Recall of both solutions is high (1 or close to 1), while Precision of both is just decent, with slightly worse performance for CGM_DDoS.

In addition, CCU of CGM_DDoS, although only in the order of tens of bps, is higher than CCU of INDDoS. This happens for two reasons: *(i)* the higher frequency of sent alerts, as CGM_DDoS sends an alert every time it spots a suspect flow, while INDDoS groups alerts by destination; *(ii)* the higher size of the digests payload, as CGM_DDoS sends 8-bytes alerts (IP source and IP destination of the flow), while INDDoS sends only 4-bytes alerts (IP address of the victim). In our

testbed, the size of a CGM_DDoS alert is around 100 *bits* after being encapsulated in a JSON structure, while the size of an INDDoS alert is around 50 *bits*.

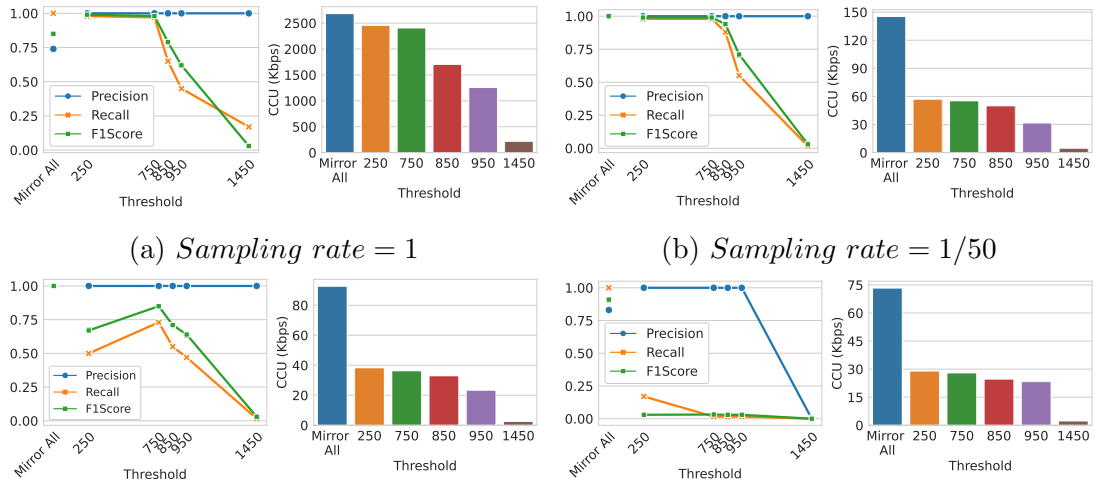
To summarize, INDDoS provides a superior detection performance compared to CGM_DDoS. However, as specified in Section 2.2.2, P4RTHENON requires a low number of false negatives for CGM, which is guaranteed by both strategies (high Recall), while it is tolerant to false positives, which can be filtered out by FGM. So, although CGM_DDoS Precision is slightly lower and CGM_DDoS CCU higher (but still low in absolute terms), its adoption in the place of INDDoS is fully justified by its much lower memory usage.

2.4.4 FGM_DDoS evaluation

In this Section, we analyze the benefits of FGM_DDoS. We provide an overview of the configurations we tested in the environment described in Section 2.4.2, setting different ACMS *thresholds* and different *sampling rates*. The goal is to explore the existing trade-offs between detection performance (in terms of Recall, Precision, F1Score) and Control Channel Utilization, as these two configuration parameters are the most impactful on the above-mentioned metrics. Furthermore, we compare these results with a naïve strategy that we call *Mirror All* and is inspired by [DC+24]: it does not provide ACMS-aided optimized mirroring, but it simply performs features extraction and forwarding from *any* packet, regardless of that it belongs to a suspect flow or not. In other words, it does not embed any ACMS logic and discriminating between benign and malicious flows is fully enforced by the control plane. As for FGM_DDoS, it is possible to reduce the burden on the control channel through sampling, i.e., by only forwarding features extracted from one packet out of N.

We tested different combinations of ACMS thresholds and sampling rates: in the following we report only the most significant combinations for the sake of conciseness. Figures 2.5 and 2.7 report the results for our tests, where for each configuration Precision, Recall, F1Score and CCU are reported.

With respect to CCU reported values, we want to stress that in FGM_DDoS the size of a digest, including the packet's features, is around $2Kb$, i.e., 20 times the size of the CGM_DDoS one. Moreover, in CGM_DDoS a much lower number



(a) *Sampling rate* = 1

(b) *Sampling rate* = 1/50

(c) *Sampling rate* = 1/75

(d) *Sampling rate* = 1/100

Figure 2.5: FGM_DDoS vs. Mirror All: Detection performance and Control Channel Utilization for different thresholds (sampling rate fixed).

of digests is sent to the control plane, as only one digest per suspect flow, in any time window, is forwarded to the control plane. This is the reason why CCU for FGM_DDoS is several orders of magnitude higher than for CGM_DDoS (as reported in Table 2.6).

Figure 2.5 reports the detection performance and CCU under four chosen sampling rates, namely, 1, 1/50, 1/75, 1/100, and varying the ACMS threshold. Note that Mirror All is insensitive to the threshold as ACMS is not adopted, and thus in the left-hand-side subfigures its Precision, Recall and F1Score values are reported as single points. We can see that by increasing the threshold a negative impact on Recall is experienced, as the number of false negatives significantly increases. In fact, only flows with very high asymmetry rates are deemed suspect by ACMS and thus some malicious flows, with lower asymmetry rate, are neglected by ACMS. On the other hand, choosing a higher threshold has a very good impact on CCU, as features extracted by packets belonging to fewer flows (i.e., only those suspect) need to be forwarded to the control plane. Instead, Precision is not strongly affected and is always high, meaning that LUCID has a very good ability to filter out false positives.

Figure 2.5 also shows that lowering the sampling rate from 1 to 1/50 is beneficial

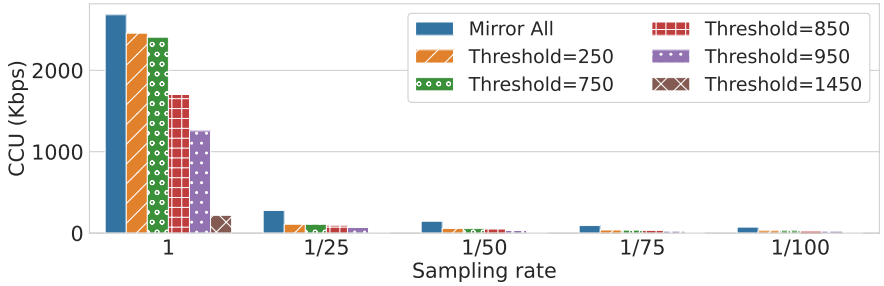
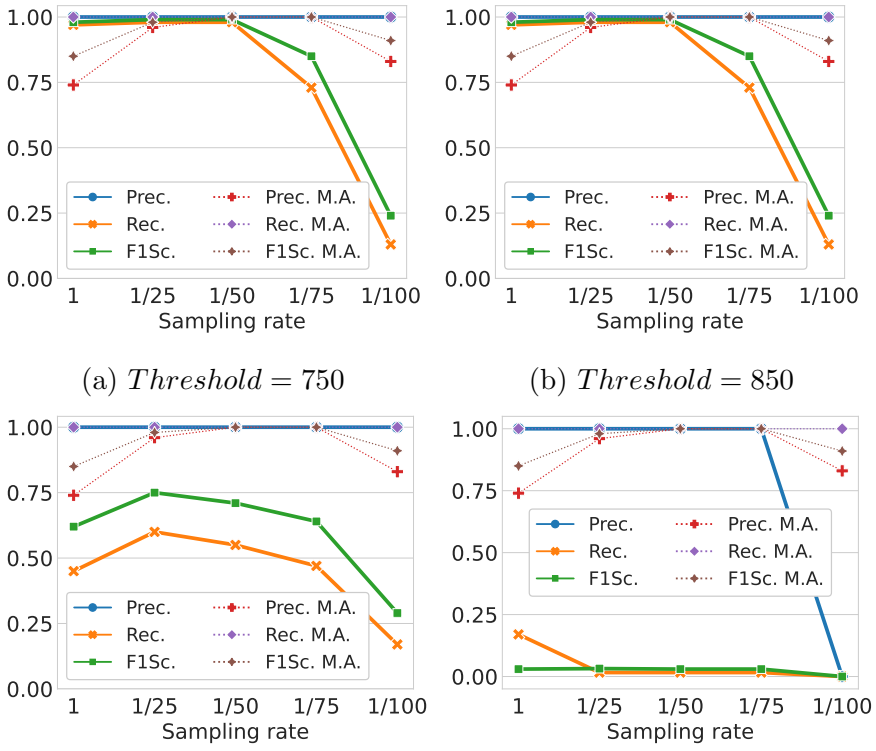


Figure 2.6: FGM_DDoS vs. Mirror All: Control Channel Utilization for different sampling rates (threshold fixed).



(a) $Threshold = 750$ (b) $Threshold = 850$

(c) $Threshold = 950$ (d) $Threshold = 1450$

Figure 2.7: FGM_DDoS vs. Mirror All: Detection performance for different sampling rates (threshold fixed). M.A. = Mirror All, Prec. = Precision, Rec. = Recall, F1Sc. = F1Score.

for both detection performance and Control Channel Utilization for high thresholds. CCU is lowered by one order of magnitude, while the detection performance (in terms of F1Score) increases. This phenomenon may seem counter-intuitive, however, by lowering the amount of data sent to the control plane, congestion on the control channel is reduced with consequent benefits on detection performance. In fact, congestion causes uncontrolled digests' discard, meaning that lower congestion reduces the amount of noise (in terms of flows' patterns alteration) given as input to LUCID. By further decreasing the sampling rate, e.g. 1/75 and 1/100, the number of packets' features sent to the control plane decreases up to a point that LUCID has not enough data to perform a proper classification. CCU is low but Precision, Recall, and F1Score are also low regardless of the threshold value.

The same trend is confirmed by looking at Figs. 2.7 and 2.6, which report the detection performance (Fig. 2.7) and CCU (Fig. 2.6) under four chosen value of the ACMS threshold, namely 750, 850, 950, 1450, and while varying the sampling rate.

Fig. 2.7 shows that the detection performance peeks for sampling rates higher than 1/50. However, the most important trend is clearly highlighted in Fig. 2.6: whenever sampling is performed, CCU for both Mirror All and FGM_DDoS drops significantly. For very low sampling rates ($< 1/75$) the same considerations as those done for Fig. 2.5, with respect to high thresholds, apply: in these cases, the amount of informative data sent to the control plane is too limited to ensure robust detection performance. Also the case for threshold values of 750 and 850 is interesting. With respect to detection performance (Fig. 2.7) they behave the same for any sampling rate, but CCU (Fig. 2.6) is reduced by 30% in the case of a threshold of 850.

By comparing FGM_DDoS with Mirror All, we can see that Mirror All performs best for sampling rates of 1/50 and 1/75. Its counter-intuitive worse detection performance with a sampling rate of 1 is due to the high congestion on the control channel. However, in all the cases, Mirror All leads to a much higher CCU than FGM_DDoS. Specifically, the same detection performance of Mirror All can be obtained by FGM_DDoS with a sampling rate 1/50 and threshold of 750, while reducing CCU by around three times.

In summary, our results show that by choosing the most appropriate sampling

rate and ACMS threshold our strategy makes it possible to find a good balance between detection performance and amount of traffic on the control channel.

Strategy	Precision	Recall	F1Score	CCU (Kbps)	Memory(Kb)
CGM_DDoS	0.80	0.97	0.88	0.03	9.8
[Din+21]	0.86	1	0.93	0.001	3145.7
FGM_DDoS	1	0.98	0.99	55.3	9.8
Mirror All	1	1	1	92.7	0
P4RTHENON	1	0.98	0.99	22.7	9.8

Table 2.7: Overall comparison between the different strategies.

2.4.5 Overall evaluation

Table 2.7 summarizes the results obtained in Sections 2.4.3 and 2.4.4, with respect to the following strategies and related configuration parameters:

- CGM_DDoS: $Threshold = 750$, CMS of size $(d = 2) \times (w = 1024)$;
- INDDoS: $Threshold = 60$, BACON Sketch of size $(d = 3) \times (w = 1024) \times (m = 1024)$ [Din+21].
- FGM_DDoS: $Threshold = 750$, $Sampling\ rate = 1/50$, CMS of size $(d = 2) \times (w = 1024)$.
- Mirror All: $Sampling\ rate = 1/75$.
- P4RTHENON: $Threshold = 750$, $Sampling\ rate = 1/50$, CMS of size $(d = 2) \times (w = 1024)$.

P4RTHENON combines CGM_DDoS and FGM_DDoS via data plane pipeline reconfiguration, as specified in Sections 2.2 and 2.3. The parameters of each strategy has been chosen to maximize the detection performance (in terms of F1Score) as first objective and, in the case of multiple settings with the same detection performance, the one that minimizes CCU.

Table 2.7 shows how P4RTHENON, FGM_DDoS and Mirror All outperform in terms of F1Score the in-network strategies that are fully executed in the data plane (i.e., CGM_DDoS and INDDoS). However, Recall is always high, meaning

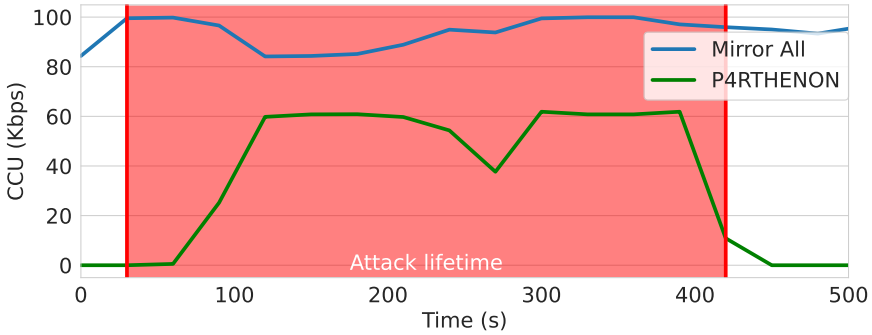


Figure 2.8: Control Channel Utilization over time between P4RTHENON and Mirror All in their best configurations.

that all strategies, also those fully executed in the data plane, are good at effectively identifying true positives (i.e., the malicious traffic). It follows that the strategies relying on LUCID as an ML engine in the control plane (i.e., FGM_DDoS, Mirror All and P4RTHENON) have a much higher Precision, meaning that by deeply analyzing in the control plane the traffic features extracted from packets is very effective it to keep the number of false positives low. In the case of FGM_DDoS and P4RTHENON this property can be effectively exploited to filter out in the control plane the flows that are identified as suspect by ACMS in the data plane, which instead are benign. Aside from detection quality considerations, resource allocation and utilization are the aspects that make our proposed solution stands out. If we analyze CCU we can see that the in-network strategies lead to minimal usage of the control channel, while the others, for which feature extraction and forwarding to the control plane is needed, pay the price of a much higher average channel occupation. However, FGM_DDoS and especially P4RTHENON have a reduced CCU with respect to Mirror All, of around 40% and 75% respectively, as they benefit from the presence of ACMS to only forward features from suspect flows. P4RTHENON reduces CCU even further by having almost no interaction between control and data plane when CGM_DDoS is installed and attacks are not under stricter scrutiny.

Moreover, by looking at the occupied memory in the switch, we can stress again how INDDoS allocates a much higher amount of memory ($3145.7Kb$) than ACMS ($9.8Kb$), which is used in CGM_DDoS, FGM_DDoS and P4RTHENON. Instead,

Mirror All does not require any data structure in the data plane, so it does not consume memory. This, however, comes at the expense of a significantly higher CCU.

Finally, a comprehensive look at Table 2.7 shows how P4RTHENON, thanks to its peculiarities, strikes the best balance between detection performance, CCU, and memory occupation with respect to the other strategies. Figure 2.8 confirms the speculation we drew from Table 2.7 and reports a comparison between the two strategies with the best detection performance, i.e., Mirror All and P4RTHENON, in terms of CCU overtime during an attack, which is marked by a red area. The attack starts at $\bar{t} = 30s$: for P4RTHENON, CGM_DDoS is in place before this time instant, and a negligible amount of data is sent on the control channel. After $\bar{t}$, CGM_DDoS starts identifying suspect flows and after another Δt, at $t' = 60s$, FGM_DDoS is installed and optimized mirroring starts (correspondingly, CCU increases). Then, the attack ends at $t'' = 420s$ and, in the next time window, CGM_DDoS is restored and CCU drops to almost zero. By looking instead at Mirror All, we can see an almost constant CCU of around 90 Kbps as the features are extracted and forwarded from *any* packet, also when no attack is happening. Moreover, when the attack is in place, the data plane logic adopted by P4RTHENON (i.e., ACMS) makes it possible to save much control channel bandwidth by only forwarding features from suspect flows.

2.4.6 Data plane pipeline reconfiguration evaluation

We performed some experiments to evaluate the *system downtime* when a real-time P4 pipeline reconfiguration is performed to swap between CGM_DDoS and FGM_DDoS. It is important to stress that such an evaluation is strongly dependent on the adopted emulated environment and software switch target, and further tests will be performed as future work on hardware testbeds to confirm our findings. In our experiment we swapped between CGM_DDoS and FGM_DDoS 100 times and we measured the downtime during each transition, then calculating mean and variance. The computed mean is 263.4 ms, with a very low variance (0.6). Reconfiguring the pipeline, at least on Mininet with bmv2, is quick and stable.

2.5 Discussion

Minimizing data exchanged on the control channel for data-driven monitoring tasks is pivotal in complex networks. In fact, introducing a new feature or service should not be detrimental to the system. P4RTHENON is a scheme that supports the employment of lightweight and precise monitoring tasks to meet these requirements. It leverages P4-assisted real-time reconfiguration of programmable network devices, with minimal overhead and traffic loss.

We demonstrate the validity of our scheme by formulating a P4RTHENON-assisted solution to detect volumetric DDoS attacks. This strategy leverages two phases: *(i)* a pre-filtering stage to select the important portions of suspect traffic to analyze, and *(ii)* a fine-grained strategy that leverages optimized packet features' mirroring from the data plane towards the control plane, where a ML-based specialized agent attests what portion of suspect traffic is indeed malicious. This use case shows how P4RTHENON can reduce the cross-plane communication overhead by almost 80% while keeping high DDoS detection rates.

Being the use case is of practical significance, we foresee to proceed in its improvement by investigating its performance on a hardware testbed and by automating parameters' optimization (i.e., ACMS threshold and sampling rate) according to the traffic shape. In addition, we believe that the presented approach could be applied with profit to other, more complex network monitoring scenarios; our line of research will be correspondingly widened to encompass other use cases for a broader validation of P4RTHENON.

Chapter 3

Continual detection upgrade*

We previously analyzed how we can empower network monitoring with data plane programmability to detect attacks in-line with the computation capabilities of an ML-powered control plane. In this chapter, we are going to analyze how to continually upgrade the anomaly detector by selecting relevant features on the data plane. Machine Learning (ML) has lately become a prominent research area for the networking community, with applications in a broad range of topics such as traffic classification [Wic+22], routing [Cha+23], congestion control [AYC20], and traffic forecasting [Qia+22]. In particular, in-network ML has become very attractive, as it allows to leverage the expressiveness of ML models at data plane speed [Swa+22; XZ19]. The common denominator between many in-network ML use cases is to train a model in the control plane using annotated historical data, and then deploy the model in the data plane for near real-time inference [SSB18; SB18; XZ19]. Unfortunately, the training data will eventually become outdated (a phenomenon formally known as *"distribution shift"* or *"concept drift"*), causing the deployed ML model to suffer from performance degradation [MC18; Mal+22]. While the necessity for frequent model updates has already been raised [ALS23], three fundamental questions remain: (1) **When should we update our model?** The answer is (in theory) fairly simple: *continuously*. We should assume that the input patterns observed by a deployed ML model may change at any point in time;

*Part of the material presented in this chapter is based on the following publications: Nicola Di Cicco et al. "Poster: Continual Network Learning". In: *Proceedings of the ACM SIGCOMM 2023 Conference*. 2023, pp. 1096–1098

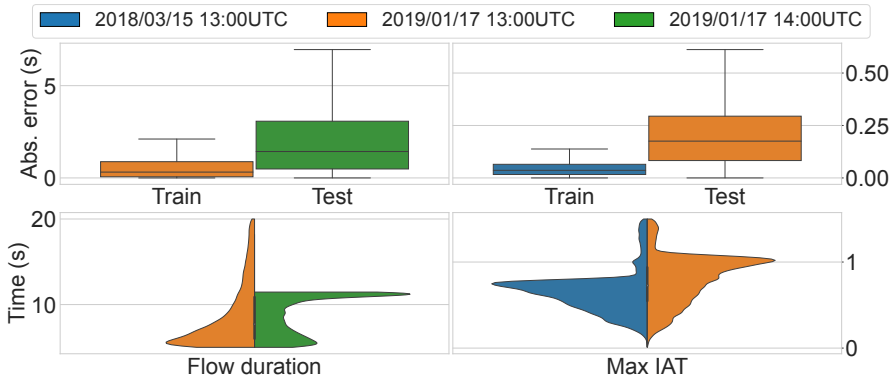


Figure 3.1: Distribution shift of TCP flow features from a real-world commercial backbone link [CAI19]. The distribution shift of flow duration and inter-arrival time (IAT) causes performance degradation of a ML model trying to predict them.

(2) **Which data should we select for model updates?** Again, the answer is (in theory) simple: *only the data that is useful for learning new things.* (3) **What should our model learn?** In principle, *everything.* We want a model that dynamically expands its predictive power without forgetting past experiences.

In this chapter, we aim to take a step towards designing a solution that answers those questions. We propose combining Active Learning (AL) [Set12], which enables filtering relevant information from a vast pool of unannotated data, and Continual Learning (CL) [DL+22], which allows us to learn from streaming data *without forgetting past concepts.* The former, implemented in the switch ASIC, allows us to choose *the right amount of information* that shall be mirrored to the control plane, where the model is updated continually. Finally, the new model can be installed back in the data plane.

Implementing this solution is nontrivial and needs answering the following research questions: (1) how to implement AL-based filtering in the data plane?; (2) how selective should AL be for network learning?; (3) which ML models are most suitable for continual learning of network traffic?; and (4) how to dynamically reconfigure the data plane?

3.1 The case for continual learning

We run some tests on real-world traces to characterize the amount of distribution shift in TCP flow features. We extracted commonly used features from real-world traces [CAI19] (e.g., flow duration, inter-arrival time (IAT), and packet size statistics). We observed a shift in the flow duration and in the maximum IAT for small time scales (∼one hour) and for large time scales (∼a year), respectively. To quantify the impact of these shifts, we consider ad-hoc regression tasks (because CAIDA traces do not have task-specific class labels) where the targets are either the flow duration or the maximum IAT. We observe (for visualization purposes, we focus on the ranges [5, 20]s for flow duration and [0, 1.5]s for inter-arrival time) that the test error is significantly larger than training, a phenomenon that is imputable to the observed feature drifts (Fig. 3.1). Indeed, classical ML models will work properly only if the train and test data are approximately i.i.d. [Bis06]. As such, practical in-network ML calls for smart, adaptive approaches.

Why can't we run existing proposals in a loop? Literature has been active in proposing efficient means for offloading trained ML models to the data plane [Zhe+22; Swa+23; CSF22; Swa+22]. We here consider an orthogonal problem: how to train a ML model continually from packet streams with the optimal amount of annotated training data. Though Online Learning approaches have been explored [MC18; Swa+22], they 1) assume that every streamed data point is labeled, and 2) do not pay attention about forgetting the past as long as the model is fit to the current experience. In our proposal, we want not only to learn adaptively, but also to remember (and therefore exploit) everything that was observed in the past. In this way, our model will not need additional data for re-learning already-observed concepts.

3.2 Our approach

Fig. 3.2 illustrates our proposal: to incorporate in a single closed-loop framework the following building blocks:

1. Model training: update the ML model over the time with CL.

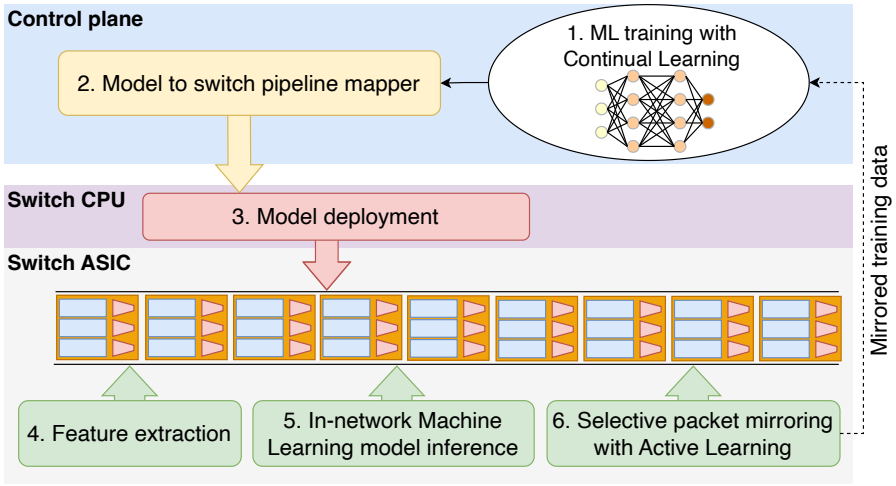


Figure 3.2: Our approach. The ML model is created (1) and then deployed in the switch ASIC (2), to perform inference at data plane speed (3). Selective mirroring (4) with Active Learning is deployed to keep the ML updated with Continual Learning.

2. Model deployment: deploy the new ML model in the data plane.

3. In-network inference: enable inference at data plane speed.

4. Selective mirroring: mirror to the control plane only the data useful for expanding the knowledge of the model with AL.

As a proof-of-concept experiment, we consider a subset of the CIC2019 dataset for DDoS classification [Sha+19]. We consider DDoS classes to represent disjoint learning tasks, which are presented to the model in sequence. For each task, the model must not only discriminate between benign and malicious flows but also place the malicious flows in the right class.

We implement a baseline Continual Random Forest (CRF), consisting of a RF augmented with a replay buffer storing the most informative past exemplars. We use the vote count as AL query strategy, selecting only data points whose predictions had less than 90% majority. We retrain after each query, which is computationally efficient for RFs. We consider an Adaptive Random Forest (ARF) as a purely online (but not continual) state-of-the-art baseline [Gom+17; MC18]. In contrast to CRF, ARF assumes that every data point is labeled. We also

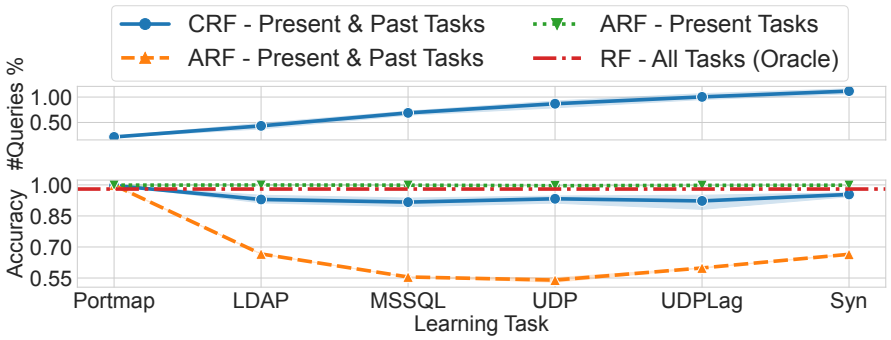


Figure 3.3: Adaptive vs. Continual Random Forests for class-incremental DDoS classification on CIC2019. At the end of the stream, CRF achieves performance close to an "oracle" while requiring only ∼1% of the data.

consider an "oracle" RF trained on the full dataset as an upper-bound on the average performance over all tasks.

Fig. 3.3 shows the performance of CRF and ARF over the sequential tasks, and the percentage of queried labels by CRF relative to the full stream size. A purely adaptive learner such as ARF, though able to master individual tasks, quickly forgets past concepts. Instead, our baseline CRF achieves a performance close to the oracle upper-bound, while requiring labeling only ∼1% of the observed samples.

3.3 Challenges

Challenge #1: implementing AL-based filtering in the data plane. Vote count in our baseline CRF is a decent query strategy, but information-theoretic quantities [Hou+11; BS+23] are among the state-of-the-art. Their data plane implementation is not trivial, as it would require floating-point arithmetics. Even if not standard, authors in [Pat+22] propose a way to implement floating-point arithmetics in P4.

Challenge #2: how selective should AL be. A small selectivity implies a large mirroring overhead, whereas a large selectivity implies a potential information loss. Applying AL to streaming data is, as of today, a novel twist on classical techniques [Sar+23]: investigating these trade-offs opens up interesting research directions.

Challenge #3: choosing the right CL strategy. Our baseline leverages a slowly-growing experience buffer, which may not be desirable. Strategies for maintaining the buffer of fixed size can be investigated [Reb+17]. Other solutions, e.g., regularized neural networks, do not require any storage overhead other than the model [Kir+17], but are ill-advised for tabular data [SZA21]. Ultimately, the choice depends on the available storage/computational resources and the goodness-of-fit to the characteristics of task-specific data.

Challenge #4: runtime dataplane reconfiguration. Currently, if we want to add a new functionality to a switch, we need first to reroute the traffic of that switch, flush a new image in its ASIC and then restore the original traffic policy configuration. This process can lead to dramatic consequences if performed carelessly [Jan21]. Programming the switch at run-time is possible [Xin+22], but not for RMT [Bos+13], the common commercial devices architecture [Int20; AMD19]. Researchers have also explored means to enable isolation between offloaded programs [SZ20; ZBH18; Wan+22], which we will investigate to isolate the Active Learning processing and the rest of the pipeline.

Part III

Safe distributed alerting with DLTs

Chapter 4

IOTA to safely propagate data plane alerts*

In this Part, and particularly in this chapter, we will investigate how to combine the in-line attack detection with DTLs to distribute alerts in decentralized systems. Modern systems heavily adopt decentralized systems that rely on a network of nodes for computation and data storage. These systems facilitate the collaborative and distributed use of computational resources, instead of relying on a central authority, leading to more efficient resource utilization and greater resilience. Attackers, however, find in the very architectural features of distributed technologies exploitable vulnerabilities. One of the most famous attacks that leverage these architectures is the Distributed Denial of Service (DDoS) [OCD16], which seeks to disrupt network services and host connectivity in a distributed environment by overloading the network with unnecessary requests. Avoiding and mitigating DDoS attacks is a primary concern for many organizations.

Software-defined Networking (SDN) is a cutting-edge networking approach that divides management over control and data plane layers. In SDN, the physical network layer is seen as fully programmable, resulting in increased customization of data packet processing. Such a feature has greatly contributed to its widespread across different cloud infrastructures. In this direction, the Programming Protocol-

*Part of the material presented in this chapter is based on the following publications: Amir Al Sadi et al. "P-IOTA: A cloud-based geographically distributed threat alert system that leverages P4 and IOTA". in: *Sensors* 23.6 (2023), p. 2955

independent Packet Processors (P4) has emerged as an innovative programming language, operating at the data plane level, to configure network devices in a highly customizable manner. P4 allows full programming of networking devices while being target-independent. Furthermore, the capability of programming the data plane boosts the ability to detect network attacks.

How to effectively and securely share information to detect attacks is a challenging task. Distributed ledger technologies (DLTs), such as blockchain, are digital systems spread across multiple locations that securely store information (transactions) without a central entity. Blockchain represents the most famous type of DLTs: its data structure foresees a chain of blocks connected through hashes and validated by 3rd-party entities (i.e., miners or validators) following a consensus protocol. However, the validation process consumes significant amounts of time and energy, which can hinder blockchain's efficiency and adoption for information sharing. To address these concerns, alternative DLTs such as IOTA are recently emerging as promising solutions. IOTA offers the same security features of the blockchain (i.e., immutability, traceability, and transparency) while addressing efficiency concerns.

The differences between IOTA and blockchains have been widely investigated in the literature [Als+22; Auh+22; Reb+21], with numerous studies showing that IOTA is a superior solution in terms of scalability, transaction rate, efficiency, and reducing energy consumption. IOTA outperforms traditional blockchains due to its low latency and the ability to send transactions without any fees [Als+22]. [Reb+21] presents an in-depth comparison of the performance of multiple consensus protocols where IOTA achieves the best performance with a transaction rate that is several orders of magnitude higher than the other protocols. Therefore, its lightweight consensus protocol makes it one of the few truly suitable technologies for Internet of Things (IoT) devices [Auh+22]. These properties are particularly relevant in SDN-based environments due to the strict latency requirements of the data plane. At the state-of-the-art, there are very few examples of research proposals that integrate P4 and DLTs. We argue that one of the main reasons is that traditional blockchain-like technology, such as Ethereum, introduces considerable overhead to compute blocks, clashing with the real-time attack detection capabilities of P4.

This chapter presents P-IOTA, a system that leverages P4 and IOTA to detect ongoing network attacks in real-time. Our solution offers high performance and enables real-time communication of events in a distributed manner. P-IOTA can be employed to address a wide range of attack scenarios. We developed a proof-of-concept using a well-known P4 implementation for DDoS attack detection and we simulated a SYN flooding attack. The experimental results demonstrate that P-IOTA outperforms blockchain-based proposals in detecting and communicating attacks in real-time while correcting false attack notifications.

The chapter is organized as follows. In Section 4.1, we introduce the relevant background information regarding DTLs. The relevant literature is reviewed in Section 4.2 to present the background concepts used in our work and the limitations of existing solutions. In Section 4.3 we describe P-IOTA, including its main components and features. Then, Section 4.4 documents the experimental results of a simulated SYN flooding attack. Finally, we discuss our results in Section 4.5.

4.1 Background

In this section, we provide essential background information about DLTs with a specific emphasis on IOTA.

4.1.1 Distributed Ledger Technology

Distributed Ledger Technologies (DLTs) are types of distributed database that avoids data centralization and do not require administration functionality. The stored information is replicated on multiple nodes that maintain a copy of the entire database. Since DLTs do not use centralization or third-party entities, the data source is built collaboratively, allowing multiple entities to contribute data. Unlike traditional databases, data memorized on a DLT can neither be modified nor deleted, as they are usually implemented as append-only data structures. They rely on Peer-to-Peer (P2P) networks as they are decentralized systems. The lack of a centralized control entity avoids the single point of failure issue. For this reason, DLTs adopt consensus protocols to keep the nodes in the network synchronized. Trust between participants is established through these protocols, based on strong cryptographic principles.

It is possible to distinguish many categories of DLTs according to specific characteristics. The first factor is the data structure chosen to store the information. The most popular are blockchains and directed acyclic graphs (DAGs). As the name suggests, blockchains store information in blocks linked together by hash pointers. This mechanism is natively tamper-proof since changing a block would break the chain. A directed acyclic graph is a data structure no longer organized as a linked list of blocks but as a cycles-free directed graph. DLTs can be divided into two different access models: permissionless and permissioned. In the first model, the ledger is public and open access, hence, anyone can participate in the network and the consensus protocol. It is fully decentralized across unknown parties. In the second model, participation is mediated by permissions: participants have restrictions on writing, or both reading and writing. Both models are partially decentralized. DLTs can also be classified into tokenized and tokenless ledgers. In a tokenized ledger, transactions involve some type of purely digital asset (token) represented within the ledger. Tokens generally serve two main purposes: (i) as an economic incentive for protocol participants to form consensus in decentralized systems, (ii) as a way to prevent spam and DoS attacks, since each operation involves a nominal fee and creating a large number of transaction is expensive. In a tokenless scenario, the ledger does not offer any incentive to join and does not expect any payment for implementing smart contracts. For this reason, tokenless ledgers are typically permissioned and thus a strong trust has already been established during the registration process. Some ledgers also allow the simulation of a Turing machine. For example, Ethereum or Hyperledger Fabric can execute Turing machines. This allows programs written in Turing-complete programming languages to be stored and executed directly on the ledger. These programs are often called smart contracts.

4.1.2 IOTA

First-generation blockchains exhibit significant efficiency issues [Cha+18] that make them unsuitable for environments where resources can be extremely heterogeneous (i.e., IoT). IOTA [Pop] is a next-generation DLT engineered to tackle the scalability limits of the blockchain while still providing the same security ca-

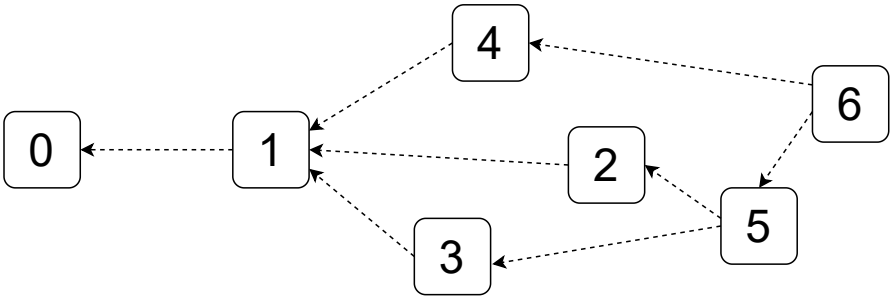


Figure 4.1: The tangle.

pabilities such as immutability, traceability, and transparency. IOTA owes its high scalability to the adopted data structure named *Tangle* (sketched in Fig. 4.1), a DAG composed of several connected nodes that store transactions. Each node is a transaction, while each edge represents a validation of that transaction. The tangle enables achieving remarkable performance due to the lack of a middleman since there are no block producers (i.e., miners and validators). Thus, everyone can submit a transaction and attach them to different nodes. However, in order to achieve a secure shared state, a new transaction has to verify the two transactions to which it is directly connected. Furthermore, since transactions are not validated by someone that has to be rewarded, it also enables zero-value transactions. This feature is particularly relevant for certain scenarios where there are a huge amount of data to send resulting in an extremely large number of transactions. Since zero-value transactions do not involve any transfer value, they are attached to the tangle without the need to be validated by participants of the network (i.e., double spending cannot occur), thus, remarkably reducing the time to share information.

An IOTA network can be deployed as private or public. A private network only provides access to certain users. On the other hand, a public network can be accessed by anyone without any kind of restrictions: every participant is aware of the history of transactions and sent new transactions.

IOTA distinguishes clients and nodes. A client is any entity (i.e., human or not) that submits transactions to a node, to have them attached to the tangle; nodes have to verify the correctness of the transactions and in case of success, add them to the tangle. Furthermore, an IOTA network comprises additional node types

named *Coordinator* and *Permanode*. In each IOTA network, there is a unique Coordinator that regularly produces *milestones*, trusted signed transactions used by nodes to confirm transactions. The signature guarantees that nobody can fake the signatures on milestones, thus, milestones are always legit. In particular, a transaction is confirmed only when directly or indirectly referenced by a milestone that nodes have validated. The use of the Coordinator is temporary, it will be soon removed in incoming updates. Permanodes are responsible for keeping the history of all the transactions that occurred. Such a component is particularly relevant in specific scenarios since nodes may be constrained devices that cannot memorize the entire tangle. Therefore, they periodically delete recorded transactions using a pruning operation.

4.2 Related Work

This section reviews existing works on the integration of blockchain with SDN and P4. Furthermore, it also analyzes some research efforts that employ P4 for thwarting SYN flooding attacks.

4.2.1 SDN and Blockchain

The combination of SDN and blockchain can find various applications, motivated by both the necessity to address SDN's inherent security issues and infrastructural problems such as limiting energy consumption of devices [RR17]. Some researchers exploited the distributed nature of blockchains to secure IoT infrastructures. Yazdinejad et al. [Yaz+20a] proposed a scheme to provide an efficient and secure mechanism to transfer files between IoT devices, to overcome the computational limitations of such devices. IoT devices are clustered around their respective SDN controller and are able to communicate over a P2P network using a public blockchain. The computational need is eased by removing the Proof of Work (PoW) process thanks to the controller's role, the clustered nature of the architecture, and an ad-hoc distributed trust algorithm. Inside the clusters, a private blockchain is used to keep track of the newly added IoT devices and every transaction. To transfer a file between devices, a preliminary block that contains

the sender and receiver signature and public keys is designed. After the block is validated by the network, the file is then sent to the intended recipient, which is the only one that can decode it. A similar use case is shown in [BS17], where with the use of OpenStack and Pythereum, a blockchain-enabled SDN is implemented and tested. The role of the blockchain in this architecture is to present indelible and transparent records of any file transfer, that the network then validates. Jiasi et al. [Jia+19] present a proof-of-concept practical design in which a blockchain layer is placed between the control and data layers, to record network events and resources associated with every controller and build smart contracts that automatically implement security protocols.

To tackle the Single Point Of Failure (SPOF) architectural vulnerability, while having the purpose of enhancing SDN's security level, Abou et al. [AEHHK19] propose an architecture that incorporates the blockchain as a way to make multiple SDN-based domains collaborate and share DDoS attack information in a decentralized manner. This work exploits a Smart Contract where collaborators can publish and share blacklisted IPs. The authors deployed the Smart Contract on an Ethereum testbed network. The choice of deploying the solution on a public blockchain enabled information sharing between different clouds to achieve collaboration, especially needed in IoT environments as shown in [TPK17].

Rahman et al. [Rah+21] present a framework that exploits the Ethereum blockchain to publish all the flow rules of the switches: the controller periodically creates a block as an update only if all the switches agree on the proposed list of rules. The immutability and consistency of the blockchain allow the management of flow rules and the detection of their violations on devices. However, the authors reach the conclusion that deploying this kind of architecture in the real world is rather complex because of the amount of transaction needed, which can entail a considerable cost. Similarly, Sharma et al. [Sha+17] present a distributed secure SDN architecture for IoT using the blockchain technology concept to improve security, scalability, and flexibility, without the need for a central controller. The blockchain is employed as a distributed peer-to-peer network where non-authenticated members can interact with each other without a trusted intermediary. The blockchain is deployed in order to allow untrusted interactions to update a flow rule table, securely verify and validate a version of the flow rule

table, and download the latest flow rules table for IoT forwarding devices. In addition, the DistBlockNet architecture provides proactive and reactive incident prevention by dynamically adapting to the threat landscape without having to include security administrators to manually process a huge number of advisories and approvals.

4.2.2 Blockchain Interaction with P4-enabled Switches

The efficiency of the solutions described in section 4.2.1 is validated by comparing it with existing models, as highlighted in [Als+21]. However, we believe they do not fully exploit the potential of the programmable data plane. To the best of our knowledge, only few works attempted to integrate blockchain and SDN by leveraging the P4 language.

Febro et al. [Feb+22] present a botnet DDoS defense framework using P4, SDN, and blockchain at the network edge. It implements a synchronized defense within an organization or also spanning multiple organizations. The framework comprises two main agents, ShieldSDN and ShieldCHAIN. ShieldSDN is an SDN controller managing edge devices with P4 capabilities, responsible for synchronizing packet filters provided by the switches within an organization using a smart contract deployed in the Ethereum blockchain. It is responsible for inter-organization synchronization: when a publisher or organization wants to share attack fingerprints with the community, ShieldCHAIN creates a transaction. The subscriber can then retrieve the current state of attack, and ShieldCHAIN then installs the required countermeasures in the data plane. The authors performed four experiments to validate their solution against botnets-based DDoS. The main drawbacks of this solution include requiring high computational power to run the public blockchain and with that added overhead.

Yazdine et al. [Yaz+20b] propose a P4 and blockchain-enabled packet parser (BPP) in the data plane and implemented on FPGA. BPP implements a custom header in P4 that fits the blockchain structure. BPP is able to recognize blockchain hash blocks to enforce control policies, such as match+action tables using specific fields in these packets. The workflow outlined by the authors involves the data plane, the control plane, and the application layer. It begins with packet pro-

cessing in the BPP module, which leverages the intrusion detection functions in relation to attack types (five categories of patterns: Normal, Remote to Local (R2L), DoS, User to Root (U2R), and Probing Attack (Probe)) to detect attacks. If an anomaly is detected, a transaction is prepared to be validated and then added to a block in the ledger for that specific attack. Then, the BPP submits a transaction to a validator in the control plane and alerts the controller. Subsequently, the SDN controller can use a Merkle Tree to evaluate the transaction. If this process is successful, the transaction will be marked as valid in the whole network. Concurrently, the blockchain will be updated and the BPP will be re-programmed, if necessary.

Ref.	SDN	P4	IoT	Ethereum	Recording events	File transfers	Security
[Yaz+20a]	X		X	X		X	
[BS17]	X				X		
[AEHHK19]	X			X			X
[TPK17]	X		X		X		X
[Rah+21]	X		X	X			X
[Sha+17]	X		X				X
[Feb+22]	X	X	X	X			X
[Yaz+20b]	X	X					X

Table 4.1: Related works list of analyzed topics.

4.2.3 P4 for Thwarting SYN Flooding Attacks

SYN flooding attacks are a type of attack that attempts to concurrently establish a large number of connections to disrupt the networking capabilities. The attack involves flooding the network with SYN/ACK TCP packets targeting a host in an effort to establish TCP connections and block the available ports. P4 is a promising candidate for designing and deploying in-network detection and mitigation strategies for SYN flooding attacks. It allows keeping track, in real-time, of the ratio of SYN/ACK sent in a flow compared with the corresponding ACK-/FINs. The literature shows how real-time data plane detection enables accurate

assess whether a network flow is malicious or not. One such implementation is reported in P-SCOR [Mel+20], where a simple asymmetric-flow detection algorithm is proposed.

Shen et al. [She+21] propose a P4-based SYN and UDP flooding mitigation strategy that combines the 2 steps of attack identification:

1. Source authentication: by using an SYN cookie, the source of traffic authenticates in the system.

2. Anomaly detection: after the authentication step, the real three-way handshake takes place and the P4 program supervises the correctness of the process.

Such operations allow early detection of ongoing attacks on the authentication phase and prune the remaining malicious traffic in the second phase. The results show a drastic reduction in the server SYN queue usage when the P4 firmware is deployed in the network. Similarly, another three steps solution is proposed by Lin et al. [Lin+20]. Differently from[She+21], the authors merge overlapping switch rules to minimize the number of dropped benign flows. Three concurrent components make up the solution. A detection oversees the ratio of SYN/ACK and ACK/FIN packets related to each network flow. Meanwhile, a merging phase is employed: here, multiple entries on a switch are simplified to a larger prefix to minimize the number of installed rules. The defense mechanism matches each attacker in a flow table rule: if the rule installed in the switches is LPM and an attacker is in that IP range the rule is deleted, while if the rule is an exact match the IP is dropped.

SYN flooding attacks are one of the most threatening attacks in distributed environments, hence, we claim they can be a remarkable example to showcase the potential of our solution.

The scientific research on P4 solutions that incorporate blockchain technologies is limited. Most of the existing solutions choose the Ethereum blockchain, and the focus is mainly on security applications in IoT environments. There is a lack of studies on P4 and blockchain in cloud environments and few works that leverage the blockchain to propagate P4 alerts in the case of detected attacks. To the

best of our knowledge, there are no works that employ alternative DLTs such as IOTA, which has been shown to be more efficient than popular blockchain-based solutions, especially for IoT environments.

4.2.4 Considerations about P4 Employment in Blockchain Solutions

The scientific literature on P4 solutions that exploit blockchain technologies is very limited, with most of the existing solutions that choose the Ethereum blockchain. Table 4.1 shows what are the main topics treated by each paper mentioned in the related works. These works mainly focus on security applications in IoT environments. Two of the analyzed works focus on recording events, while only one covers file transfers. By reviewing the literature, we observed a lack of studies on P4 and blockchain in cloud environments and few works that leverage the blockchain to propagate P4 alerts in the case of detected attacks. On the other hand (as shown in Section 4.2.3), a large number of works use P4 as the enabling technology to deploy detection strategies to spot abnormal behaviors on the data plane, i.e. SYN flooding attacks.

To the best of our knowledge, there are no works that employ alternative DLTs such as IOTA, which has been shown to be more efficient than popular blockchain-based solutions, especially for IoT environments.

4.3 P-IOTA Architecture

In this section, we present the design of our solution, which is depicted in Fig. 4.2. P-IOTA is tailored for distributed networks that belong to multiple organizations, such as cloud infrastructures, where computational and networking resources are often spread across geographic locations and critical information is stored. These types of networks are often treated as local networks (e.g., cloud-hosted installations) by system administrators, but it can be challenging to quickly detect and block threats while spreading alerting information in such a distributed environment.

The main goal of this framework is to facilitate the dissemination of network

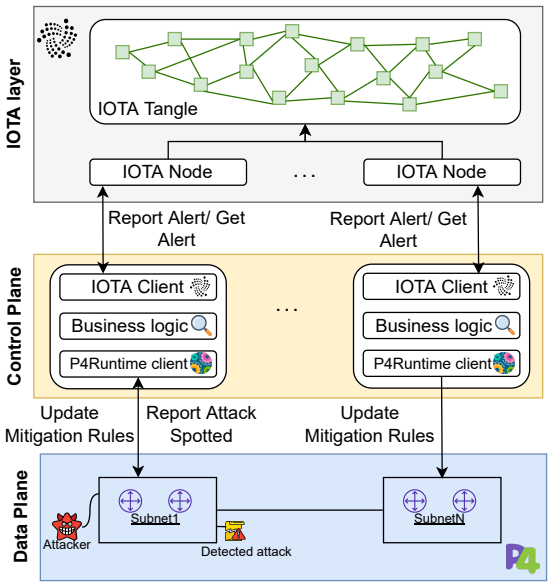


Figure 4.2: The P-IOTA architecture.

attack alerts through IOTA. In order to generate alerts in a highly customized and efficient way, P-IOTA leverages an SDN-based architecture, involving a P4 data plane layer that detects network attacks. The generated alerts are then used to notify other controllers through the IOTA layer. The infrastructure consists of three main components:

- IOTA Layer: whose main role is to notify and log alarms from the data plane, and to share mitigation strategies. The IOTA layer notifies the portions of the network that can be impacted by the detected attack and disseminates the policy that should be applied to mitigate it.

- Control Plane: is responsible for managing and configuring the underlying physical network. It contains multiple local network managers (i.e., controllers), each of which controls a specific subnet. The controllers interact with the IOTA layer using an IOTA client and with the data plane using P4Runtime.

- Data Plane: this layer hosts the physical devices that forward traffic. By using P4 and the programmable data plane, part of the detection intelligence

can be moved from the control plane to the data plane. This allows for deep packet inspection and network-level probing to detect anomalies.

P-IOTA is designed to propagate real-time alerts from the data plane and deliver them quickly throughout a distributed environment. The IOTA tangle maintains an immutable list of alerts in the form of a log, allowing for further investigations on the attack history offline without the risk of log cleaning. Moreover, the tangle can be leveraged to share the countermeasure needed to mitigate detected attacks. Hence, our solution offers intrusion detection capabilities on the data plane in the form of in-network detection, offloading a significant amount of detection intelligence to networking devices.

In this chapter, we demonstrate how IOTA significantly reduces the overhead compared to traditional blockchain solutions, highlighting its potential in the network security field.

4.3.1 IOTA Layer

The IOTA layer comprises the IOTA nodes that hold a unified view of the tangle. Our decision to use IOTA to share information across different sites [Maz+22] is motivated by the following features:

- Efficient lookup: each transaction can be tagged, making it easier to collect IPs from the tangle. If another DLT were adopted, an additional tag within the transaction message would significantly slow down the time it takes to find a transaction.
- Zero-value transactions: IOTA enables neglecting cryptocurrency, reducing the complexity of managing IPs. In contrast, each controller would need to have sufficient funds to perform operations.
- Scalability: the tangle allows parallel validation of transactions without any intermediary. Such capability overcomes blockchain-based solutions, where a transaction is not recorded until it is stored in a block.

Finally, the tangle structure also shortens the time needed to record a new transaction. Transactions are recorded on the tangle as soon as they are created,

whereas, in blockchain-based solutions, they must wait until they are stored in a block.

IOTA Node

In a federation, each participating enterprise should have at least one IOTA node to receive notifications from other organizations. However, a company may not collaborate with external parties and may have multiple sites located in different regions. Therefore, to reduce latency and facilitate swift mitigation, a company may choose to deploy an IOTA node at each of its sites.

IOTA Tangle

The Tangle is the data structure employed to share information, such as alarms and mitigation, among different controllers. This information is shared through zero-value transactions that do not require validation and, hence, help maintain a unified view of the tangle while keeping low latency and energy consumption. As discussed in the previous section, these features make IOTA a suitable choice for SDN-based scenarios where threat alerts and mitigation have to be quickly disseminated among devices that may have limited capabilities.

4.3.2 Control Plane

The control plane is responsible for managing, configuring, and monitoring the physical network. It consists of multiple geographically dispersed controllers which are in charge of managing a single network. These controllers work together and receive alerts from the IOTA layer, which informs the correct nodes of potential attacks, as shown in Fig. 4.3. Each controller acts as the primary management point for a local network. It keeps track of the status of networking devices, communicates with other controllers to make decisions about local network management strategies, and provides common control place services to monitor and administrate the data plane.

The control plane is made up of multiple controller instances, which are connected through messages. The components of each controller are:

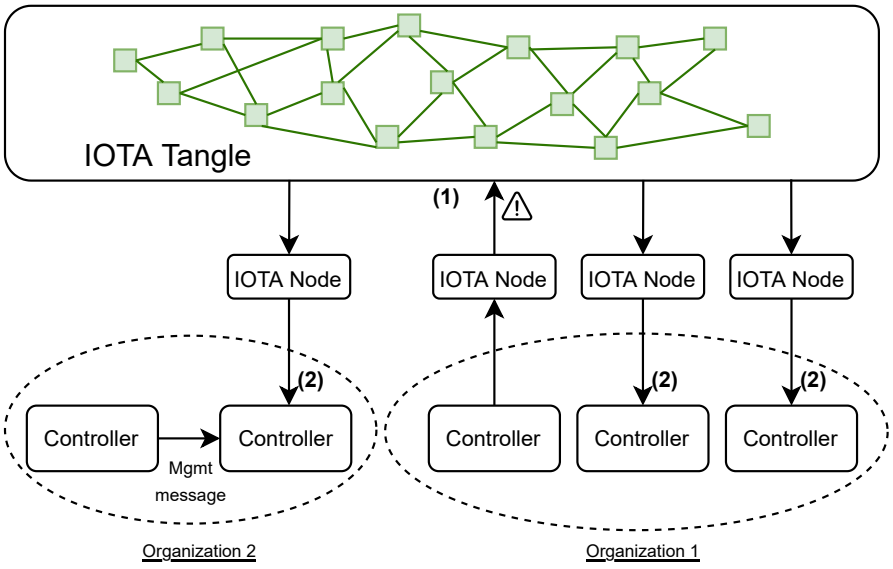


Figure 4.3: An example of organizations managing controllers associated with different subnets. In a federated environment, if an alert (1) is generated from a node, all interested controllers across organizations are notified (2).

- IOTA Client: which is responsible for connecting the controller instance to the corresponding IOTA node in the IOTA layer. It communicates with the IOTA node to send and receive alerts.

- Controller Business Logic: which handles the forwarding alerts to the IOTA client, sending management messages (e.g., congestion, link failures, etc) to other controllers, and communicating with P4Runtime.

- P4Runtime: which is in charge of interacting with the networking through the P4Runtime Southbound Interface. It receives alerts generated from the data plane and installs the rules to react to these alerts.

As highlighted in Fig. 4.3, organizations may manage multiple controllers, and if a controller detects a potential attack it will communicate it to the IOTA node, which then notifies the interested controllers through the IOTA tangle.

IOTA Client

The role of each controller in detecting and mitigating attacks is accomplished through the integration of an IOTA client. This client serves as a bridge between the controller and the IOTA node, allowing the exchange of information between the controller and the IOTA tangle. One of the advantages of using IOTA clients is their lightweight design, which makes them suitable for deployment on devices with limited resources.

Control Plane alerting and management

This component plays a key role in managing and controlling the underlying network. It is responsible for expressing policies and communicating configuration or resource changes to neighboring controllers via management messages. The management messages are used to exchange information between physically close controllers, while the IOTA layer is in charge of disseminating alarms and mitigation strategies across a distributed network. In summary, the main functions of a controller include:

- Management messaging: sending messages to communicate with neighbors' controllers.
- Alerting: forwarding alerts coming from the data plane to the IOTA node for further dissemination.
- Network configuration: interacting with and configuring the underlying network for forwarding or mitigation purposes.

This controller acts as the centralized core that manages the subnet and demonstrates the intelligence of the administration.

P4Runtime Client

This component is the client for the Southbound Interface that connects the controller business logic and the data plane level. P4Runtime abstracts the underlying

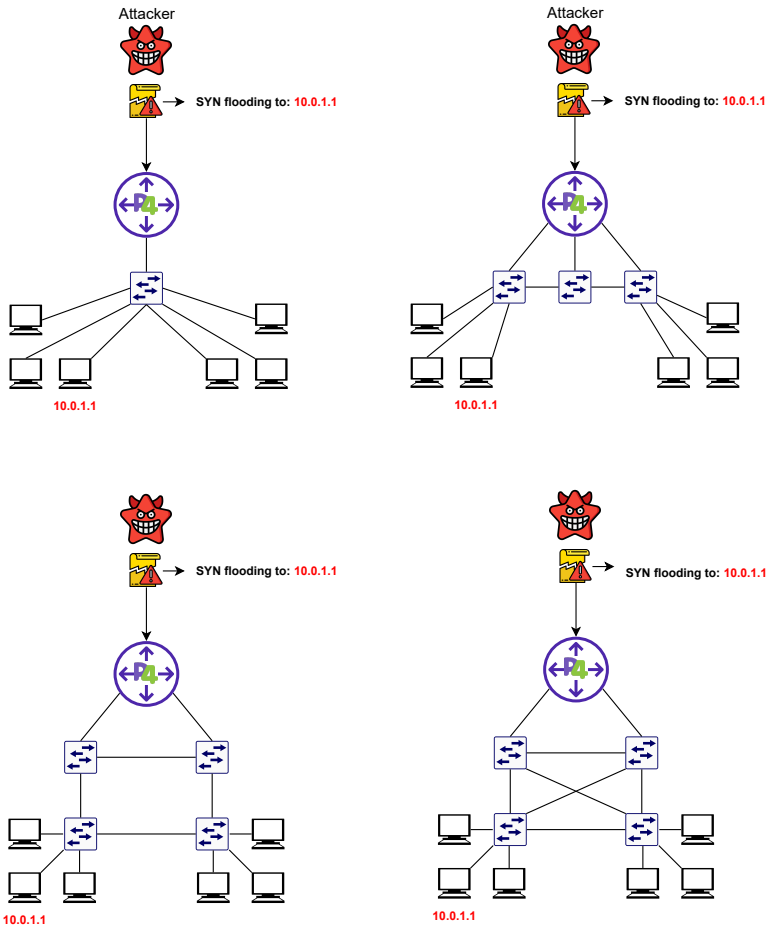


Figure 4.4: Network Topology used for the testing phase. From the top-left corner, clockwise: Single Switch Topology; Linear Topology; Ring Topology; Full Connected Topology.

hardware or software and offers agnostic APIs to the control plane to communicate with the physical network. The P4Runtime client is responsible for receiving communications from the data plane and performing two key functions:

- Installing match-action rules: it installs rules that specify forwarding logic or threat detection and mitigation strategies.
- Event listening: it listens for alerts from the data plane that indicate potential threats.

4.3.3 Data Plane

The data plane is in charge of processing and forwarding traffic. It includes networking devices such as switches and routers. Each controller is paired with one or more P4 border routers that are capable of monitoring the traffic flowing in a given subnet and detecting abnormal behaviors.

This allows the P4 switch to centrally inspect each network flow and determine if an attack is taking place. The programmability of P4 and the data plane has enabled us to design pipelines that incorporate the detection of ongoing network attacks and normal forwarding behaviors. In Fig.4.4, we report the network topology used for experiments.

Over different possible attack scenarios, we focus on the two we considered more relevant:

- An organization is comprised of multiple physical subnets within the IOTA-controlled network. If an attacker is detected, the alert must be propagated to each geographically dispersed network subnet.

- The same physical network is used by multiple organizations (such as in public or hybrid cloud platforms). In this scenario, an attack may potentially affect each organization operating in that portion of the data center.

P-IOTA handles both of these scenarios in a consistent manner since each IOTA node is tied to its organization. Similarly, as depicted in Fig. 4.4, each controller is connected to its IOTA node and can configure its network independently.

4.4 Case Study

To validate the proposed architecture and compare it with the existing literature, we consider a real-world use case scenario. This section aims to showcase a practical implementation of the SYN flooding detection and alerting workflow using the P-IOTA architecture. Therefore, we conducted a proof-of-concept evaluation of P-IOTA by focusing on SYN flooding, which is a common and harmful networking attack in distributed environments. This attack falls within the DDoS,

notoriously known to disrupt the network forwarding capabilities and to leverage SDNs to threaten cloud infrastructures [YY15]. As discussed previously, the programmable data plane in P4 can mitigate these threats, as it addresses the centralized nature of traditional SDN controllers. There have been several successful implementations of P4 in mitigating DDoS attacks, demonstrating the effectiveness of the technology in securing distributed networks.

We conducted a proof-of-concept evaluation of our architecture by implementing a SYN flooding scenario. To detect the DDoS attack, we used the InDDoS solution proposed by Ding et al. [Din+21]. This solution, which is fully located on the data plane, identifies potential DDoS victims based on data structures and thresholds. The solution has been validated with state-of-the-art datasets and has shown high detection precision. We deployed InDDoS using its open-source code available at [Din]. We selected this solution as it aligns with our scenario: the Southbound Interface is minimally used, with each alert consisting of just 4 bytes (an IP). To simulate the network environment, we used Mininet [Min21] and bmv2 with a single-switch and two host network topologies. The attack was generated using the Linux utility Hping3 [Lin22].

4.4.1 Experimental Setup

We set up an IOTA network and evaluated the time it takes to make all controllers aware of the victims' IPs. To do this, we used zero-value transactions to share information on the IOTA network. The transactions were embedded with the attacked IP and were made immutable by the tangle. However, this may lead to false positives if an IP is wrongly reported. Therefore, we enriched the message of transactions with an "action" field that indicates the type of operation being performed (i.e., add or delete). To delete an IP incorrectly detected as suspicious, a controller has to send a transaction where the action field is set to "delete" and the IP field reports the wrong IP. An example of the message structure used to share information is shown in Listing 1.

IOTA enables binding a tag to a transaction, simplifying how IPs are collected. Each controller retrieves all the transactions indexed by a specific tag and builds the firewall rule table. In case multiple subnets are simultaneously attacked, IOTA

```
{
   "action": <add | delete>,
   "IP": <IP>
}
```

Listing 1: Message structure.

will receive as many transactions as the number of detected attacks. All these transactions are indexed through the same tag. Hence, the controllers leverage that tag to retrieve all the corresponding alerts. The pseudocode of the algorithm implemented by IOTA clients is shown in Alg. 4.

Algorithm 4: IOTA Client

Input: –
Output: –

```
1  Procedure SendAlert
2      tag ← "newAlert"
3      action, IP ← getAlertFromController()
4      message ← createMessage(action, IP)
5      sendToIOTA(tag, message)
6  Procedure ReceiveAlert
7      tag ← "newAlert"
8      messages ← getMessagesFromIOTA(tag)
9      foreach m ∈ messages do
10         action, IP ← m
11         sendAlertToController(action, IP)
```

However, since the order of the collected transactions may be different from that of the detection, it is necessary to embed a temporal reference within each transaction, resulting in a different structure of the message shown in Fig. 1. The controllers then use this information to reconstruct the temporal order properly.

In the scenario described, the primary concern is to detect and notify an attack as soon as possible to minimize the attack window. Therefore, the following experiments were implemented:

- **Experiment 1: Notify of Detected Attack** - The first experiment aims

to evaluate the time needed to notify a controller from another organization about a detected attack.

- **Experiment 2: Update a Wrong Detection** - The second experiment is about updating a wrongly reported alert.

Furthermore, to better justify the effectiveness of our solution, P-IOTA's performances are compared to the framework presented in [Feb+22], which is the only work in the literature that employs a DLT for similar purposes. For the sake of fairness, we conduct the same experiments:

- **Experiment 3: Collect Alerts** - The third experiment measures the performances of the P4-based data plane layer.
- **Experiment 4: Publish** - The fourth experiment refers to organizations that share information, such as the victim's IP, with a community through transactions published on the tangle.
- **Experiment 5: Subscribe** - The fifth experiment involves community members that retrieve alerts previously published on the tangle.
- **Experiment 6: Packet Filter Installation** - The sixth experiment installs the appropriate filtering rules on switches based on the collected information to mitigate the ongoing attack.

4.4.2 Experiments

Each experiment was conducted by simulating a workload of 100 detected alerts, which was repeated 100 times for accuracy and consistency. In the scenario under consideration, the main objective for a community is to synchronize a defense posture in the lowest possible time, so our analysis focuses on the latency metrics required for the main operations. Fig. 4.5 shows the results of the first and second experiments. In Fig. 4.6, we devise the remaining experiments based on the components under evaluation. The results of the experiments that pertain to the SDN components can be seen in Fig. 4.6a for Experiments 3 and 6, while the results of the experiments related to the IOTA network are shown in Fig. 4.6b for Experiments 4 and 5.

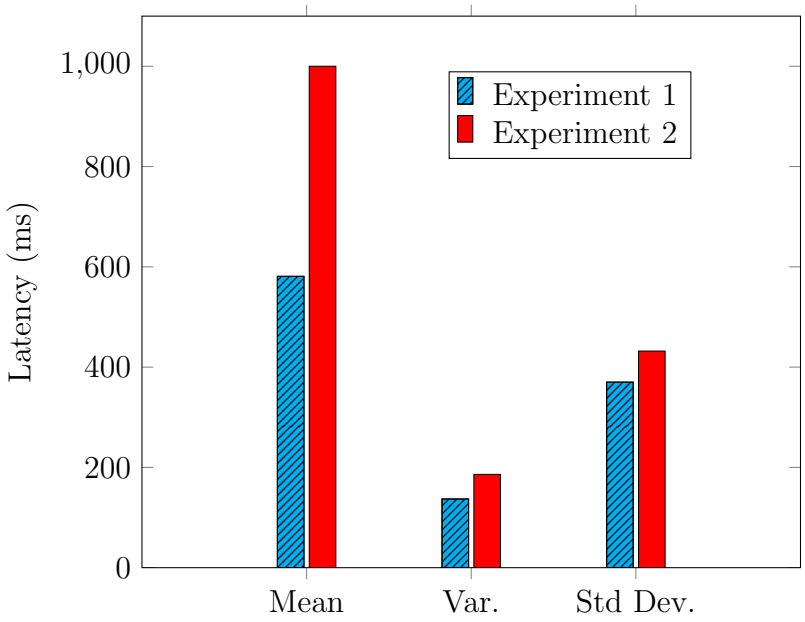


Figure 4.5: **Experiment 1 and 2** - Latency statistics.

4.4.3 Experiment 1: Notify a Detected Attack

Firstly, we evaluate the time that elapses between the notification of an attack by a controller and its availability to all the other controllers in the control plane. In particular, the elapsed time includes the creation of a transaction, its retrieval through indexing, and its conversion into a useful representation. The results are shown in Fig. 4.5, where two types of latency time (declined among mean, variance, and standard deviation) are represented. We can claim that a notification, reported in blue requires an average of about 500ms to make the alert available to other organizations.

4.4.4 Experiment 2: Update a Wrong Detection

In the second experiment, we evaluate the ability to update wrongly reported attacks. In this case, the average latency is almost doubled. We expected such an outcome due to the immutability feature of DLT. As a transaction cannot be removed from the tangle, modification deems two transactions: one to invalidate the previous one and another one to update it. The results are shown in red in

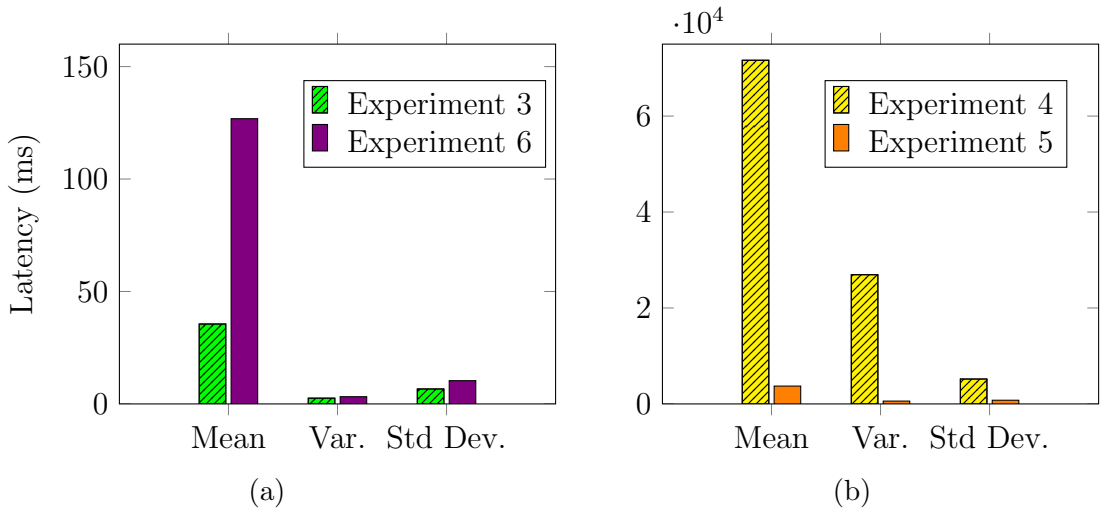


Figure 4.6: **Experiments 3, 4, 5, and 6** - Latency statistics for SDN components (a) and for the IOTA network (b).

Fig. 4.5.

4.4.5 Experiment 3: Collect Alerts

The third experiment measures the time required by the P4 target to generate and send an alert to the controller. This time is the sum of the latencies collected in the following three steps:

- Create the digest packet describing the alert;
- Send it over the P4Runtime channel;
- Extract the alert in the control plane.

A programmable P4 switch allows for describing custom features that improve the performance of certain actions. This is reflected in the results of this experiment, as P-IOTA is able to shrink the content of the alert up to 4 bytes, i.e. the IP address of the victim. The comparison between P-IOTA and[Feb+22], whose results are reported in Table 4.2, demonstrate that P-IOTA outperforms the compared approach by three orders of magnitude. Fig. 4.6a shows the mean, variance, and standard deviation of Experiment 3, collected over 100 measurements.

4.4.6 Experiment 4: Publish

The fourth experiment aims to demonstrate the effectiveness of our proposal in sharing threat intelligence with the community. The results of the experiment, which involves embedding each detected alert within a transaction, are shown in Fig. 4.6b. As the number of detected alerts increases, so does the number of transactions published on the tangle. Thus, detecting 100 alerts will result in 100 transactions published on the tangle. Our proposal performs better due to the fact that IOTA does not have the concept of blocks, allowing transactions to be attached to the tangle as soon as they are collected by underlying layers.

4.4.7 Experiment 5: Subscribe

The fifth experiment proves the advantages of using IOTA's index feature for retrieving transactions from the tangle. According to the results shown in Fig. 4.6b, it takes P-IOTA less than 4 seconds to collect 100 transactions, representing alerts. The close-to-zero variance and standard deviation indicate high consistency in the time taken to collect transactions. As anticipated, Fig. 4.6b also demonstrates that the average latency for reading transactions is significantly lower, by one order of magnitude, compared to the latency for publishing.

4.4.8 Experiment 6: Packet Filter Installation

The sixth experiment assesses the time to install a mitigation rule delivered through the IOTA layer. The rule is deployed by P-IOTA using the P4Runtime API and the Southbound Interface of P4 4.3.2. Similarly to the third experiment, we compared our solution to [Feb+22]. We demonstrate that P-IOTA outperforms the compared approach since we only install one rule to perform the mitigation (Table 4.2). Fig. 4.6a shows the mean, variance, and standard deviation of Experiment 6, based on 100 measurements.

4.4.9 Time and Computational Analysis

Time and computational analysis are critical in evaluating whether our proposal can be deployed in real-world scenarios. Fig. 4.7 outlines that, in the IOTA net-

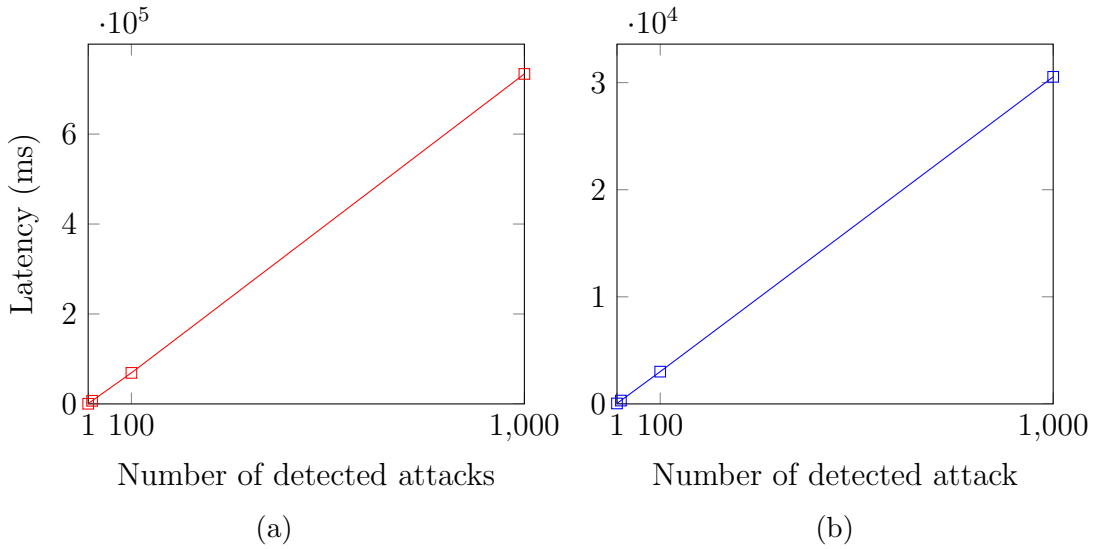


Figure 4.7: The latency trend of published detected attacks (a) and their retrieval (b).

work, the latency increases with an approximated $O(n)$ complexity as the number of notified attacks scales up, both for publishing detected attacks (Fig. 4.7a) and retrieving them (Fig. 4.7b).

Moreover, computational considerations are essential in evaluating the practicality and efficiency of the IOTA network. The IOTA tangle is designed to be lightweight and energy-efficient, making it ideal for deployment on low-power devices. Official experiments [IOT22] have shown that the IOTA network can operate successfully on devices such as Raspberry Pi 3 and 4 with very low energy consumption, ranging from 2J to 6J approximately. This is a significant advantage for the IOTA network, as it not only reduces its environmental impact but also makes it more accessible and cost-effective for a wide range of applications, including those based on SDN.

Regarding the SDN layer, Figure 4.8 illustrates the correlation between the number of alerts detected and the time required to forward them to the IOTA node. The graph shows a linear relationship for a rate of up to 7000 detected attacks. However, beyond that threshold, the latency increases gradually due to the limited bandwidth of the Southbound Interface, which has a maximum capacity of 14 Mbps in bmv2. It is worth noting that this test is not applicable to

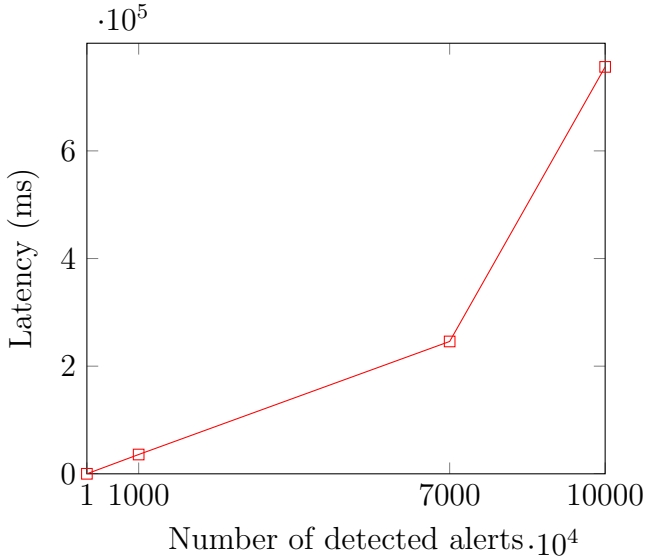


Figure 4.8: The relationship between the number of detected alerts and the latency in the Southbound Interface channel.

the retrieval phase. Installing thousands of rules on a switch can cause congestion in the match-action table, which should be minimally used.

4.4.10 Discussion

As a yardstick for comparison, we consider a proposal that uses Ethereum, which is one of the most widely used blockchains. In 2022, Ethereum switched to a PoS consensus protocol, with a block-adding time of 12 seconds as stated in the official documentation [Eth22]. However, adding a block to the chain does not guarantee its validity. To ensure the block's validity, it is necessary to wait until it is finalized, meaning it cannot be modified without a significant amount of ETH getting burned. In Ethereum, this is done through "checkpoint blocks" that are issued every 32 blocks added. If a pair of checkpoints attracts votes, representing at least 2/3 of the validators, all blocks prior to the least recent checkpoint are considered finalized. Therefore, it is necessary to wait for at least 64 blocks, approximately 12 minutes, to ensure block validity.

According to literature [Als+22; Auh+22; Reb+21], IOTA emerges as the best solution for the proposed case study because of its low latency, high throughput,

	Exp. 3	**Exp. 4**	**Exp. 5**	**Exp. 6**
[Feb+22]	64000 ms	517330 ms	100000 ms	80000 ms
P-IOTA	35.54 ms	71640 ms	3720 ms	126.83 ms

Table 4.2: Comparison between P-IOTA and [Feb+22].

and low power consumption. These are key features in scenarios where fast response times are necessary to mitigate attacks. Additionally, routing devices often have limited resources, making it imperative to use lightweight protocols like that of IOTA.

These considerations are also supported by the results in Table 4.2, which compare P-IOTA and [Feb+22] in terms of average latency. The experimental results show that our solution significantly outperforms solutions that adopt Ethereum technologies, decreasing the time taken to alert the other nodes, including the time to forward the alert from data to the control plane and the time to notify other nodes.

4.5 Discussion

In this chapter, we present P-IOTA, an architecture for detecting attacks and alerting potentially affected nodes that are geographically distributed. Our proposal leverages the P4 programmable data plane to implement the detection logic and uses IOTA to disseminate alarms to nodes belonging to the same organization or, in the case of the federation, to different organizations. P-IOTA also enables keeping the history of the detected attacks.

We implemented a prototype of our solution to evaluate its performance while reporting and notifying threat alerts during an SYN flooding attack. Specifically, we measured the latency in sending a notification and updating incorrect alerts. The experimental results demonstrate that IOTA enables these operations with a latency lower than 1 ms, outperforming traditional blockchains that typically take minutes to confirm a block.

In light of the foregoing results, we believe that this work proves that IOTA is a promising technology for alerting nodes about threats in SDN-based environments. It can be also leveraged to handle various attack scenarios in which multiple en-

tities need to be notified (i.e., threat intelligence). In future research, we plan to encompass mitigation policies.

Part IV

Shielding IIoT devices from intruders

Chapter 5

In-network computing for enhanced data integrity*

We previously focused on distributed network attack detection and alerting. In this Part, we are going to focus on the use of DPP to provide computation capabilities to low-end IIoT devices, to deliver more integrity or simple encryption. In recent years, due to the increasing pervasiveness of data processing capabilities across the whole network, service instantiation has been moving from core networks towards the edge of the infrastructure, leading to the emergence of the Edge Computing (EC) paradigm [Cap+19]. User devices as well as data sources can greatly benefit from this transition, as they are in constant demand for real-time and latency-sensitive processing [Yu+18]. Along with meeting these requirements, the adoption of EC also promises an optimization of network traffic, an improvement of the user experience, and an enhancement of privacy and security for a number of applications, including industrial automation. Indeed, EC can be pivotal in augmenting services offered by the network with new processing capabilities, making up for the scarcity of computing power or software compatibility typically displayed by specialized equipment, e.g., by providing low-power legacy devices with packet encryption functionalities. In doing that, DPP can be of great help, as devices in the PDP can easily be programmed to perform simple and quick

*Part of the material presented in this chapter is based on the following publications: Gaetano Francesco Pittalà et al. "Leveraging Data Plane Programmability to enhance service orchestration at the edge: A focus on industrial security". In: *Computer Networks* (2024), p. 110397

traffic processing, opening up to a large set of functionalities that can be offered to network services.

In industrial environments, clusters of sensors and actuators are typically deployed across the scenario, in arrangements often referred to as Industrial Internet of Things (IIoT) networks. They present specific architectural challenges and weaknesses [Sch+22], stemming from their peculiarities. These types of networks are usually heterogeneous, unsuitable for computation-intensive operations, and prone to security issues. In the mean time, these devices require to always be connected to the Internet while lacking energy-demanding security software modules. Data Plane Programmability (DPP)-based solutions can help tackle these problems, in multiple ways. The Programming Protocol-Independent Packet Processors (P4) language has emerged as a powerful tool to control Programmable Data Plane (PDP) devices, allowing network operators and service providers to define the behavior of the network devices at a packet-level granularity, enabling them to create custom forwarding and processing pipelines tailored to specific applications [KCBH21]. By using P4 to include cross-level headers, packet processing can be offloaded to a single or few network devices provisioned with more resources, supporting simple encryption or integrity checking. Moreover, DPP can facilitate deploying security countermeasures without touching the IIoT devices.

In this chapter we explore the potential of DPP to enhance service offloading at the Edge, focusing on industrial security applications. Including DPP in a framework for flexible service provisioning, we can enable the customization and adaptability needed to improve services offered at the Edge. This calls for the introduction of a service orchestration entity, capable of taking advantage of a pool of heterogeneous (computing and networking) resources to enable efficient and swift service provisioning. In other words, we abstract the functionalities of PDP devices and make them comparable to those of computing nodes, then we apply policy-based placement strategies to pick the most suitable resource to provide a given service. As a case study, we refer to an industrial environment, where the security of remote maintenance services can be enhanced by activating proper services offered at the Edge. We emulate the use case scenario to evaluate the performance of implementing data integrity functions on PDP devices in comparison to general-purpose computing nodes. Finally, leveraging on the results of the

emulation, we assess through simulation how the availability of P4 switches can improve the efficiency, performance, and scalability of service offloading. In the context of this work, we refer as *service* to the composition of functionalities (e.g., data processing, traffic steering, etc.) that may be deployed in a distributed way over the network and EC infrastructures, and typically offered to the user as a cohesive bundle. *Service components* (i.e., the single parts being composed) are the abstracted elements representing the underlying physical or virtual resources that can be configured to perform those functionalities.[1]

The remainder of the chapter is structured as follows. In Sect. 5.1 we offer a recollection of state-of-the-art security challenges for industrial environments, as well as solutions for service orchestration at the Edge. In Sect. 5.2 we introduce the architecture and working principles of the service orchestration framework we employed. In Sect. 5.3 we present the scenario in which this chapter is articulated, outlining the typical structure of an industrial network, its main challenges, and what issues we aim to solve with our proposed solution. In Sect. 5.4 we describe the topology for our emulated environment and provide its evaluation, along with a commentary of preliminary results that assess the potential of employing PDP devices in the delineated orchestration context. In Sect. 5.5 we showcase through simulation results the benefits offered by our approach in the offloading of services at the Edge, focusing on the described use case. In Sect. 5.6 we discuss our results and highlight future research directions.

5.1 Related Work

In this section, we first review research efforts that highlight the potential of employing DPP to cope with shortcomings of IIoT networks, with a focus on processing offloading to network devices. We then provide a summary of state-of-the-art solutions for the orchestration of services at the Edge. In doing that, we use the terms EC and Fog Computing (FC) interchangeably, as their definition may overlap in scenarios such as the one depicted in this work. Indeed, from the point of view of cloud service providers, EC resources are typically referred to as "near edge," whereas FC resources are classified as "far edge," justifying the terminology

[1] In the following, depending on the context, for the sake of readability we may refer to "the function performed by a single service component" simply as a *service*.

we adopt here [SC+19].

5.1.1 Data Plane Programmability for in-network offloading

DPP enables reshaping packet processing procedures to smartly exploit the hardware capabilities of networking resources. This architectural advantage can be leveraged to implement custom packet programming for multiple purposes. PDP devices can be employed to support security and ease communication within IoT environments.

Some work analyzing this matter can be found in the literature. P4 can be employed in IIoT scenarios, as argued in [SSG21] and in [Ves+20], which propose a framework to describe in-network computation tasks, respectively to support AI-based scenarios and event detection over a publish/subscribe architecture (using hardware targets like FPGAs and SmartNICs).

P4 can also enable communication over complex wireless networks, as argued in [Udd+18], as well as [Gyö+23] and [Eng+19], which provide examples of how to exploit P4 devices to enable peer-to-peer communication, manage link load and enable WiFi communication over distributed Internet of Things (IoT) wireless networks. The common denominator of these approaches is that P4 introduce little to no communication overhead.

One of the few DPP-based IoT architectural solutions for multi-access networks is proposed in [Tac+22], employing P4 and In-band Network Telemetry (INT) [Tan+21] over a P4 target to dynamically offload tasks on the network.

Hence, in-network offloading can help service management systems drastically cut some operational overhead and allow for finer-grained network inspection. This is especially relevant in environments revolving around IIoT, usually composed of nodes with limited computational power. Moreover, these networks use heterogeneous means of communication and require ad hoc clients and servers to send and receive information. Thus, DPP and P4 can be helpful to potentially offload part of the server-side computation since programmable targets can be used and programmed like regular nodes. In fact, a P4 switch can handle multiple or even custom protocols, and hence be used as gateways for communications inside the

plant, as well as run simple in-line computation when processing packets. These features allow for service offloading on the data plane, so that network devices can be instrumented to perform network services computation at line rate. In this chapter, we show the advantages of such offloading features in an IIoT context focusing on security applications.

5.1.2 Service orchestration at the Edge

The EC paradigm enables efficient and distributed computation closer to the network's edge, bringing significant benefits in terms of latency, privacy, and bandwidth utilization. However, effective resource orchestration in EC environments remains a challenging task, due to the dynamic and heterogeneous capabilities of such systems.

While substantial steps have been made to standardize service orchestration systems, e.g. the OpenFog [Ope17] and Mobile Edge Computing (MEC) [Mao+17] projects, open-source implementations of fully-compliant orchestration systems are still lacking. However, we present here a comprehensive overview of state-of-the-art orchestration systems, to give context to the system we relied on for our evaluations.

In [San+17] the authors propose a novel orchestration architecture for FC environments. Such architecture is divided into three tiers, namely "cloud tier", "edge cloudlets", and "edge gateways", and arranged by the distance from the user to the requested resources (e.g., the edge gateway nodes are positioned closer to the user than the edge cloudlets nodes, offering applications with better latency performance). The workload placement is regulated to meet the demands of fog applications. The authors of [DB+17] created a prototype EC orchestrator, addressing the heterogeneity of the IoT environment as well as the capabilities of involved devices and the imposed constraints. The orchestrator is logically centralized, and an agent needs to run in every controlled node, to manage local virtual instances (i.e., containers). They focused their evaluation on two critical performance factors, namely the time performance of the system while orchestrating different services in different configurations (e.g., varying image size, image location, etc.), and the success rate of the system in instantiating the services as

requested. A hybrid architecture to manage resources in the Fog-to-Cloud continuum is presented in [Vel+17]. It suggests a solution for the distributed management of applications and services in the IoT and Fog domains, while it recommends using a centralized strategy for the orchestration in the Edge and Cloud domains, to benefit from global awareness of the available network resources. In [Ala+18], the authors provide a layered and modular architecture with containerized services and microservices that operate on the Fog-to-Cloud continuum. A hierarchically lower layer is responsible for performing sensing and operations, while a middle layer handles intermediate computing resources and routing, and an upper layer deals with wider-view operations such as long-term global storage. The aforementioned works propose new architectures for service orchestration in EC scenarios, emphasizing the distributed character of these systems and the demand for accurate monitoring and data on the availability of resources for both nodes and services. These solutions, however, only combine the information on available resources to pick a node on which to deploy the service, without considering different service provisioning models, nor considering the specific performance users expect of services after their activation. On the contrary, the authors of [Dav+21] propose a service-centric approach, that leverages the inherent flexibility offered by the cloud-native Everything-as-a-Service (XaaS) model to provision services in a more dynamic way. This solution still makes use of data gathered from available resources to decide where and how to deploy services, but in doing so it also considers the nature of the requested service, including the possibility of deploying it in multiple ways, along with the present state of the system. This orchestrator was subsequently extended in [Pit+22] and in [DC+23a], making it a suitable choice for our purposes here. The novelty of this work resides in (i) the adoption of PDP devices for the offloading of security-critical network service functions in an IIoT scenario, and (ii) the inclusion of DPP as a tool to enhance availability and flexibility of services in a XaaS-aware service orchestration framework.

5.2 Orchestrating Services using Programmable Data Planes

Orchestrating services over a heterogeneous set of resources requires to coordinate monitoring, management, and decision processes. In this section, we describe the structure, functional elements, and working principles of the service orchestration framework we employed. Such framework is based on that introduced in [Dav+21; Pit+22]. It retains the original XaaS approach, which makes the system aware of different service deployment models, borrowing from the XaaS deployment paradigm typical of Cloud Computing scenarios, and extends it with the support for services offered by networking resources. Specifically, the orchestrator can instantiate services according to five different service provisioning paradigms:

- Infrastructure-as-a-Service (IaaS), for the deployment of a generic virtualization engine (e.g., Docker) on a computing resource;
- Platform-as-a-Service (PaaS), to provide the user with a software framework (e.g., the Python SDK) including tools, libraries, and interpreters, for the execution of generic programs;
- Software-as-a-Service (SaaS), to offer a specific application (e.g., a Web-based one) that users may access through a dedicated interface;
- Function-as-a-Service (FaaS), to provide the user with a lightweight service component, reduced to a single function (e.g., real-time video transcoding), handled entirely by the computing resource on which it is deployed in an event-driven, serverless manner;
- Programmable-Data-Plane-as-a-Service (PDPaaS), exposing packet level processing a networking resources programmability (i.e., PDP devices) to the user, e.g., to steer traffic or process packets as they cross the network, with potentially significant performance advantages.

Leveraging the flexibility offered by this approach, the orchestration system can combine different paradigms to obtain the desired service efficiently and effectively. For instance, if a service is not natively available on a computing resource, but such

resource is capable of hosting a compatible version of it, then the requested service can be deployed there. In other words, the orchestrator may decide to deploy an application (i.e., a SaaS-native element) on top of a virtualization engine (i.e., a IaaS-native element) running on a computing resource that did not previously offer that specific application. This introduces great adaptability, albeit at the expense of greater service activation complexity that may entail drawbacks such as an increased service activation time.

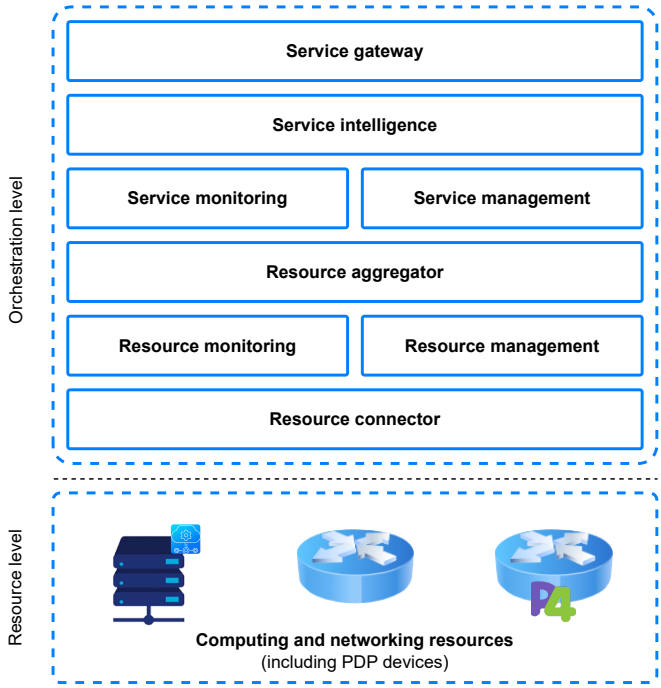


Figure 5.1: Service orchestration framework architecture.

The orchestration system architecture is represented in Fig. 5.1. At a macroscopic level, the framework consists of two layers, namely the *Orchestration level* and the *Resource level*. The former comprises the functional elements of the logically centralized orchestrator, while the latter encompasses all the distributed and dynamic network resources available in the infrastructure for the orchestrator to activate services.

The functional elements of the service orchestrator are structured in a way that reflects the need for abstraction of the service activation process, while also facilitating a modular implementation. Starting from the top of the figure and

moving downwards, the point of contact between the orchestrator and the external world is provided by the *Service gateway*, which allows users and other systems to access the functionalities of the orchestrator. Immediately below, it comes the *Service intelligence* element, which implements the core functionalities of the orchestration process; it is tasked with making service activation decisions to satisfy external requests while enforcing predefined policies. The monitoring information is made available to the decision-making module by the *Service monitoring* element, which handles the processed telemetry data that describes the status of active services and the availability of components to instantiate new ones. Its functions are complemented by the *Service management* element, which handles the lifecycle of services, ensuring that their deployment and decommissioning is operated in compliance with the decisions coming from the *Service Intelligence*. The role of the *Resource aggregator* element is that of providing the abstractions that allow the elements above to work with abstracted service components that lack any technology-specific details. Similarly to their overlaying counterparts, the *Resource monitoring* and *Resource management* elements are in charge of collecting telemetry data and handling the deployment processes, respectively, but at a lower level of abstraction, interacting with resource domains with their specific interfaces. Lastly, the *Resource connector* element is in charge of the communication between the service orchestrator and the underlying infrastructure, facilitating the integration of diverse domains into a pool of resources visible to the orchestration processes.

Here, we introduce the support for some additional services that may leverage DPP, depending on the availability of PDP devices. We make the practical assumption that the code needed to provision the newly introduced services is already available in the PDP devices, to simplify their deployment by just activating them when needed, without running a re-configuration of the pipeline.

Activating a service through the orchestration system typically starts with the intended user requesting a list of the offered services. This list reports information in a symbolic format, representing the available services as well as those that the orchestrator can deploy. The user may then request the activation of a specific service. The activation process is entirely handled by the cooperation among the functional elements in the service orchestrator. The steps of such process are

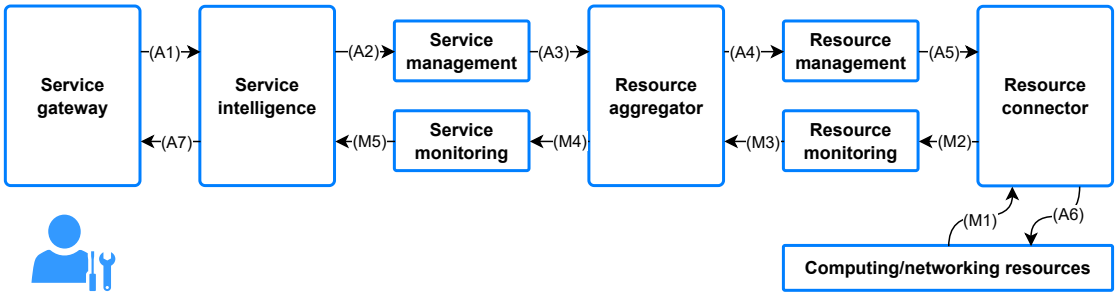


Figure 5.2: Interactions between functional elements of the orchestration system.

displayed in Fig. 5.2, where the interactions pertaining to the activation of services are labeled with A, whereas those related to the monitoring process are labeled with M. The former chain of actions is triggered by the request coming from the user, while the latter happens periodically. The numbers on the labels represent the order in which those interactions take place. In summary, when receiving a service activation request, the orchestrator will leverage the periodically refreshed monitoring information on the available resources to determine how to activate the requested service. It will then trigger the required steps to configure the underlying resources for the provisioning of the service to the user. At the end of the procedure, the orchestrator will inform the user of the outcome, allowing it to access the service. To showcase the support for the PDPaaS paradigm, we extended the orchestrator with the ability to interact with the PDP. From the point of view of the orchestrator, and specifically of its *Service intelligence* element, all resources are comparable, as they are described in terms of their features by means of abstractions provided by the *Resource aggregator* element. This makes it so that PDP devices are regarded in the same way as any other resource, allowing PDP resources to be included, alongside computing resources, in the same pool from which the placement algorithm picks for the activation of the requested service. Also consistently with the implementation of the other instances of the XaaS paradigm, the same principles for the monitoring and management of computing resources are applied to PDP resources. In practice, this translates to the introduction of a REpresentational State Transfer (REST) interface on the control plane of the programmable switches, enabling the interaction with the orchestrator for monitoring and management purposes.

The principles and mechanisms described in this section can be leveraged to address significant issues in the reference scenario, as detailed in the following.

5.3 Industrial security: a focus on legacy devices

In this section, we outline the Industrial Control Systems (ICS) context to which the use case discussed in this chapter pertains. We specifically describe some of the most important challenges on the orchestration and management of services in this context, and how our solution can help tackle some of those challenges, such as those related to remote maintenance.

ICS are composed of interconnected Cyber and Physical components that monitor and manage physical processes. They are responsible for the safety and operations of the industrial process, which implies the management of heterogeneous hardware and software. They include devices such as sensors, actuators, supervisory control and data acquisition (SCADA) systems, Human Machine Interfaces (HMI), and dedicated subsystems such as programmable logic controllers (PLC) [CDT21]. This heterogeneity obviously translates into system complexity, which implies more effort to manage and prevent anomalies. The Purdue Enterprise Reference Architecture [Wil94] is the reference networking architecture for ICS systems, adopted in the ANSI/ISA-95 standard, and we can use it to analyze each segment of a typical ICS.

As depicted in Fig. 5.3, the Purdue Architecture divides the ICS network into six layers which are arranged into three logical segments: the layers from 0 to 3 constitute the Manufacturing Zone, while levels 4 and 5 constitute the Enterprise Zone, with a Demilitarized Zone of convergence between them.

The Enterprise Zone, also referred to as Information Technology (IT) network, incorporates traditional IT devices and systems where the primary business functions of the enterprise occur, including the orchestration of manufacturing operations and services. On the other hand, the Manufacturing Zone is known as Operational Technology (OT) network because it contains systems and devices responsible for the control, monitoring, and automation of physical processes. At level 0 of the Manufacturing Zone, sensors and actuators are deployed to interact directly with the physical process while level 1 is composed of PLCs which im-

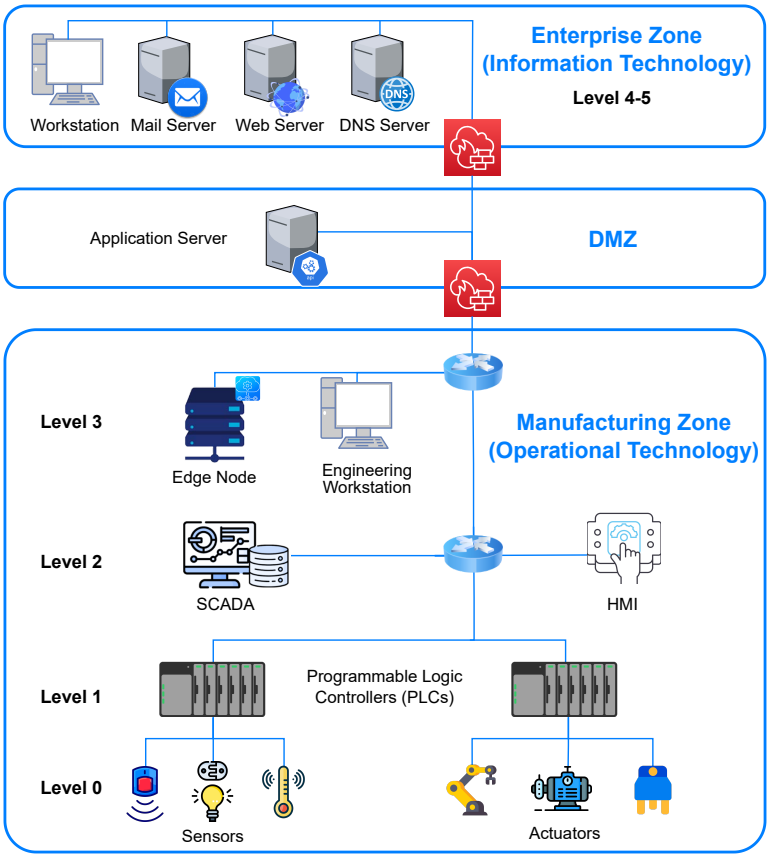


Figure 5.3: ICS Purdue Reference Architecture.

plement systems control logic by observing sensor readings and by consequently updating actuators signals. Level 2 (SCADA, HMI) and level 3 devices are responsible for control, data acquisition, and monitoring in order to manage plant operations. Edge nodes at level 3 are also responsible for running applications that need to interact with OT devices and for providing security for the Manufacturing network. In addition, devices belonging to those two levels can communicate with the Enterprise Zone through the demilitarized zone (DMZ), which manages the connection between the IT and the OT networks while maintaining the two worlds isolated from each other. The DMZ serves as a controlled buffer zone, enabling secure data exchange and access management while maintaining distinct security protocols suited to each network's priorities: IT focuses on data integrity and confidentiality while OT prioritizes system availability and physical safety.

Additionally, the DMZ architecture aids in regulatory compliance by establishing a clear boundary that can be audited according to industry standards [Iso]. The reliability of the OT network is paramount: failures cannot be acceptable due to the critical nature of the physical processes monitored and faults would imply shutting down the entire industry, leading to potential economic losses. Finally, it is important to highlight that the risk impact is also different considering that, in the IT network, the principal risk is the loss or unauthorized alteration of data. Instead, in the OT environment, a security breach can jeopardize both production and equipment, while in the worst case, can cause a loss of lives or environmental damage [CDT21]. This alignment with cybersecurity best practices underscores the DMZ's vital role in safeguarding industrial infrastructure, making it an essential component of modern network design.

5.3.1 Threats on legacy devices: the OPC UA protocol

ICS employs a wide range of protocols, depending on the specific objectives of each system. Real-time constraints and legacy hardware are two of the most important challenges that industrial protocols are specifically made to address. Legacy components in particular are one of the main source of issues in modern ICS. Legacy devices usually lack proper security measures. At the same time, protecting and managing these devices becomes necessary because replacing them is often a complex and expensive process. These devices typically show limited flexibility and frequently lack the capability to be updated to meet modern standards. As a consequence, these devices may exhibit shortcomings such as the absence of mechanisms for firmware management. This often leads to issues derived from outdated software (SW) and firmware (FW) such as limited connectivity, insecurity of channel communication, unverified data integrity, uninsured data confidentiality, or lack of access control monitoring policies.

In this context, a new building model, OPC UA [OPC22a] has emerged as the de-facto standard for machine-to-machine communication because, compared to other common industrial features, it enables platform-independent and secure communication by design. It has become popular with the advent of Industry 4.0, a paradigm that aims to create new business logic and markets while opening the

old OT industrial segment to the Internet, legacy devices included.

Adopting this protocol in an industrial environment enables the integration of heterogeneous hardware, which is instead a constraint brought by proprietary protocols (like Siemens S7). At the same time, the advanced security-by-design capabilities of the protocol reduce some of the typical security risks of ICS (such as lack of message authentication or encryption). Two communication strategies are possible, namely client-server and publisher-subscriber. Even though OPC UA does not strictly enforce the use of security mechanisms, both communication models allow messages to be signed to ensure authenticity and encrypted to add confidentiality. Actually, OPC UA messages can be exchanged in one of three Security Modes: *None* for unprotected communication, *Sign* for authenticated communication, *SignAndEncrypt* for authenticated and encrypted communication [OPC22b].

Despite the benefits of the protocol, supporting the security features of OPC UA by product vendors, libraries implementing the standard, and end-users remains challenging, preventing companies from adopting proper security mechanisms. Currently, roughly 14.6% of OPC UA device vendors do not support security features at all, while 64.6% of them present issues or errors in the Trustlist management, enable Rogue Client, Rogue Server, and Man-in-the-middle attacks, and only 20.8% of them correctly implement the security features offered by the OPC UA protocol [EMT22].

We argue that, despite its potential, the OPC UA architecture needs to be backed by additional mechanisms when the adoption of its security features is limited or nonexistent.

5.3.2 Use case: Remote Maintenance of Industrial Plants

With the ever-growing need for operational efficiency and negligible downtime, remote maintenance is becoming a critical tool for industrial plant maintainers. In fact, the possibility of connecting with the industrial plant remotely (e.g., from the premises where the IT is located) enables quick ordinary reconfiguration and prompt reaction to unexpected events, at all times. In this section, we describe a typical remote maintenance use case and outline the security threat that may

occur.

In our use case scenario, a technician needs to operate on equipment situated in the industrial plant, but from an external location, through a remote connection. Such connection may carry text-based data (e.g., for command line operation on a terminal) or multimedia data (e.g., for visual control of machinery).

The technological requirements to establish this kind of remote session are mostly already met in modern industrial environments. However, their activation is often hard to automate, since the OT is generally managed in local networks, and it may include specialized hardware (e.g., an industrial gateway that performs security duties) [HHH22]. Moreover, it is prone to security and privacy threats, arising from the exchange of potentially critical information outside of the private network of the company. Furthermore, in order to guarantee maximum efficacy, the Quality of Service (QoS) perceived by the user (i.e., the technician, in this example) should be high enough to support the workflow without impediments or delays.

The traffic generated by the remotely controlled equipment needs to cross multiple network segments. For instance, the data may be generated on a device connected to a server (e.g. OPC-UA) located on a PLC in the OT segment. From there, it needs to cross the OT network, which is composed of devices such as PLCs, SCADAs, and actuators, which typically have low computation capabilities. Then it goes through the IT network, which usually hosts most of the computation power. Finally, it crosses the Internet, which exposes it to a number of potential threats [AHZ19]. All things considered, the technology implied is not enough to grant confidentiality and integrity of the data, while of course availability depends on the switching and firewall configuration. More specifically, we can analyze the Confidentiality, Integrity, Availability (CIA) risks of each of the traversed domains. Based on computation capability, the crossed path can be divided into two portions, namely OT-IT (low capability route) and IT-Internet (high capability route). In the OT-IT portion, confidentiality is not threatened, as data travels within the private network of the company. Its integrity, however, can be undermined by the fact that legacy devices do not usually support integrity checks. On the other hand, in the IT-Internet portion, data is prone to confidentiality threats, since more personal or company data are shared as well as

integrity issues that may cause data flow disruption or alteration. In both portions, data availability is threatened by out-of-domain connections that can lead to denial-of-service attacks.

Companies often leverage Virtual Private Network (VPN) technology to establish a private end-to-end connection between the client and the IT (which usually comprises a VPN server), ensuring Confidentiality and Integrity of the transmitted data. All things considered, we can still identify two main points that need to be addressed to achieve automated deployment of safe remote maintenance sessions in the industrial plant. The first is the lack of CIA in the OT-IT portion, due to the scarce computation capabilities of the involved devices. The second is the automatic deployment of remote service components to support the maintenance session on client request, keeping CIA requirements into account.

For the purpose of this work, we have defined and implemented instances of such support services, which we refer to as Maintenance Services (MSs), and their goal is to enhance the reliability of the remote maintenance routine, consistent with the use case presented in this section. This is achieved by applying hashing to the data flow, according to one of four different hashing algorithms, namely CRC32, XXH64, MD5, and SHA256. The data flow may be either text-based (e.g., for remote terminal operation) or video-based (e.g., for remote inspection), resulting in a total of eight new services offered to the user.

In Sect. 5.4, we provide a proof of concept to tackle the aforementioned concerns by leveraging service orchestration and PDPaaS through the activation of MSs.

5.4 Proof of Concept: Emulation of an Industrial Environment

This section must be intended as a motivation chapter to introduce the evaluation on Section 5.5. In fact, the testbed used to draw the results shown in this section is emulated. Thus, the results must be interpreted as a simple prototype of our solution, which we believe can be reasonably scaled up to a real-world scenario.

Figure 5.4 shows the emulated industrial environment used for the tests, which is inspired by the architecture described in Sect.5.3.

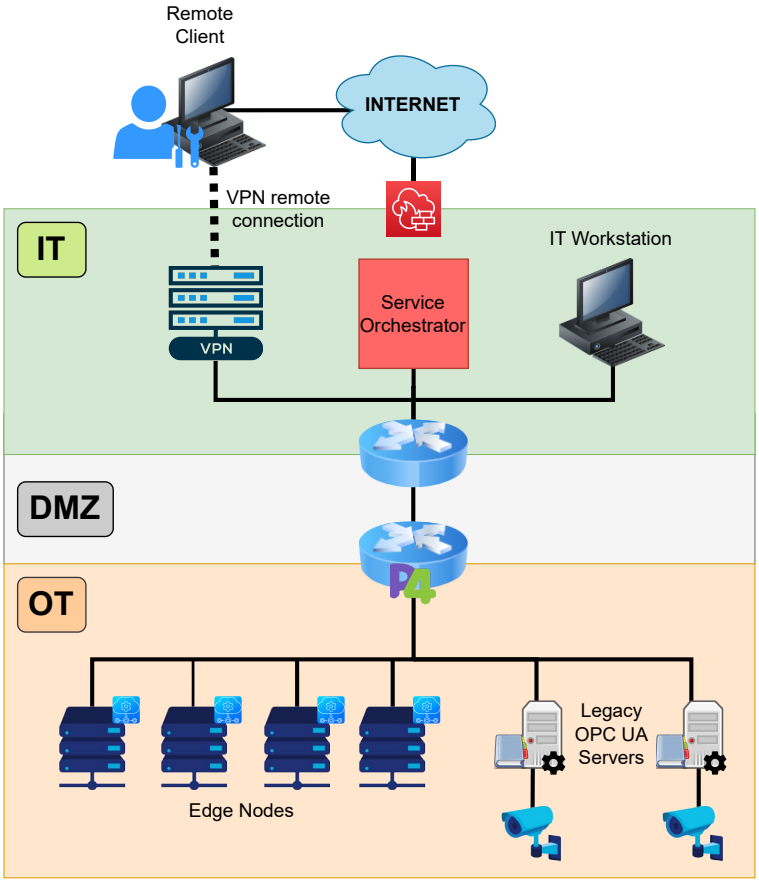


Figure 5.4: Topology of the Proof-of-Concept implementation.

The IT and the OT are separated by a DMZ: on the IT side, it is accessed via a traditional router, while the OT devices are reached through a P4 programmable switch. The P4 switch can be managed by the orchestration framework to provide specific PDPaaS services. Since the goal is to establish a remote maintenance connection, our system provides a remote VPN connection to the industrial network for a client located on the Internet. The VPN server is placed in the IT network, alongside the workstations and the orchestrator presented in Sect.5.2. In particular, the orchestrator is responsible for deploying services both in the IT and the OT. The types of services that can be managed by the orchestrator are also described in Sect.5.2.

When compared to the Purdue reference architecture, the OT network is simplified to only one level, in which two main elements are located, namely OPC UA

servers and Edge Nodes. Edge Nodes supply computational power to the Manufacturing Zone and can support the deployment of PDPaaS services to provide integrity along the OT-IT path. OPC UA servers, instead, are legacy devices with low computing and networking capabilities. Considering this aspect, we assume that they can only operate in Security Mode None. In addition, OPC UA servers are connected to monitoring cameras, as described in [OPC19], which observe the assembly line and allow the remote maintainer to discover faults in the production processes.

5.4.1 Experimental setup

We implemented the emulated industrial testbed using the Kathará framework [Bon+18]. Our Kathará topology and all the test scripts are open-source and public at [RAS23]. All elements of the testbed are implemented as Docker containers, except for a separate node on IT premises hosting the Service Orchestrator. The remote client hosts an OPC UA client process capable of connecting to a VPN server located in the IT network. We built a specific Docker container image published to Docker Hub to implement these functionalities, employing the opcua-asyncio[2] Python library for the OPC UA client functionalities. Edge nodes and IT workstations are general-purpose Debian-based Docker containers; edge nodes have higher computing capabilities, in terms of RAM and CPU power, than the others. OPC UA servers, on the other hand, are equipped with less RAM and less CPU power, to account for the fact that they represent legacy devices, and they are based on the same image as the ones that represent the remote client. The VPN server is an OpenVPN server that pushes the route to reach the OT private subnets to the remote client. Between the OT and the IT segments, there is a traditional Open vSwitch [Pfa+15] which acts as a firewall by means of IPTables rules, with the goal of allowing only VPN traffic. A container emulating the P4 switch is placed between the IT nodes and the Open vSwitch connecting the OT and the IT networks. It is equipped with a simple level 2 forwarding pipeline[3] and configured to only forward traffic to the end hosts: the P4 code is compiled for the reference

[2] `https://github.com/FreeOpcUa/opcua-asyncio`

[3] `https://github.com/UniboSecurityResearch/P4-Forch_KatharaTopo/blob/master/router1/root/p4/program.p4`

virtual target bmv2. On top of this, we developed a P4 program for each payload size and hashing function, i.e., 256 and 1024 bytes, for a total of 8 pipelines (i.e., 2 different payload sizes and 4 hashes). To develop the hash functions, we exploited the idea of P4 extern [Cona]. An extern is an API that uses an external dependency, which can be queried by the target. Each hash function is implemented in the form of

$$hash(output, input[])$$

where the output is the hash produced by hashing the concatenation of the input payload chunks, each chunk being 256 bytes since bmv2 only allows for variables with a size up to 2048 bits. Each hash extern leverages standard C++ implementations of the hashing functions.

Each pipeline hashes the payload in the ingress queue control, by simply calling the extern and adding the hashed payload in the custom field at the start of the payload, as summarized in Alg. 5.

Algorithm 5: Hashing in the ingress pipeline

Input : a packet *packet* containing
H packet header, *hash* payload hash custom field,
P payload;

Output: $hashed_P$ = hashed P

1 $hash \leftarrow hash(hashed_P, P)$: we set the custom hash field in the packet, which is then carried in the network.

We tested the topology on a Ubuntu 20.04 LTS Server with 14GB of RAM and 3 CPU cores KVM machine.

5.4.2 Hashing performance comparison

To evaluate our solution we compared the performance of the programmable nodes and traditional edge nodes in performing the CRC32, XXH64, MD5, and SHA256 hashing functions. To do so, we ran some tests calculating the effectiveness of the two different options to hash OPC UA packet payloads. The results obtained can then be used by the orchestrator when deciding on whether a switch or an edge node should be chosen as a resource to deploy the integrity hashing function on the plant.

As we can see from Fig. 5.5, the payload processing time of the switch increases depending on the type of hash function.

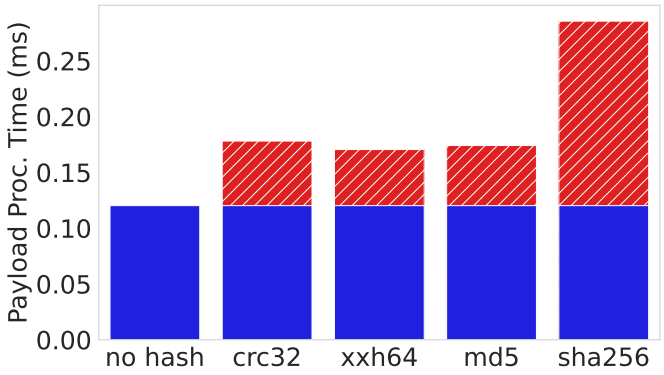


Figure 5.5: Payload Processing Time for 1024 bytes payloads. In blue, is the time to process the OPC UA payload without having it, and in red is the added time to hash it with different functions.

We chose a simple metric to calculate the added hashing overhead, the Total Time Increment (TTI). Given the average processing time to process an OPC UA packet by a switch ($T_{baseline}$) over a number N of analyzed packets with a processing time T_i

$$T_{baseline} = \frac{\sum_{i=1}^{N} T_i}{N}$$

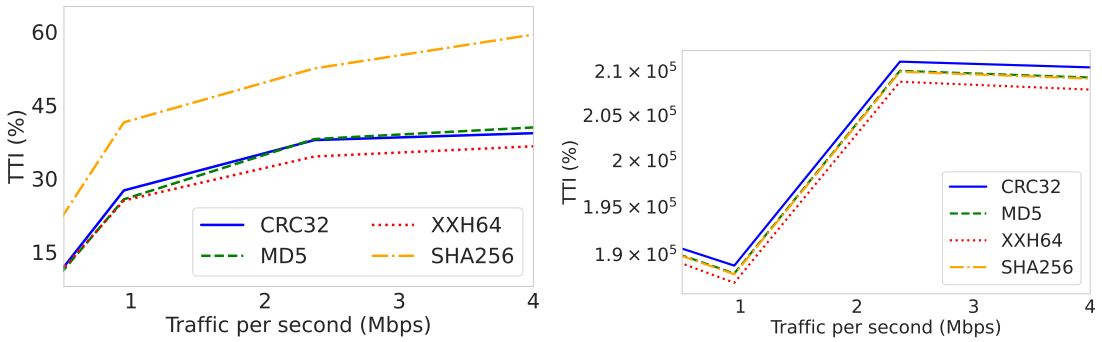
and given the time to hash an OPC UA packet by a switch or an edge node (T_{hash}) over a number N of analyzed packets with a processing time H_i

$$T_{hash} = \frac{\sum_{i=1}^{N} H_i}{N}$$

The Total Time Increment (TTI) is defined as the percentage increase of the hashing time with the baseline, calculated as:

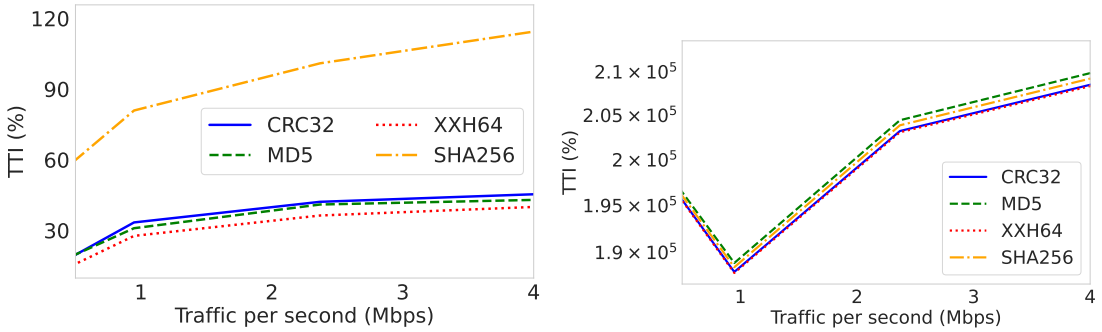
$$TTI(\%) = \frac{T_{hash} - T_{baseline}}{T_{baseline}} \cdot 100$$

We calculated the TTI for three different payload sizes: 256 bytes (Fig. 5.6) and 1024 bytes (Fig. 5.7), for different traffic rates. We gathered data for traffic



(a) *switch* (b) *edgenode*

Figure 5.6: Total Time Increment values for 256 *bytes payload* packets measured on a switch (a) or an edge node (b), for each hashing function (CRC32, XXH64, MD5, SHA256) with variable traffic volume.



(a) *switch* (b) *edgenode*

Figure 5.7: Total Time Increment values for 1024 *bytes payload* packets measured on a switch (a) or an edge node (b), for each hashing function (CRC32, XXH64, MD5, SHA256) with variable traffic volume.

lower than 4 Mbps since in our configuration the topology has packet loss for traffic above that threshold. These limitations come from the use of a virtualized environment and bmv2.

Figures 5.6 and 5.7 show the TTI for OPC UA payloads of 256 bytes, comparing switch and edge node hashing to the baseline. By looking at the figures, we can summarize two main outcomes:

1. Computing hashes on the edge nodes entails a TTI that is 4 orders of magnitude bigger than hashing on the switch.

2. The TTI of the switch varies depending on the hashing function.

Statement 1. is explained by the fact that the processing time of the edge node is calculated between the ingress and egress interfaces of the container. In fact, this time is the composition of the time to move the packet sniffed with the Scapy[4] library from kernel to user space, the time to hash the payload with a Python script, and the time to move back the packet from user to kernel space. Moreover, the node TTI depends more on the traffic rate than the hashing function.

The switch TTI results roughly confirm the expected behavior in terms of hashing complexity [RAD15]. In fact, the SHA256 hash calculation takes more steps than CRC32, XXH64, and MD5.

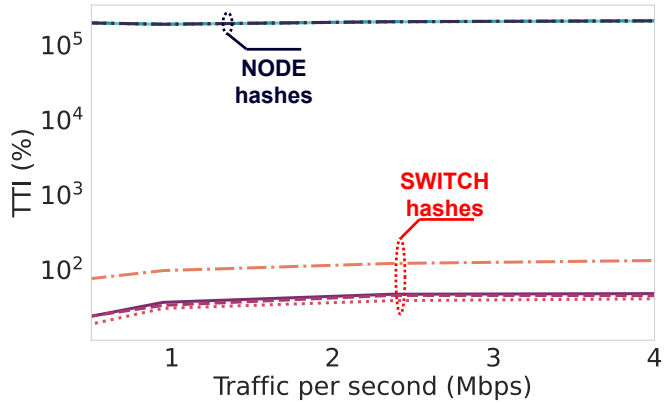


Figure 5.8: TTI resulting from performing the hashing on the switch or on the edge node.

Fig. 5.8 groups the TTI curves for 1024 bytes payload, using a logarithmic TTI scale. Each set of curves is graphically grouped with a colored circle, based on the type of device that performs the hash. The logarithmic scale outlines the difference between the node TTIs and switch TTIs.

As an overall comparison, Fig. 5.9 sums up how the average processing time for a packet on the switch completely outperforms the respective one on the edge node. This suggests that, if choosing between an edge node and a switch for service placement purposes, the obvious choice is the programmable switch.

[4] `https://scapy.net/`

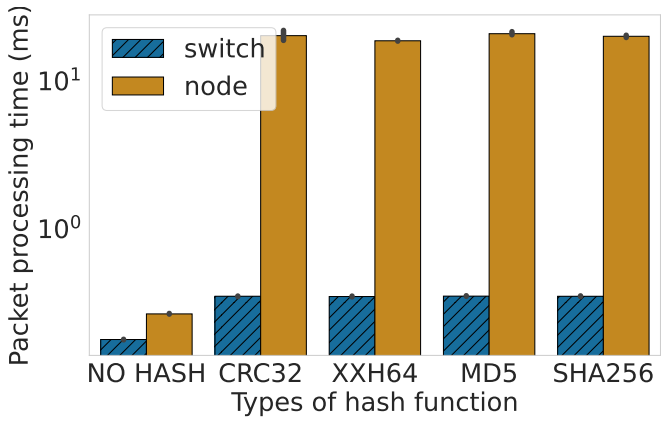


Figure 5.9: Edge node and switch average delay comparison for different hashing functions.

5.5 Evaluation Process and Results

As implied by the results presented in Sect. 5.4.2, offloading the computation required for MSs to PDP devices allows to achieve better performance compared to using conventional computing devices for the same task, while also requiring no replacement of legacy devices that are already deployed in the scenario. In this section, we evaluate the feasibility and the performance of the orchestration of those services in an environment such as the one described in Sect. 5.4. To do so, we implemented a Python-based discrete event simulator.

In the simulator, the orchestration system is actually reduced to its *Service intelligence* element, as no real telemetry data is collected from – and no management action is applied to – underlying resources. The intelligence of the simulated orchestrator is in charge of making placement decisions while enforcing a given policy, and, as the simulation progresses, the availability of resources is affected by the decisions made up to that point. For performance evaluation purposes, we implemented three policies, so as to have the orchestrator prioritize different aspects while making its decisions.

The first policy is denoted as *Random* (R), and instructs the orchestrator to make a completely random decision on the action to apply to any service request, including the choice of where and how to activate the service or block the service activation altogether. Such actions are uniformly distributed and independent of

one another, meaning that each service request might be served by a computing resource, a networking resource (if available), or be blocked, with the same probability. This placement algorithm is represented in Alg. 6.

Algorithm 6: Service placement with *Random* policy

Input : set R^c of computing resources
$R^c = \{r_i^c \mid i \in \mathbb{N} \wedge i \leq \text{amount of computing resources}\}$; set R^n of networking resources,
$R^n = \left\{r_j^n \mid j \in \mathbb{N} \wedge j \leq \text{amount of networking resources}\right\}$;
combined set R of available resources
$R = \{r \in R^c \cup R^n \mid r \text{ is not completely busy}\}$

Output: service placement decision p

1 $p \leftarrow$ *choice*$(\{r \in R\} \cup \{\text{block}\})$, where *choice* represents a random selection of an element from the input set, with uniform probability

The second policy is called *Load balancing* (LB), according to which the orchestrator is expected to balance the service placement between computing and networking resources, with a particular focus on network utilization. In other words, when enforcing this policy, the orchestrator tries to balance the overall load due to service placement across the resources. This favors the choice of a networking resource over a computing one when the network load grows, leveraging the better ability of networking resources in providing specific services. As detailed in Alg. 7, when applying this policy, the orchestrator sorts the available resources based on the collected metrics (CPU, RAM, disk, network usage, etc.), from the least to the most loaded. It then picks the resource with the lowest network occupation between the first computing resource and the first networking resource in the sorted list.

The third and final policy considered is referred to as *User QoS* (UQoS) because it aims at optimizing all aspects perceived by users, including the service activation time and overall latency, as discussed in 5.4. The rationale behind this is in the consideration that activating a service on a PDP device with a pre-configured pipeline is much faster than doing it on a computing resource, as will be further argued later on in this section. Additionally, the performance in terms of service fruition of PDP devices have been shown to be superior to that of computing resources. For these reasons, if the objective is to enhance the overall user experi-

Algorithm 7: Service placement with *Load Balancing* policy

Input : set R of available resources r as defined in Alg. 6; current status s of each resource as a collection of metrics, $s(r) = \{\text{CPU, RAM, etc.}\}, \forall r \in R$;
Output: service placement decision p

```
if R = ∅ then
    p ← block
    return
if |R| = 1 then
    p ← the only available resource
    return
Create the ordered set R_O by sorting elements of R by their metrics,
  starting from the least busy one
if no networking resource in R_O then
    p ← the first element of R_O
    return
if network load of least busy networking resource < network load of least
  busy computing resource then
    p ← the least busy networking resource
else
    p ← the least busy computing resource
```

ence as much as possible, networking resources should be favored over computing ones at all times. In line with what is represented in Alg. 8, when activating a service, the orchestrator will always select the P4 switch if it is available, or the least occupied edge node otherwise, blocking the request if none of these resources is available.

In each simulation run, the orchestrator receives a sequence of service requests, which may require any of the supported services to be activated, including the newly-implemented MSs, that may leverage DPP. The service requests are generated following a Poisson process, with the service duration being exponentially distributed. Specifically, service requests are randomly generated out of a pool of 50 different services, of which 8 are MSs and 42 belong to the other service models with the following proportion: 10% IaaS, 20% PaaS, 30% SaaS, and 40% FaaS. Each request may pose different requirements in terms of computing power, mem-

Algorithm 8: Service placement with *User QoS* policy

Input : set of available resources R as defined in Alg. 6; current status s of each resource as defined in Alg. 7
Output: service placement decision p

```
if R = ∅ then
    p ← block
    return
if |R| = 1 then
    p ← the only available resource
    return
Create the ordered set R_O by sorting elements of R by their metrics,
  starting from the least busy one
if ∃ networking resource ∈ R then
    p ← the least busy networking resource
    return
p ← the least busy resource
```

ory, storage, and network capabilities, as well as in terms of service provisioning mechanisms (referred to the XaaS paradigm detailed in Sect. 5.2). Based on the policy to be enforced, the details specified in the request, and the current status of the underlying resources, the orchestrator needs to make a decision on how and on which resource to activate the service, or to block the request in case no suitable resource is available. In the simulation, no actual telemetry data is collected, so all resources are considered to be fully available at the beginning of each run, while at each simulation step they are assigned to an incoming service request, and released at its completion.

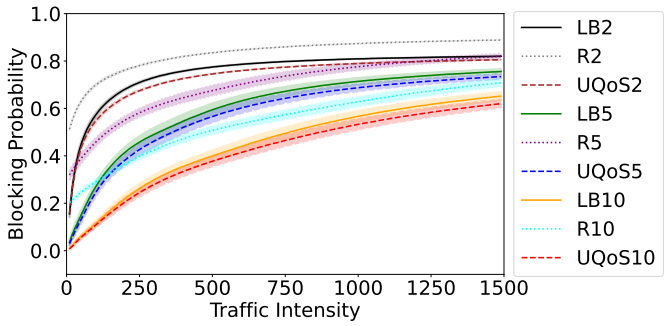
We estimate the service activation delay, defined as the additional time required by the orchestrator to activate the service (not including network time), by counting the amount of times that a MS was activated according to each of the different models supported, and multiply that amount by the average time required to activate a service in that way, as evaluated in [Pit+22].

The simulation scenario stems from the topology depicted in Fig. 5.4, instantiated in the simulator in three different configurations, with either 1, 4, or 9 compute nodes and a P4 switch available, for a total of 2, 5, or 10 available items

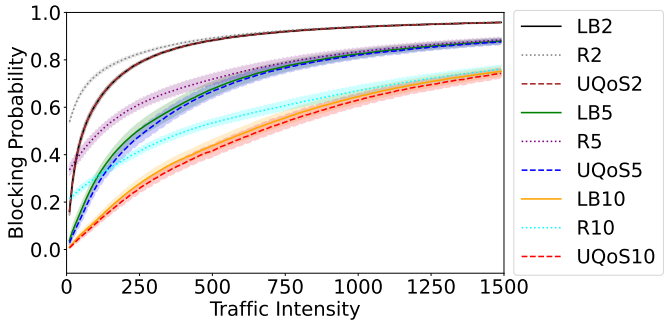
in the resource pool the orchestrator can choose from. Industrial environments are very diverse, in that the amount and distribution of computing and networking resources can vary widely among different premises. In such environments, we expect to have a limited amount of computing resources in the IT and at most one programmable switch (along with potentially multiple non-programmable ones) between OT and IT. We argue that the different scenarios we showcased represent real-world industrial ones, both from the point of view of the technological solutions involved (i.e., P4) [CDT21], as well as from that of the infrastructural network models [Don+24]. Based on the same topology, we further distinguish two cases, differentiated by whether the P4 switch is available or not – in other words, whether the orchestrator can rely on heterogeneous (computing and networking) resources or not.

We simulated each different scenario given by the combination of configuration of nodes, switch availability, policy, and traffic intensity. In particular, available nodes were allowed to vary freely between simulations, but at least one of them was required to support the IaaS model. Moreover, we let the traffic intensity (the ratio between arrival rate and service rate) vary from 10 to 1500, in steps of 10. For each scenario, we run the simulation 20 times, with different seeds, to obtain different incoming service requests at each run. We generated and submitted 10^5 service requests for each run. We then computed the sample mean over the 20 runs and reported them in the figures as the solid and dashed lines, surrounded by shaded areas depicting the 95% confidence interval.

To begin with, we assessed the probability of a service request being blocked by the orchestration system. This was computed as the ratio between the number of blocked requests and the number of offered ones. As shown in Fig. 5.10, the results show an Erlang-B formula behavior, as can be expected, given the distributions of the service request arrival rate (Poisson) and of the service duration (exponential), as well as the fact that the orchestrator behaves like a multiple-server system without queuing space. We can also confirm the intuitive expectation that the blocking probability is higher for scenarios with a lower amount of available resources, and for those using the *Random* service placement policy. The difference between Fig. 5.10a and Fig. 5.10b is in the availability of the P4 switch. In the former case, the switch can be leveraged to provision services. In the latter case,



(a) PDP device available



(b) PDP device unavailable

Figure 5.10: Probability of the orchestration system in blocking a generic service request due to insufficient available resources. The curves represent the performance of different service placement policies with different amounts of available resources, as indicated by acronyms and numbers in the legend.

the switch is assumed pre-loaded with other tasks, making it unavailable for the orchestrator. The policy that is most influenced by this difference is the *User QoS*, as it prioritizes the use of the PDP device. Also the *Load Balancing* policy shows some sensitivity to the availability of networking resources for a very small amount of overall resources, as in that case the policy can no longer reduce the load on the compute nodes.

As a further evaluation, we assessed the impact of the activation of MSs on the workflow of the user. In other words, we wanted to evaluate the delay caused by the additional deployment time of the service components needed to apply hashing to the text or video stream that the user employs to perform remote operations and compare the impact of different service policies. We defined a metric, called Mean Activation Time (MAT), that is computed at the end of the simulation of

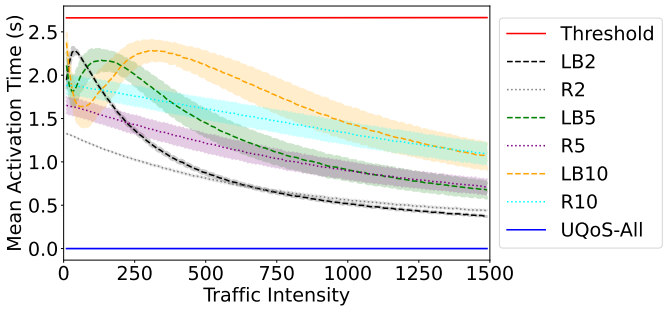
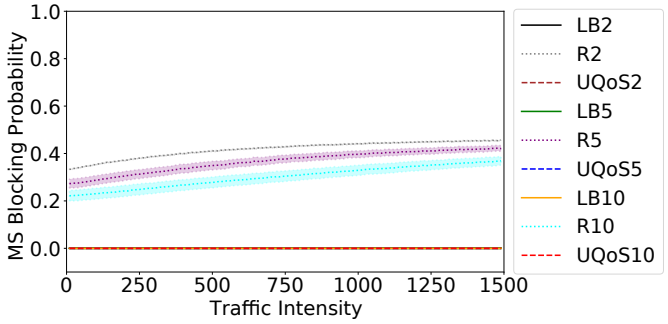


Figure 5.11: Mean Activation Time of Management Services for different service placement policies and amounts of available resources.

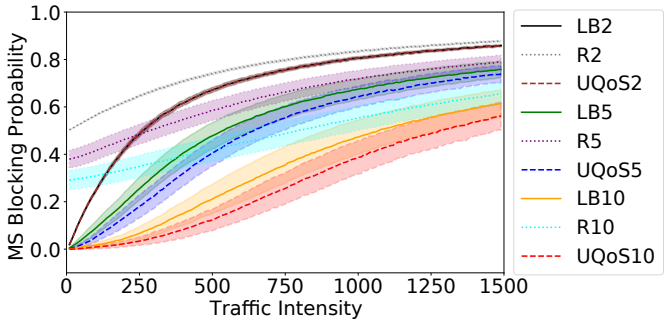
each scenario, as the ratio between the total time taken by the activation of MS instances over the number of such instances activated in the simulation run, where all kinds of service requests are generated. In practical terms, if MAT is zero, the activation of services by the orchestrator did not introduce any tangible delay in the remote operations of the user. For instance, this is possible when the MS is deployed on a P4 switch pre-loaded with the code needed to perform the service, requiring only a signal (e.g., a REST HTTP request) to activate the additional traffic processing, without the need for a reconfiguration of the pipeline. To be precise, this would still inevitably require a small time, but we can reasonably approximate it to zero, as it would be seamless to the workflow of the user. Any value of MAT larger than zero means that the user is subjected to a small delay while the hashing mechanisms are activated.

This MAT is represented in Fig. 5.11, focusing on the case when a PDP device is available. Such switch is consistently leveraged by the policy *User QoS* for the deployment of services, allowing that policy to achieve a negligible MAT. The MAT related to the other policies depends on the traffic intensity. In particular, the decrement of the MAT with larger values of traffic intensity is due to the decreasing availability of edge nodes, forcing the two policies to choose between the PDP device or blocking the request. To fully understand this result, one must consider that the activation of services on compute nodes takes longer for the first MS instances deployed on them, due to the need to download the required software (e.g., the Docker image) and set it running. The activation time decreases as further requests for the same services are served, as the software components

will already be on the node. When considering small values of traffic intensity, the impact of the first requests (i.e., the ones taking longer to be activated) will still be relevant with respect to the following ones (i.e., those served in a shorter time), leading to a higher value of MAT. In line with this rationale, one would expect all curves to exhibit a maximum for the lowest value of traffic intensity.



(a) PDP device available



(b) PDP device unavailable

Figure 5.12: Probability of the orchestration system in blocking a request for a Management Service due to insufficient available resources. The curves represent the performance of different service placement policies with different amount of available resources, as indicated by acronyms and numbers in the legend.

However, that is not so for the *Load balancing* case, as this policy places services on the resource that is least involved in handling network traffic, generally favoring compute nodes in an initial phase, until the point where it can no longer keep up with the increasing traffic intensity as resources are saturated, resulting in choosing the PDP device. The small number of requested MSs in comparison to other generic services (on average 8 out of 50) are typically served by the PDP device. On the other hand, the *Random* policy exhibits a trend compliant with its working

principle of randomly selecting a service placement action, and such actions are restricted to activating the service on the PDP device or blocking the request. The horizontal line denoted as "Threshold" represents the mean of the MAT value across all the policies when the PDP device is completely busy, thus unavailable to the service placement process. The fact that this line is never crossed is due to the availability of the PDP device, showing that, when the nature of the services allows it, leveraging DPP always leads to better performance compared to using compute nodes alone.

As a last evaluation, we assessed the blocking probability again, but limiting the requested services only to those that can be deployed on the programmable switch, i.e., to MSs. In Fig. 5.12a we can see that if the switch is available, the only policy experiencing blocking is the *Random* one, as the other policies always pick the switch to provide the service. This is the reason for the apparent better performance of the *Random* policy for low values of traffic intensity in Fig. 5.11: the policy is actually blocking numerous requests. Conversely, when the switch is not available, the situation is comparable to when the switch is available but overloaded, making Fig. 5.12b resemble Fig. 5.10a.

5.6 Discussion

In this chapter, we demonstrated how DPP can be employed to enhance service orchestration over a heterogeneous infrastructure, consisting of a diversified set of resources. We focused on an industrial environment, which is particularly prone to security threats, and suggested solutions to mitigate the issue. We considered a number of different methods to improve data integrity, implementing services employing them, and making them available to a service orchestration system that is able to activate them at need over heterogeneous resources. The evaluation consisted of two parts, the former focused on the data integrity mechanisms, and the latter on the service orchestration solution.

We demonstrated how the use of PDP devices can be largely beneficial to the execution of data integrity mechanisms compared to regular container-based services on the edge nodes. In fact, the programmable switches completely outperform the edge nodes in terms of processing time since the integrity calculation is placed

inside the network nodes and performed on the flowing traffic. We evaluated the orchestration solution with the use case of remote maintenance in mind, addressing various technical, policy, and performance aspects. We focused on the offloading of computation for MS to PDP devices, showing that this approach offers superior performance compared to conventional computing devices without requiring the replacement of legacy equipment. We considered multiple distinct service placement policies that allow the orchestration systems to make intelligent decisions in line with their objectives and resource availability. We aimed at replicating real-world conditions, where MS orchestration must handle a dynamic sequence of service requests with varying requirements. We measured the likelihood of service requests being blocked due to resource constraints, as well as the delay introduced by MS activation, showing that the presence and availability of PDP devices may have a significant impact on the achieved user QoS.

In conclusion, this research contributes to a deeper understanding of how to optimize the orchestration of management services in industrial settings. By addressing technical, policy-driven, and resource-specific aspects, our study provides guidance for achieving better resource utilization, reduced service activation delays, and improved user experiences.

Chapter 6

Switch-to-switch IIoT tunneling*

In this chapter, we are going to present how DPP can be used to implement simple switch-to-switch tunnelling in low-computation IIoT networks. Communication between assets and systems is one of the key concepts behind the Industry 4.0(I4.0) paradigm. It exploits two logical network infrastructures: the legacy Information Technology (IT) network that connects the many distributed applications referring to the management and logical operations of the company, and the Operations Technology (OT) network [Ber+23]. Generally speaking, the OT is the set of all the hardware and software components that detect or cause changes in the production or operational processes of a manufacturing plant [Gar22].

The OT network was traditionally made up of many separate sections, serving specific machines or production lines, implementing the communication of the so-called Industrial Control Systems [CDT21]. ICSs are equipped with sensors and actuators and are usually controlled by Programmable Logic Controllers (PLCs). ICSs were traditionally physically isolated from IT. This has changed since machines, devices, and production lines compliant with the I4.0 paradigm are capable of collecting and communicating with each other, to achieve highly scalable levels of coordination and collaboration [Gho18].

The opening of OT to IT and the Internet exposes it novel cybersecurity threats

*Part of the material presented in this chapter is based on the following publications: Lorenzo Rinieri et al. "In-Network Encryption for Secure Industrial Control Systems Communications". In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE. 2024, pp. 190–194

to the manufacturing environment (with several examples that have made headlines in newspapers [HF+18]), making IIoT security a top priority concern for ICSs. Securing communication between devices through encryption should be a basic requirement, especially for networks exposed to the Internet. However, many ICS protocols do not implement any encryption, exposing OT devices to textbook vulnerabilities. One of the reasons why this is common practice is that the OT devices are designed to prioritize efficiency and speed, making encryption just unneeded overhead. Furthermore, operational continuity is always preferred to functional updates: hence, it is common practice to keep legacy software running on the OT rather than performing regular updates. On a side note, edge OT devices are usually simple gateways and data collectors that are unlikely to perform any packet or flow inspection. Consequently, it is difficult for the developer to enforce any flow-based encryption to protect vulnerable traffic. In this chapter we argue that we can address these issues by showing that Software Defined Network (SDN) based *in-network computation* can be used to *provide secure communication between the OT and the IT*, without modifying the end devices. The end user is not affected by this solution, as it is used to filter out any possible communication-related device-to-device threats. The solution presented exploits *data plane programmability* and the P4 language. P4 lets the network programmer describe how the switch should process the packets, thus adding functionalities to the network, by exposing the packet processing logic to the control plane to enable a systematic, fast, and complete reconfiguration. This allows flexibility, intended as the capacity of changing functions or services at will [Sad+23; Pit+24]. The chapter is structured as follows. In Section 6.1 we describe the use case we considered to test this work. In Section 6.2 we present the experimental test bed that was implemented to experiment and validate the proposed solution and then in Section 6.3 we present performance numerical results. In Section 6.4 we discuss our results.

6.1 Use case scenario

We consider a scenario where several OT components must communicate but do not support encryption. In their stead encryption will be managed, when needed, directly by the network nodes using in-network processing.

The considered OT architecture is a manufacturing plant where working stations perform some complex task, integrating several components and machines. These working stations coordinate by exchanging data over the OT network, as well as talking with the management processes in the IT. We assume that *the network section within the working station is secure*, since it is isolated. The attacker can harm the system exploiting *the interconnection part of the OT as well as in the interworking between the OT and the IT* [EMT22]. For this reason, communication between these network sections should be secured via encryption.

This use case considers Modbus, a stable and well-known ICS communication protocol. Open, standard, and widely, it allows broad monitoring and management of individual PLCs within an ICS. It accounts for about 10% of the new installations [HMS24]. Nonetheless, we argue this solution can be seamlessly applied to multiple protocols.

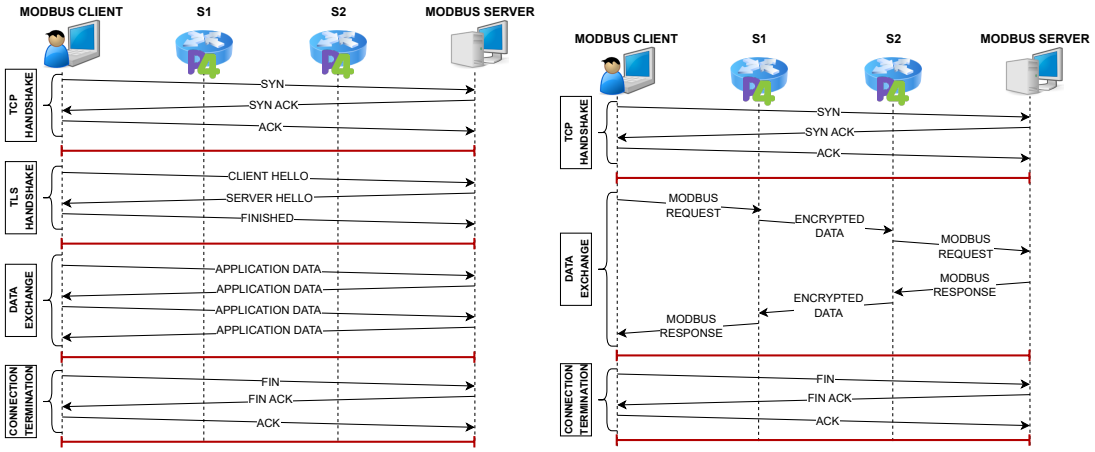
Modbus can be simply emulated in virtual test beds with limited effort while still maintaining its full functionalities. In this use case we consider plaintext Modbus, which is what can be found in most deployed of the OT plants nowadays. However, a standard implementation of Modbus supporting TLS for end-to-end encryption was released in 2018 [Modb], which we will use as a baseline offers to compare our solution. Fig. 6.1a shows the typical call flow of a Modbus-TLS connection [Modb]. The figure shows that the TCP three-way handshake is followed by the TLS handshake and by the encrypted application data exchange. Here encryption happens end-to-end from the source to the destination. Fig. 6.1b shows the call flow of our solution. We assume that the end nodes do not support encryption, wither because they do not possess enough computation capabilities or because they have not been updated to avoid plant downtime. For this reason, we place P4-enabled programmable switches in the network border areas, which we consider safe.

The switches are responsible to:

- Analyze the crossing traffic, with a custom processing P4 pipeline - we will exhaustively explain the P4 logic later in the chapter.
- Identify the traffic that may be considered at risk, based on the destination or content of the packet.

- Start a switch-to-switch encrypted tunnel whenever traffic at risk is found between two nodes.

It should be noted that end users are completely unaffected by our solution.



(a) Sequence Diagram of Modbus TCP security over the TLS protocol.

(b) Sequence diagram of our in-network encryption approach.

Figure 6.1: Comparison of the sequence flow of Modbus security and our proposed approach.

6.2 Testbed set-up

The operational principle described above was implemented in a virtual environment, as shown in Figure 6.2. We exploited the Kathará framework [Bon+18], in which all elements are emulated by Docker containers. We built specific Docker container images published to Docker Hub to implement the Modbus client and server functionalities, employing the PyModbus [Pyt] Python library. For comparison purposes, the Modbus client and server were developed to support both traditional Modbus communication without any security guarantees and secure Modbus communication over TLS.

The Modbus Client and Server are connected by P4 switches implemented by means of two Docker containers. They are equipped with a simple level 2 forwarding pipeline and configured to only forward traffic to the end hosts: the P4

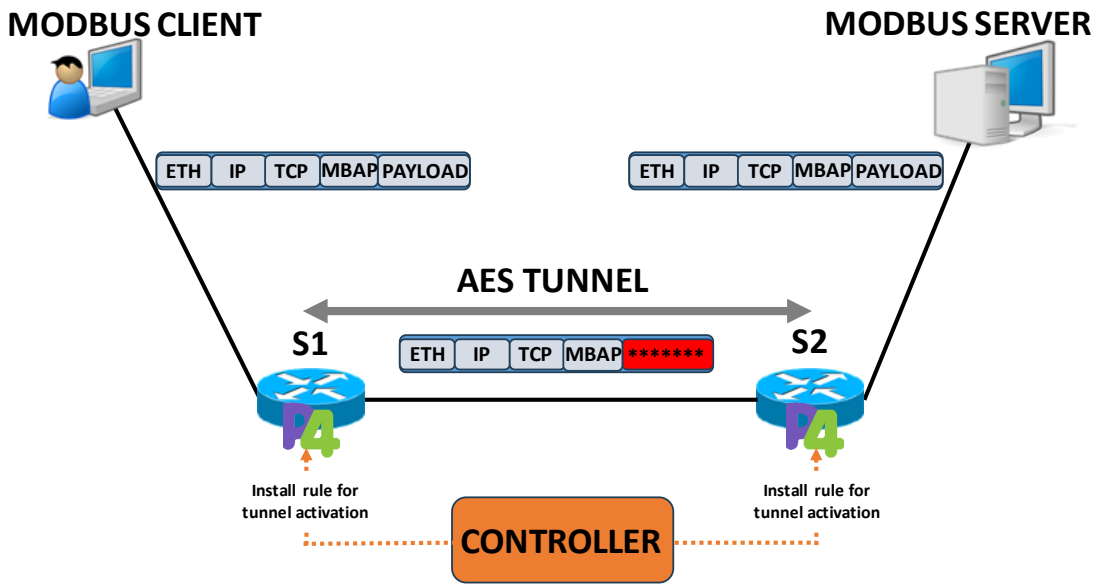


Figure 6.2: Virtualized testbed implemented for the use case scenario.

code is compiled for the reference virtual target bmv2. A custom implementation of the control plane is also implemented, which manages network monitoring and configuration. We use the *P4Runtime* APIs [Con20], the standard Southbound API protocol to interact with P4 devices, to reconfigure the P4 pipeline at runtime.

We developed a specific P4 pipeline to handle the Modbus payload [Moda], with variable length. The steps performed are as follows:

1. the parser extracts the Ethernet and the IP headers from each packet;

2. if the *Protocol* field of the IPv4 header is equal to six (i.e. the TCP protocol), then the TCP header is extracted;

3. knowing the field *Total Length* of the IPv4 header field and the field *Data Offset* of the TCP header, the P4 parser calculates the actual length of the TCP header, including the TCP options, and the total payload size;

4. if the source and destination ports are equal to 502 (i.e. the default port assigned by IANA to the Modbus protocol), the Modbus Application (MBAP) header is parsed;

5. finally, the Modbus payload is extracted depending on the value of the *Length* field in the MBAP header.

The described parsing allows the switch to recognize the source and destination endpoints that require encryption. With this information and with the parsed Modbus payload, the switch is able to secure the data exchanged between those specific sources and destinations.

We chose and implemented 128-bit key AES to secure the channel. In the proposed use case, symmetric AES keys can be pre-installed in the P4 switches at configuration time or can be exchanged between the controllers via Diffie-Hellman key agreement [Gil16] at runtime and then installed on the switches. Practically speaking, the secure tunnel between the two bmv2 switches can be activated:

- once and for all at system start-up, in such a way that all communication requiring encryption will be conveyed through that tunnel;
- ad hoc for any specific traffic flow depending on the communication endpoint, so that a new encrypted traffic channel is established of each specific flow.

The controller is configured by the network administration to install the suitable pipelines at any given moment. If they choose to configure it to support ad hoc tunnels, the controller will install a pipeline that will automatically activate a new encrypted tunnel for the specific flow, with no need for high-level actions.

To develop the AES encryption and decryption functions, we use a P4 extern [Cona]. An *extern* is an API that uses an external dependency, which can be queried by the target. In this case, the implemented extern leverages standard C++ implementations of the AES 128-bit encryption and decryption functions. Each switch performs both encryption and decryption, depending on the flow direction. As depicted in Fig. 6.2, only the Modbus payload is encrypted inside the secure tunnel: considering that the maximum payload size of Modbus TCP is 253 bytes [Moda] each encrypted chunk will have a dimension of 128 or 256 bytes, given the AES key size. This is an important aspect to highlight, since we can fit our solution inside the boundaries of bmv2, which allows for variables with a size of up to 2048 bits.

Each pipeline encrypts and decrypts the payload in the ingress queue control, by simply calling the extern and emitting the encrypted/decrypted payload in the Deparser pipeline, as summarized in Alg. 9 and Alg. 10.

Algorithm 9: Encrypt action in the bmv2 Ingress Pipeline

Input : The Modbus payload p of the input packet.
Output: The encrypted data *enc* correspondent to the Modbus payload p.

1 $enc \leftarrow Encrypt(p)$: call P4 extern AES Encrypt function;
2 Update the value of the IPv4 header field $Total\ Length$;
3 Update the value of the Modbus header field $Length$;
4 $p.setInvalid()$: do not emit the original payload in the Deparser;
5 $enc.setValid()$: emit the encrypted payload in the Deparser.

Algorithm 10: Decrypt action in the bmv2 Ingress Pipeline

Input : The encrypted Modbus data *enc* of the input packet.
Output: The decrypted Modbus payload p.

1 $p \leftarrow Decrypt(enc)$: call P4 extern AES Decrypt function;
2 Update the value of the IPv4 header field $Total\ Length$;
3 Update the value of the Modbus header field $Length$;
4 $enc.setInvalid()$: do not emit the encrypted data in the Deparser;
5 $p.setValid()$: emit the decrypted payload in the Deparser.

6.3 Results

We argue that the proposed approach may achieve realistic performance for the use case of securing ICS communications while complying with industrial real-time constraints. In addition, the proposed solution allows to secure critical network segments even when low computing power and legacy devices are at one end of the communication.

We performed three types of tests to prove the effectiveness of this approach, as described in the following. All the tests were run in the virtualized industrial testbed described in Section 6.2 exploiting Kathará version 3.7.1 and Docker engine version 25.0.0, hosted in a Ubuntu 22.04 LTS PC (Linux kernel version 5.15.0-

92.102) with 32GB of dual-channel RAM and an i7-1265U 12th generation Intel processor.

6.3.1 Overall Performance

To evaluate the overall performance of the proposed in-network encryption solution, first of all, we designed a test considering two Modbus operations:

1. *Read Input Registers*, where we read the first register of the Modbus server;
2. *Write Single Register*, where we write a random value in the first register of the Modbus server.

We performed the aforementioned operations in three different scenarios:

1. traditional Modbus communication without any security mechanism;
2. In-Network encryption via in-network computation;
3. Modbus security over the TLS protocol, exploiting the Modbus TLS extension.

For each considered scenario, we repeated each operation 100000 times, recording the time needed to complete it. The average times expressed in milliseconds with their corresponding standard deviations are reported in Fig. 6.3, grouped by type of operation (i.e. read or write). As we can see, the performance of this solution is very close to the traditional Modbus communication and better than the Modbus TLS.

6.3.2 Encryption and Decryption Overhead

We estimated the encryption and decryption overhead by calculating the average Packet Processing Time and the average Packet Dequeuing Time inside the two switches for Modbus *Read Input Register* and *Write Single Register*. The results are the average of 100000 samples and are reported respectively in Fig. 6.4 and in Fig. 6.5. Taking into account that each switch processes one Modbus request and one Modbus response for every *Read Input Register* and *Write Single Register* operation, we consider the encryption and decryption costs as acceptable.

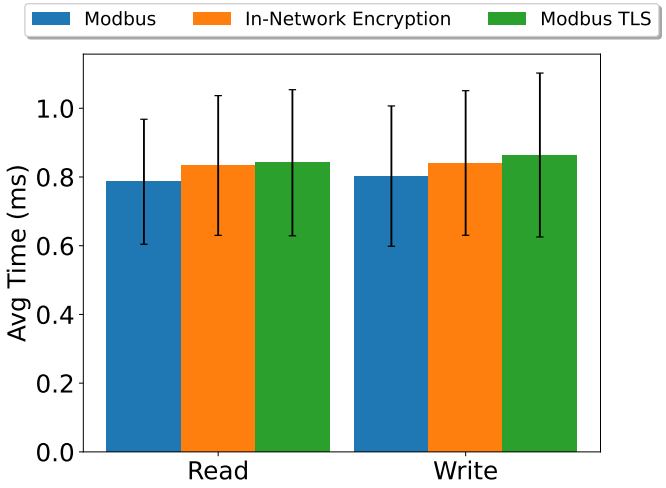


Figure 6.3: Comparison of the time needed to complete *Read Input Register* and *Write Single Register* Modbus operations, in the three scenarios: Modbus, In-Network encryption, and Modbus TLS. The reported values are the average of 100000 samples, and the error bars report the standard deviation of the measure.

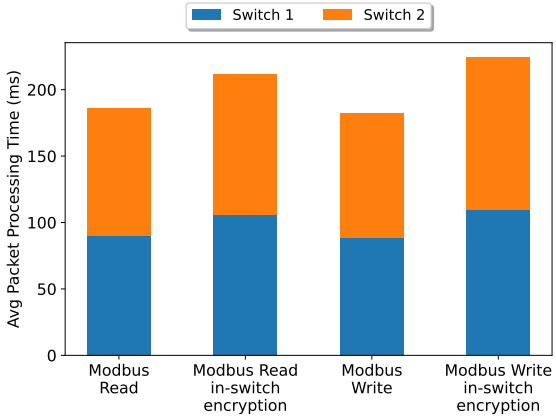


Figure 6.4: Average Packet Processing Time of *Read Input Register* and *Write Single Register* Modbus operations, calculated inside the two P4 switches, with and without the encrypted tunnel.

6.3.3 Key Exchange Overhead

The last test was designed to assess the time needed to establish the encrypted tunnel: we compared it with Modbus and Modbus TLS connection establishment times. The average results with their corresponding standard deviations are outlined in Fig. 6.6, computed over 10000 connection attempts.

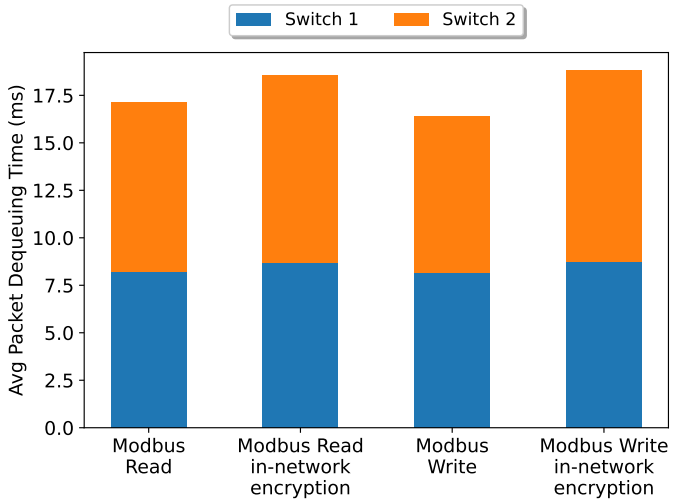


Figure 6.5: Average Dequeuing Time of *Read Input Register* and *Write Single Register* Modbus operations, calculated inside the two P4 switches, with and without the encrypted tunnel.

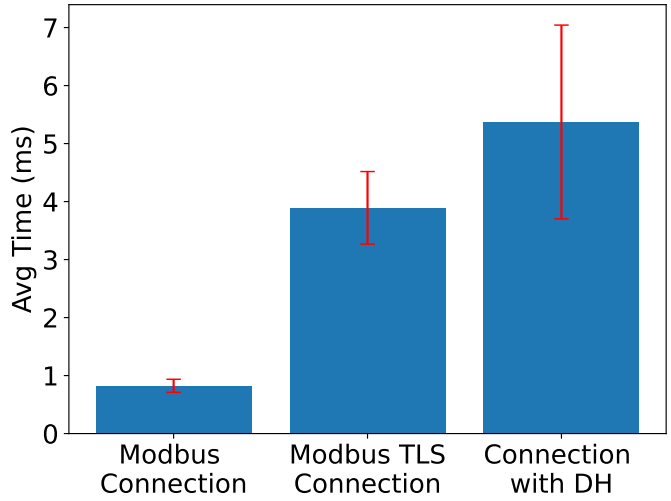


Figure 6.6: Connection establishment average time comparison.

In particular, the three bars depicted represent:

1. Modbus Connection time, i.e. the time to complete the TCP three-way handshake.

2. Modbus TLS Connection establishment time comprises the TCP and the TLS handshake times.

3. Connection with Diffie-Hellman (DH) encompasses **(1)** the time to perform the Diffie-Hellman key exchange for the negotiation of the symmetric key between two controllers, **(2)** the time to install the symmetric key into the P4 switches, and **(3)** the TCP handshake time.

Most notably, the average connection with the DH exchange and the TLS connection establishment times are within the same order of magnitude. Deploying the encrypted tunnel after the execution of DH permits achieving a higher level of security by avoiding the reuse of symmetric keys [Ela+16].

However, it is important to highlight that if the symmetric keys are pre-installed in the P4 switches the encrypted tunnel establishment time is reduced to the Modbus Connection time.

6.4 Discussion

In this chapter, we demonstrated that it is possible to exploit P4 programmable switches to provide secure communication in Industrial Control Systems while respecting the real-time constraints of these environments. As a use case scenario, we considered a widespread OT protocol, Modbus, and its extended version which enables confidentiality of the transmitted data through the TLS protocol. The results show that our solution offers performance better than Modbus TLS and not significantly worse than unencrypted Modbus. Sacrificing end-to-end encryption increases the security posture of vulnerable devices, without OT downtime or unwanted configurations.

Part V

Conclusions and bibliography

Challenges and open problems

In this section, I will present some challenges and open problems stemming from the work carried out during my thesis. The points raised here are limited to the scope of the thesis, as an exhaustive analysis of these topics would require a much broader and more detailed exploration.

Increasing anomaly detection precision: Estimating traffic anomalies with precision in a programmable data plane, particularly in the context of the P4 language, presents significant challenges. One issue is the problem of thresholding, where dynamically adapting thresholds based on historical traffic patterns can help refine anomaly detection. But how do we set the right threshold? Many effective ways of updating the threshold can be found in literature [Hyn11; BM61], and can be simplified and implemented in the data plane. However, most of the promising solutions in literature [DSS21; Yan24] require control plane support to calibrate and update the threshold depending on the overall trend of the traffic. On the same note, probabilistic data structures such as count-min sketches [CM05], often used for efficient traffic estimation, introduce another layer of complexity. These structures, while compact, are prone to collisions, leading to false positives and false negatives that undermine the accuracy of anomaly detection. Indeed, using a local control plane to update the switching rules, mitigates most of these limitations. However, specific problems [Zha+20; Liu+21] require updating the pipeline logic at runtime the required detection logic. This is indeed a powerful feature, but requires recompilation and redeployment of the program, which momentarily halts packet forwarding. This pause, albeit brief, is particularly disruptive in systems with stringent latency requirements, such as data centers, where even minimal interruptions can significantly degrade performance. These challenges high-

light the inherent difficulty of achieving precise and efficient anomaly detection in programmable data planes, especially when resource constraints and operational continuity must be balanced. Hence, although multiple works have pushed the boundaries of anomaly detection in P4 in terms of promptness and precision with relatively low added overhead and seamless data plane updates ([Swa+22; Zhe+22; CSF22; Liu+16]), we argue that still a lot can be done (an example can be what we propose in Chapter 3).

Full network programmability: While programmable P4 aim to fully softwarize networking hardware, the practical implementation often falls short due to the constraints imposed by specific hardware targets. These targets typically expose hardware-specific APIs that limit the expressiveness of P4, restricting the ability of developers to fully leverage the language's potential [Pat+18]. For example, writing a P4 pipeline for a Tofino [Int20] switch is significantly more complex than for a bmv2 software switch, as Tofino imposes strict resource limitations such as fixed memory sizes, table depths, and pipeline stages. Such differences force developers to make target-specific pipelines and trade-offs in their designs, ultimately making more error-prone and less portable code. These limitations extend beyond programming expressiveness to the entire detection system, influencing how effectively anomalies can be identified and managed. This creates challenges for developers, who must navigate the intricacies of specific platforms without a unified perspective on available resources. To address this, there is a need for a reliable and stable intermediate representation of system requirements [Krö+24]. Such a representation would allow network engineers to specify their needs in a platform-agnostic manner, offering a consistent and clear view of the underlying infrastructure. This approach would not only streamline resource management but also enable a more holistic understanding of the programmable data plane's capabilities, fostering better alignment between system requirements and hardware realities.

Fully unleashing P4 capabilities in IIoT threat monitoring: P4 has significant potential as a foundation for an Intrusion Detection System (IDS) in Industrial Internet of Things (IIoT) environments, offering the flexibility and pro-

grammability required to address the unique challenges of these systems [SSG21; Zan+22; Zan+23]. My thesis has only begun to explore the surface of what is possible with P4 in this context. With its ability to define custom packet parsers and match-action pipelines, P4 can be used to develop solutions tailored to the heterogeneous characteristic of industrial networks, including implementing custom protocols.

P4 nodes are capable to be programmed to take localized decisions, such as dropping suspicious packets, deprioritizing non-critical traffic, or rerouting it for deeper inspection. This decision-making process could be further developed with distributed traffic analysis, allowing multiple P4 devices to collaboratively enforce security policies or inject ad-hoc pipelines to enhance security across the network (we discussed some examples in Chapters 5 and 6). This vision of seamlessly adding security would be particularly valuable in IIoT networks, where downtime should always be avoided.

The integration of Distributed Ledger Technologies (DLTs), such as IOTA [AS+23], into the IIoT security infrastructure adds another layer of robustness by providing an immutable and distributed archive of critical information. For example, device certificates and permissions could be securely stored on a DLT, allowing the network to verify whether a specific device is authorized to operate within the system. The immutable nature of DLTs ensures that this information cannot be tampered with, providing absolute certainty and trust in the authentication process. This combination of P4's programmability and DLTs' security properties could create a powerful and resilient system capable of adapting to threats while maintaining a trusted, decentralized repository for critical data. However, fully realizing this potential will require further research into scalable designs, real-time reconfigurability, and robust anomaly detection algorithms tailored to P4-based systems.

Detecting otherwise invisibile threats with ad-hoc data structures: P4 presents a powerful yet underutilized tool for detecting traffic anomalies directly within the network, offering the potential to address sophisticated attacks that evade traditional detection methods. One such example is pulse wave attacks, where adversaries send sharp, transient spikes in traffic that often go undetected

[Kie23]. Often times, monitoring systems – especially those that base their detection on polling statistics from the data plane – fail to capture these brief anomalies. Current state-of-the-art solutions, such as ACC-turbo [Alc+22], attempt to mitigate this issue by clustering traffic based on features extracted from headers, such as IP or TCP fields, and deprioritizing traffic that exhibits similar patterns. However, this approach can be circumvented by designing attack traffic that deliberately avoids similarity in the monitored features – such as generating traffic with a wide range of dissimilar IP addresses to bypass clustering mechanisms based on IP fields.

In my ongoing research, I am exploring the potential of reconfigurable registers, such as Stat4 [GHV21], as a way to address this challenge. These structures can be used to sample traffic distributions and track key statistics like packet counts, SYN requests, or distinct flows over time. Leveraging such mechanisms, P4-based systems could detect deviations in traffic patterns indicative of anomalies with greater precision. Additionally, by deploying these data structures across different nodes and configuring them to monitor distinct statistics, it becomes possible to drill down into the specifics of an attack, offering a scalable and distributed approach to anomaly detection. This research underscores how P4's flexibility can address critical gaps in current detection systems, but much remains to be explored to fully harness its potential.

Conclusions

In this thesis, we explored how the programmable data plane can serve as a pivotal infrastructural component in the network, to enhance overall system security. By leveraging the flexibility of P4, we developed multiple frameworks and proof-of-concept solutions to seamlessly detect and mitigate complex attacks, provide in-network computation to bolster security, and achieve faster and more efficient monitoring by reducing the data processed by the monitoring system. These contributions showcased the versatility of P4 in multiple systems, regardless of size or available resources, by designing simple pipelines with minimal processing stages and memory usage.

However, we also acknowledged the inherent challenges of network device softwarization, particularly its potential impact on packet processing and traffic disruption in high-speed environments. Despite these limitations, our research showed that little-to-no-state P4 pipelines can effectively detect threats while maintaining high-speed performance, underscoring their practical viability.

Looking ahead, there remains significant potential to further advance in-network security. Probing the network to extract only essential monitoring information could facilitate early threat detection and swift countermeasure deployment. Additionally, establishing standardized guidelines for resource-efficient in-network computation would enable broader adoption of such techniques while ensuring minimal impact on network operations.

This thesis underscores the broader potential of P4 as an integral component for intrusion detection and system security. Particularly in IIoT and legacy systems – where traditional monitoring solutions are either impractical or non-existent – data plane programmability offers an unprecedented opportunity to empower system administrators to anticipate and counter threats effectively. By enabling

affordable and lightweight security appliances, P4 – and similar technologies either existing or that will be developed in the future – can significantly enhance the security posture of systems that are otherwise challenging and costly to protect.

Bibliography

[AEHHK19] Zakaria Abou El Houda, Abdelhakim Senhaji Hafid, and Lyes Khoukhi. "Cochain-SC: An intra-and inter-domain DDoS mitigation scheme based on blockchain using SDN and smart contract". In: *IEEE Access* 7 (2019), pp. 98893–98907.

[AHZ19] Muhammad Rizwan Asghar, Qinwen Hu, and Sherali Zeadally. "Cybersecurity in industrial control systems: Issues, technologies, and challenges". In: *Computer Networks* 165 (2019), p. 106946.

[Ala+18] Muhammad Alam et al. "Orchestration of Microservices for IoT Using Docker and Edge Computing". In: *IEEE Communications Magazine* 56.9 (2018), pp. 118–123. DOI: 10.1109/MCOM.2018.1701233.

[Alc+22] Albert Gran Alcoz et al. "Aggregate-based congestion control for pulse-wave DDoS defense". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. 2022, pp. 693–706.

[Als+21] Amir Alsadi et al. "A Security Monitoring Architecture based on Data Plane Programmability". In: *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE. 2021, pp. 389–394.

[AlS+22] Ali AlSabeh et al. "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment". In: *Computer Networks* 207 (2022), p. 108800.

[Als+22] Mays Alshaikhli et al. “Evolution of Internet of Things From Blockchain to IOTA: A Survey”. In: *IEEE Access* 10 (2022), pp. 844–866. DOI: 10.1109/ACCESS.2021.3138353.

[ALS23] G. Apruzzese, P. Laskov, and J. Schneider. “SoK: Pragmatic Assessment of Machine Learning for Network Intrusion Detection”. In: *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. Los Alamitos, CA, USA: IEEE Computer Society, 2023, pp. 592–614. DOI: 10.1109/EuroSP57164.2023.00042. URL: https://doi.ieeecomputersociety.org/10.1109/EuroSP57164.2023.00042.

[AMD19] AMD. *Naples DSC-100 Distributed Services Card*. https://pensando.io/assets/documents/Naples_100_ProductBrief-10-2019.pdf. 2019.

[AS+23] Amir Al Sadi et al. “P-IOTA: A cloud-based geographically distributed threat alert system that leverages P4 and IOTA”. In: *Sensors* 23.6 (2023), p. 2955.

[AS+24] Amir Al Sadi et al. “Unleashing Dynamic Pipeline Reconfiguration of P4 Switches for Efficient Network Monitoring”. In: *IEEE Transactions on Network and Service Management* (2024).

[Auh+22] Zachary Auhl et al. “A Comparative Study of Consensus Mechanisms in Blockchain for IoT Networks”. In: *Electronics* 11.17 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11172694. URL: https://www.mdpi.com/2079-9292/11/17/2694.

[AVOW18] AbdelRahman Abdou, Paul C Van Oorschot, and Tao Wan. “Comparative analysis of control plane security of SDN and conventional networks”. In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 3542–3559.

[AYC20] Soheil Abbasloo, Chen-Yu Yen, and H. Jonathan Chao. “Classic Meets Modern: A Pragmatic Learning-Based Congestion Control for the Internet”. In: *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications,*

Technologies, Architectures, and Protocols for Computer Communication. SIGCOMM '20. Virtual Event, USA: Association for Computing Machinery, 2020, 632–647. ISBN: 9781450379557. DOI: 10.1145/3387514.3405892. URL: https://doi.org/10.1145/3387514.3405892.

[Bar+21] Diogo Barradas et al. "FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications." In: *NDSS*. 2021.

[BB+16] Ran Ben-Basat et al. "Heavy hitters in streams and sliding windows". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9.

[BB+18] Ran Ben-Basat et al. "Efficient measurement on programmable switches using probabilistic recirculation". In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE. 2018, pp. 313–323.

[Ber+23] Davide Berardi et al. "When Operation Technology Meets Information Technology: Challenges and Opportunities". In: *Future Internet* 15.3 (2023), p. 95.

[BG+19] Coralie Busse-Grawitz et al. "pforest: In-network inference with random forests". In: *arXiv preprint arXiv:1909.05680* (2019).

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[BM61] Robert G Brown and Richard F Meyer. "The fundamental theorem of exponential smoothing". In: *Operations Research* 9.5 (1961), pp. 673–685.

[Bon+18] Gaetano Bonofiglio et al. "Katharà: A container-based framework for implementing network function virtualization and software defined networks". In: *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE. 2018, pp. 1–9.

[Bor+23] Davide Borsatti et al. "Mission Critical Communications Support With 5G and Network Slicing". In: *IEEE Transactions on Network and Service Management* 20.1 (2023), pp. 595–607. DOI: 10.1109/TNSM.2022.3208657.

[Bos+13] Pat Bosshart et al. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 99–110.

[Bos+14] Pat Bosshart et al. "P4: Programming protocol-independent packet processors". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014), pp. 87–95.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[BS17] Sadhu Ram Basnet and Subarna Shakya. "BSS: Blockchain security over software defined network". In: *2017 International Conference on Computing, Communication and Automation (ICCCA)*. 2017, pp. 720–725. DOI: 10.1109/CCAA.2017.8229910.

[BS+23] Freddie Bickford Smith et al. "Prediction-Oriented Bayesian Active Learning". In: *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. Ed. by Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent. Vol. 206. Proceedings of Machine Learning Research. Valencia, Spain: PMLR, 2023, pp. 7331–7348. URL: https://proceedings.mlr.press/v206/bickfordsmith23a.html.

[But17] Brandon Butler. *What P4 programming is and why it's such a big deal for Software Defined Networking*. 2017. URL: https://www.networkworld.com/article/3163496/what-p4-programming-is-and-why-it-s-such-a-big-deal-for-software-defined-networking.html.

[CAI19] CAIDA. *The CAIDA UCSD Anonymized Internet Traces - 2018-2019*. 2019. URL: https://www.caida.org/catalog/datasets/passive_dataset.

[Cal+16] F. Callegati et al. “SDN for dynamic NFV deployment”. In: *IEEE Communications Magazine* 54.10 (2016), pp. 89–95. DOI: 10.1109/MCOM.2016.7588275.

[Cap+19] Maurantonio Caprolu et al. “Edge computing perspectives: Architectures, technologies, and open security issues”. In: *2019 IEEE International Conference on Edge Computing (EDGE)*. IEEE. 2019, pp. 116–123.

[CDT21] Mauro Conti, Denis Donadel, and Federico Turrin. “A survey on industrial control system testbeds and datasets for security research”. In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2248–2294.

[Cha+18] Anamika Chauhan et al. “Blockchain and Scalability”. In: *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2018, pp. 122–128. DOI: 10.1109/QRS-C.2018.00034.

[Cha+23] Brian Chang et al. “Learned Load Balancing”. In: *Proceedings of the 24th International Conference on Distributed Computing and Networking*. ICDCN ’23. Kharagpur, India: Association for Computing Machinery, 2023, 177–187. ISBN: 9781450397964. DOI: 10.1145/3571306.3571403. URL: https://doi.org/10.1145/3571306.3571403.

[Che20] Xiaoqi Chen. “Implementing AES encryption on programmable switches via scrambled lookup tables”. In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 8–14.

[Cic] “CICIDS2017 Dataset”. In: *https://www.unb.ca/cic/datasets/ids-2017.html* ().

[Cis96] Cisco Systems. *Cisco NetFlow: Traffic Monitoring and Analysis*. Technical Documentation. Available at https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html. 1996.

[CM05] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001.

[Cona] P4 Language Consortium. *extern*. https://p4.org/p4-spec/docs/P4-16-working-spec.html#sec-external-units.

[Conb] P4 Language Consortium. *v1model*. https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4.

[Con+19] P4 Language Consortium et al. *p4lang/behavioral-model*. 2019.

[Con20] The P4 Language Consortium. *P4Runtime Specification*. 2020. URL: https://p4.org/p4-spec/p4runtime/v1.3.0/P4Runtime-Spec.pdf.

[Con21] The P4 Language Consortium. *P4 Language Specification*. 2021. URL: https://p4.org/p4-spec/docs/P4-16-v1.2.2.pdf.

[CSF22] Bruno Coelho and Alberto Schaeffer-Filho. "BACKORDERS: using random forests to detect DDoS attacks in programmable data planes". In: *Proceedings of the 5th International Workshop on P4 in Europe*. 2022, pp. 1–7.

[Dar+17] Tooska Dargahi et al. "A survey on the security of stateful SDN data planes". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1701–1725.

[Dav+21] Gianluca Davoli et al. "A fog computing orchestrator architecture with service model awareness". In: *IEEE Transactions on Network and Service Management* 19.3 (2021), pp. 2131–2147.

[DB+17] Mathias Santos De Brito et al. "A service orchestration architecture for fog-enabled infrastructures". In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2017, pp. 127–132.

[DC+20] R. Doriguzzi-Corin et al. “Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection”. In: *IEEE Transactions on Network and Service Management* 17.2 (2020), pp. 876–889. DOI: 10.1109/TNSM.2020.2971776.

[DC+23a] Nicola Di Cicco et al. “DRL-FORCH: A Scalable Deep Reinforcement Learning-based Fog Computing Orchestrator”. In: *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*. IEEE. 2023, pp. 125–133.

[DC+23b] Nicola Di Cicco et al. “Poster: Continual Network Learning”. In: *Proceedings of the ACM SIGCOMM 2023 Conference*. 2023, pp. 1096–1098.

[DC+24] Roberto Doriguzzi-Corin et al. “Introducing packet-level analysis in programmable data planes to advance Network Intrusion Detection”. In: *Computer Networks* 239 (2024), p. 110162.

[Ddo] *DDoS 2007 attack*. https://catalog.caida.org/dataset/ddos_attack_2007. Accessed: 2023-6-15. DOI: https://catalog.caida.org/dataset/ddos_attack_2007.

[DG08] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Commun. ACM* 51.1 (2008), 107–113.

[Din] Damu Ding. *InDDoS*. https://github.com/DINGDAMU/INDDoS.

[Din+21] Damu Ding et al. “In-network volumetric DDoS victim identification using programmable commodity switches”. In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1191–1202.

[Din+22] Damu Ding et al. “Design and Development of Network Monitoring Strategies in P4-enabled Programmable Switches”. In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022.

[DL+22] Matthias De Lange et al. “A Continual Learning Survey: Defying Forgetting in Classification Tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.7 (2022), pp. 3366–3385. DOI: 10.1109/TPAMI.2021.3057446.

[Don+24] Chao Dong et al. “A survey on the network models applied in the industrial network optimization”. In: *Science China Information Sciences* 67.2 (2024), p. 121301.

[dor] doriguzzi. *lucid-ddos*. https://github.com/doriguzzi/lucid-ddos/tree/master/sample-dataset.

[DSS21] Damu Ding, Marco Savi, and Domenico Siracusa. “Tracking normalized network traffic entropy to detect DDoS attacks in P4”. In: *IEEE Transactions on Dependable and Secure Computing* 19.6 (2021), pp. 4019–4031.

[Ela+16] Barker Elaine et al. “Recommendation for Key Management, Part 1: General”. In: *NIST Special Publication* (2016), pp. 51–54.

[EMT22] Alessandro Erba, Anne Müller, and Nils Ole Tippenhauer. “Security Analysis of Vendor Implementations of the OPC UA Protocol for Industrial Control Systems”. In: *Proceedings of the 4th Workshop on CPS & IoT Security and Privacy*. 2022, pp. 1–13.

[Eng+19] Patrick Engelhard et al. “Toward scalable and virtualized massive wireless sensor networks”. In: *2019 International Conference on Networked Systems (NetSys)*. IEEE. 2019, pp. 1–6.

[Eth22] Ethereum. *Ethereum: Proof-of-stake (POS)*. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/. 2022.

[Feb+22] Aldo Febro et al. “Synchronizing DDoS defense at network edge with P4, SDN, and Blockchain”. In: *Computer Networks* 216 (2022), p. 109267. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2022.109267. URL: https://www.sciencedirect.com/science/article/pii/S1389128622003310.

[Fen+21] Yong Feng et al. "In-Situ Programmable Switching Using RP4: Towards Runtime Data Plane Programmability". In: *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. HotNets '21. Virtual Event, United Kingdom: Association for Computing Machinery, 2021, 69–76. ISBN: 9781450390873. DOI: 10.1145/3484266.3487367. URL: https://doi.org/10.1145/3484266.3487367.

[Gar22] Gartner. *Operational Technology Security Reviews 2022*. https://www.gartner.com/reviews/market/operational-technology-security.. Accessed: August 26, 2022. 2022.

[Gho18] Morteza Ghobakhloo. "The future of manufacturing industry: a strategic roadmap toward Industry 4.0". In: *Journal of manufacturing technology management* (2018).

[GHV21] Sam Gao, Mark Handley, and Stefano Vissicchio. "Stats 101 in p4: towards in-switch anomaly detection". In: *Proceedings of the 20th ACM Workshop on Hot Topics in Networks*. 2021, pp. 84–90.

[Gil16] Daniel Kahn Gillmor. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*. RFC 7919. Aug. 2016. DOI: 10.17487/RFC7919. URL: https://www.rfc-editor.org/info/rfc7919.

[GK18] Maryam Ghanbari and Witold Kinsner. "Extracting features from both the input and the output of a convolutional neural network to detect distributed denial of service attacks". In: *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE. 2018, pp. 138–144.

[Gom+17] Heitor M. Gomes et al. "Adaptive random forests for evolving data stream classification". en. In: *Machine Learning* 106.9 (2017), 1469–1495. ISSN: 1573-0565. DOI: 10.1007/s10994-017-5642-8.

[Goo23] Google. *The Go programming language*. 2023.

[Gro16] Architecture Working Group. *View on 5G Architecture*. Tech. rep. Available on-line at https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-July-2016.pdf. 5G PPP, 2016.

[Gyö+23] Csaba Györgyi et al. “Adaptive Network Traffic Reduction on the Fly With Programmable Data Planes”. In: *IEEE Access* 11 (2023), pp. 24935–24944.

[Han+22] Hui Han et al. “Applications of sketches in network traffic measurement: A survey”. In: *Information Fusion* 82 (2022), pp. 58–85.

[HF+18] Kevin E Hemsley, E Fisher, et al. *History of industrial control system cyber incidents*. Tech. rep. Idaho National Lab.(INL), Idaho Falls, ID (United States), 2018.

[HHH22] Toshiaki Honda, Takashi Hamaguchi, and Yoshihiro Hashimoto. “OPC UA information transfer via unidirectional data diode for ICS cyber security”. In: *Computer Aided Chemical Engineering*. Vol. 49. Elsevier, 2022, pp. 1459–1464.

[HMS24] HMS. *Industrial network market shares 2023 according to HMS Networks*. 2024. URL: https://www.hms-networks.com/news-and-insights/news-from-hms/2023/05/05/industrial-network-market-shares-2023 (visited on 01/29/2024).

[Hou+11] Neil Houlsby et al. *Bayesian Active Learning for Classification and Preference Learning*. 2011. arXiv: 1112.5745 [stat.ML].

[Hyn11] Rob J Hyndman. *Moving Averages*. 2011.

[HZL17] Zecheng He, Tianwei Zhang, and Ruby B Lee. “Machine learning based DDoS attack detection from source side in cloud”. In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2017, pp. 114–120.

[Int20] Intel. *Tofino: P4-programmable Ethernet switch ASIC*. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html. 2020.

[IOT22] IOTA Wiki. *Energy Efficiency*. https://wiki.iota.org/learn/about-iota/energy-efficiency/. 2022.

[Iso] *ISO/IEC 62443-3-3: Industrial communication networks - Network and system security - Part 3-3: System security requirements and security levels*. Available from ISO, https://www.iso.org/standard/65696.html. International Organization for Standardization. 2013.

[Jac88] Van Jacobson. *Traceroute: Network Diagnostic Tool*. Technical Software Tool. Available at `https://traceroute.org/`. 1988.

[Jan21] Santosh Janardhan. *Facebook Outage on October 4th 2021*. `https://engineering.fb.com/2021/10/04/networking-traffic/outage/`. 2021.

[Jey+14] Vimalkumar Jeyakumar et al. "Millions of little minions: Using packets for low latency network programming and visibility". In: *ACM SIGCOMM Computer Communication Review* 44.4 (2014), pp. 3–14.

[Jia+19] Weng Jiasi et al. "Secure software-defined networking based on blockchain". In: *arXiv preprint arXiv:1906.04342* (2019).

[Kat+17] Naga Katta et al. "Clove: Congestion-aware load balancing at the virtual edge". In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 2017, pp. 323–335.

[KCBH21] Elie F Kfoury, Jorge Crichigno, and Elias Bou-Harb. "An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends". In: *IEEE Access* 9 (2021), pp. 87094–87155.

[Kie23] Pascal Kiechl. "Simulator of Distributed Datasets for Pulse-wave DDoS Attacks". MA thesis. University of Zurich, 2023.

[Kir+17] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. DOI: `10.1073/pnas.1611835114`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.1611835114`. URL: `https://www.pnas.org/doi/abs/10.1073/pnas.1611835114`.

[Kon+22] Linghe Kong et al. “Edge-computing-driven internet of things: A survey”. In: *ACM Computing Surveys* 55.8 (2022), pp. 1–41.

[Krö+24] Nicolai Kröger et al. “Performance Modeling and Analysis of P4 Programmable Devices With General Service Times”. In: *IEEE Transactions on Network and Service Management* (2024).

[KSG14] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. “Mininet as software defined networking testing platform”. In: *International conference on communication, computing & systems (ICCCS)*. 2014, pp. 139–42.

[KT22] Somayeh Kianpisheh and Tarik Taleb. “A survey on in-network computing: Programmable data plane and technology specific applications”. In: *IEEE Communications Surveys & Tutorials* 25.1 (2022), pp. 701–761.

[Li+19] Yuliang Li et al. “HPCC: High precision congestion control”. In: *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 44–58.

[Lia+23] Athanasios Liatifis et al. “Advancing sdn from openflow to p4: A survey”. In: *ACM Computing Surveys* 55.9 (2023), pp. 1–37.

[Lin+20] Ting-Yu Lin et al. “Mitigating SYN flooding attack and ARP spoofing in SDN data plane”. In: *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2020, pp. 114–119.

[Lin22] Linux foundation. *hping3*. `http://wiki.hping.org/`. 2022.

[Lin23] Linux foundation. *tcpreplay*. `https://linux.die.net/man/1/tcpreplay`. 2023.

[Liu+16] Zaoxing Liu et al. “One sketch to rule them all: Rethinking network flow monitoring with univmon”. In: *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, pp. 101–114.

[Liu+21] Zaoxing Liu et al. "Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches". In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 3829–3846.

[Mal+22] Ankur Mallick et al. "Matchmaker: Data Drift Mitigation in Machine Learning for Large-Scale Systems". In: *Proceedings of Machine Learning and Systems*. Ed. by D. Marculescu, Y. Chi, and C. Wu. Vol. 4. 2022, pp. 77–94. URL: https://proceedings.mlsys.org/paper_files/paper/2022/file/069a002768bcb31509d4901961f23b3c-Paper.pdf.

[Mao+17] Yuyi Mao et al. "A survey on mobile edge computing: The communication perspective". In: *IEEE communications surveys & tutorials* 19.4 (2017), pp. 2322–2358.

[Maz+22] Carlo Mazzocca et al. "Evaluating Tangle Distributed Ledger for Access Control Policy Distribution in Multi-region Cloud Environments". In: *Quality of Information and Communications Technology*. Cham: Springer International Publishing, 2022, pp. 296–306. ISBN: 978-3-031-14179-9.

[MC18] Pavol Mulinka and Pedro Casas. "Adaptive Network Security through Stream Machine Learning". In: *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*. SIGCOMM '18. Budapest, Hungary: Association for Computing Machinery, 2018, 4–5. ISBN: 9781450359153. DOI: 10.1145/3234200.3234246. URL: https://doi.org/10.1145/3234200.3234246.

[McK09] Nick McKeown. "Software-defined networking". In: *INFOCOM keynote talk* 17.2 (2009), pp. 30–32.

[Mel+20] Andrea Melis et al. "P-SCOR: Integration of constraint programming orchestration and programmable data plane". In: *IEEE Transactions on Network and Service Management* 18.1 (2020), pp. 402–414.

[Mel+23] Andrea Melis et al. “A systematic literature review of offensive and defensive security solutions with software defined network”. In: *IEEE Access* 11 (2023), pp. 93431–93463.

[MGB21] Sebastian Gomez Macias, Luciano Paschoal Gaspary, and Juan Felipe Botero. “Oracle: An architecture for collaboration of data and control planes to detect ddos attacks”. In: *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2021, pp. 962–967.

[Min21] Mininet Project Contributors. *Mininet*. `http://mininet.org/`. Nov. 29, 2021.

[Moda] *MODBUS Application Protocol Specification V1.1b3*. Protocol Specification. Modbus Organization, Apr. 2012.

[Modb] *MODBUS/TCP Security*. Protocol Specification. Modbus Organization, July 2018.

[Mus+22] Francesco Musumeci et al. “Machine-learning-enabled DDoS attacks detection in P4 programmable networks”. In: *Journal of Network and Systems Management* 30 (2022), pp. 1–27.

[Muu83] Mike Muuss. *The PING Program*. Technical Software Tool. Available at `https://www.ping127001.com/`. 1983.

[OCD16] Opeyemi Osanaiye, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. “Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework”. In: *Journal of Network and Computer Applications* 67 (2016), pp. 147–165.

[ON15] Keiron O’Shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[OPC19] OPC Foundation. *OPC 40100-1: Machine Vision - Control, Configuration management, recipe management, result management*. 2019. URL: `https://reference.opcfoundation.org/MachineVision/v100/docs/`.

[OPC22a] OPC Foundation. *OPC Unified Architecture Specification Part 1: Overview and Concepts*. 2022. URL: https://reference.opcfoundation.org/Core/Part1/v105/docs/.

[OPC22b] OPC Foundation. *OPC Unified Architecture Specification Part 2: Security*. 2022. URL: https://reference.opcfoundation.org/Core/Part2/v105/docs/.

[Ope] *OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)*. 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[Ope17] OpenFog Consortium Architecture Working Group. "OpenFog reference architecture for fog computing". In: *OPFRA001* 20817 (2017), p. 162.

[Pat+18] P Gyanesh Kumar Patra et al. "Toward a sweet spot of data plane programmability, portability, and performance: On the scalability of multi-architecture P4 pipelines". In: *IEEE Journal on Selected Areas in Communications* 36.12 (2018), pp. 2603–2611.

[Pat+22] Shivam Patel et al. "In-Network Fractional Calculations Using P4 for Scientific Computing Workloads". In: *Proceedings of the 5th International Workshop on P4 in Europe*. EuroP4 '22. Rome, Italy: Association for Computing Machinery, 2022, 33–38. ISBN: 9781450399357. DOI: 10.1145/3565475.3569083. URL: https://doi.org/10.1145/3565475.3569083.

[Pfa+15] Ben Pfaff et al. "The design and implementation of open {vSwitch}". In: *12th USENIX symposium on networked systems design and implementation (NSDI 15)*. 2015, pp. 117–130.

[Pit+22] Gaetano Francesco Pittalà et al. "Function-as-a-Service Orchestration in Fog Computing Environments". In: *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE. 2022, pp. 364–366.

[Pit+24] Gaetano Francesco Pittalà et al. "Leveraging Data Plane Programmability to enhance service orchestration at the edge: A focus on industrial security". In: *Computer Networks* (2024), p. 110397.

[PL19] Serguei Popov and Q Lu. "IOTA: feeless and free". In: *IEEE Blockchain Technical Briefs* 964 (2019).

[Pop] Serguei Popov. "The tangle". In: *White paper (2018)* ().

[Pyt] Python. *pymodbus*. `https://pymodbus.readthedocs.io/en/latest/index.html`.

[Qia+22] Yu Qiao et al. "Deep or Statistical: An Empirical Study of Traffic Predictions on Multiple Time Scales". In: *Proceedings of the SIGCOMM '22 Poster and Demo Sessions*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, 10–12. ISBN: 9781450394345. DOI: 10.1145/3546037.3546048. URL: https://doi.org/10.1145/3546037.3546048.

[Qin+20] Qiaofeng Qin et al. "Line-speed and scalable intrusion detection at the network edge via federated learning". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 352–360.

[RAD15] Stefan Richter, Victor Alvarez, and Jens Dittrich. "A seven-dimensional analysis of hashing methods and its implications on query processing". In: *PVLDB* 9.3 (2015), pp. 96–107.

[Rah+21] Anichur Rahman et al. "SmartBlock-SDN: An Optimized Blockchain-SDN Framework for Resource Management in IoT". In: *IEEE Access* 9 (2021), pp. 28361–28376. DOI: 10.1109/ACCESS.2021.3058244.

[RAS23] Lorenzo Rinieri and Amir Al Sadi. *P4-Forch_KatharaTopo*. Version 1.0.1. Sept. 2023. DOI: 10.5281/zenodo.8376994. URL: https://github.com/UniboSecurityResearch/P4-Forch_KatharaTopo.

[Raz+22] Kamran Razavi et al. "Distributed DNN serving in the network data plane". In: *Proceedings of the 5th International Workshop on P4 in Europe*. 2022, pp. 67–70.

[Reb+17] S. Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 5533–5542. DOI: 10.1109/CVPR.2017.587. URL: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.587.

[Reb+21] Gabriel Antonio F. Rebello et al. "A security and performance analysis of proof-based consensus protocols". In: *Annals of Telecommunications* (2021). DOI: 10.1007/s12243-021-00896-2. URL: https://doi.org/10.1007/s12243-021-00896-2.

[Rin+24] Lorenzo Rinieri et al. "In-Network Encryption for Secure Industrial Control Systems Communications". In: *2024 IEEE 10th International Conference on Network Softwarization (NetSoft)*. IEEE. 2024, pp. 190–194.

[RR17] Danda B. Rawat and Swetha R. Reddy. "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey". In: *IEEE Communications Surveys & Tutorials* 19.1 (2017), pp. 325–346. DOI: 10.1109/COMST.2016.2618874.

[Sad+23] Amir Al Sadi et al. "Real-time Pipeline Reconfiguration of P4 Programmable Switches to Efficiently Detect and Mitigate DDoS Attacks". In: *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2023.

[Sam+21] Wojciech Samek et al. "Explaining deep neural networks and beyond: A review of methods and applications". In: *Proceedings of the IEEE* 109.3 (2021), pp. 247–278.

[San+17] Daniele Santoro et al. "Foggy: A platform for workload orchestration in a fog computing environment". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2017, pp. 231–234.

[Sar+23] Akanksha Saran et al. *Streaming Active Learning with Deep Neural Networks*. 2023. arXiv: 2303.02535 [cs.LG].

[SB18] Giuseppe Siracusano and Roberto Bifulco. *In-network Neural Networks*. 2018. arXiv: 1801.05731 [cs.DC].

[SC+19] Inés Sittón-Candanedo et al. "A review of edge computing reference architectures and a new global edge proposal". In: *Future Generation Computer Systems* 99 (2019), pp. 278–294.

[Sch+22] Eryk Schiller et al. "Landscape of IoT security". In: *Computer Science Review* 44 (2022), p. 100467.

[Set12] Burr Settles. *Active Learning*. en. Synthesis Lectures on Artificial Intelligence and Machine Learning. Cham, Switzerland: Springer Cham, 2012. ISBN: 978-3-031-00432-2. DOI: 10.1007/978-3-031-01560-1.

[Sha+17] Pradip Kumar Sharma et al. "DistBlockNet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks". In: *IEEE Communications Magazine* 55 (2017), pp. 78–85.

[Sha+19] Iman Sharafaldin et al. "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy". In: *2019 International Carnahan Conference on Security Technology (ICCST)*. Chennai, India: IEEE, 2019, pp. 1–8. DOI: 10.1109/CCST.2019.8888419.

[She+21] Zi-Yang Shen et al. "Mitigating SYN Flooding and UDP Flooding in P4-based SDN". In: *2021 22nd Asia-Pacific Network Operations and Management Symposium (APNOMS)*. IEEE. 2021, pp. 374–377.

[Shi+12] Ali Shiravi et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". In: *computers & security* 31.3 (2012), pp. 357–374.

[SI+20] Alexandre da Silveira Ilha et al. "Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation". In: *IEEE Transactions on Network and Service Management* 18.3 (2020), pp. 3121–3139.

[Sir+22] Giuseppe Siracusano et al. "Re-architecting traffic analysis with neural network interface cards". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 513–533.

[Son+16] Seungbeom Song et al. "A congestion avoidance algorithm in SDN environment". In: *2016 International Conference on Information Networking (ICOIN)*. IEEE. 2016, pp. 420–423.

[SS16] Shan Suthaharan and Shan Suthaharan. "Support vector machine". In: *Machine learning models and algorithms for big data classification: thinking with examples for effective learning* (2016), pp. 207–235.

[SSB18] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. "Can the Network Be the AI Accelerator?" In: *Proceedings of the 2018 Morning Workshop on In-Network Computing*. NetCompute '18. Budapest, Hungary: Association for Computing Machinery, 2018, 20–25. ISBN: 9781450359085. DOI: 10.1145/3229591.3229594. URL: https://doi.org/10.1145/3229591.3229594.

[SSG21] Ganesh C Sankaran, Krishna M Sivalingam, and Harsh Gondaliya. "P4 and netfpga based secure in-network computing architecture for ai-enabled industrial internet of things". In: *IEEE Internet of Things Journal* (2021).

[Swa+22] Tushar Swamy et al. "Taurus: a data plane architecture for per-packet ML". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022, pp. 1099–1114.

[Swa+23] Tushar Swamy et al. "Homunculus: Auto-Generating Efficient Data-Plane ML Pipelines for Datacenter Networks". In: *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. ASPLOS 2023. Vancouver, BC, Canada: Association for Computing Machinery, 2023, 329–342. ISBN: 9781450399180. DOI: 10.1145/

3582016.3582022. URL: https://doi.org/10.1145/3582016.3582022.

[SZ20] Radostin Stoyanov and Noa Zilberman. "MTPSA: Multi-Tenant Programmable Switches". In: *Proceedings of the 3rd P4 Workshop in Europe*. EuroP4'20. Barcelona, Spain: Association for Computing Machinery, 2020, 43–48. ISBN: 9781450381819. DOI: 10.1145/3426744.3431329. URL: https://doi.org/10.1145/3426744.3431329.

[SZA21] Ravid Shwartz-Ziv and Amitai Armon. "Tabular Data: Deep Learning is Not All You Need". In: *8th ICML Workshop on Automated Machine Learning (AutoML)*. 2021. URL: https://openreview.net/forum?id=vdgtepS1pV.

[Tac+22] Takuji Tachibana et al. "Open Multi-Access Network Platform with Dynamic Task Offloading and Intelligent Resource Monitoring". In: *IEEE Communications Magazine* 60.8 (2022), pp. 52–58.

[Tan+21] Lizhuang Tan et al. "In-band network telemetry: A survey". In: *Computer Networks* 186 (2021), p. 107763.

[THL20] Lu Tang, Qun Huang, and Patrick PC Lee. "A fast and compact invertible sketch for network-wide heavy flow detection". In: *IEEE/ACM Transactions on Networking* 28.5 (2020), pp. 2350–2363.

[TPK17] C. Tselios, I. Politis, and S. Kotsopoulos. "Enhancing SDN security for IoT-related deployments through blockchain". In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. 2017, pp. 303–308. DOI: 10.1109/NFV-SDN.2017.8169860.

[Udd+18] Mostafa Uddin et al. "SDN-based multi-protocol edge switching for IoT service automation". In: *IEEE Journal on Selected Areas in Communications* 36.12 (2018), pp. 2775–2786.

[Uni] UniboSecurityResearch. *p4-runtime-go-client*. https://github.com/UniboSecurityResearch/p4runtime-go-client/tree/cnn-integration.

[Vel+17] Karima Velasquez et al. "Service Orchestration in Fog Environments". In: *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017, pp. 329–336. DOI: 10.1109/FiCloud.2017.49.

[Ves+20] Jonathan Vestin et al. "Toward in-network event detection and filtering for publish/subscribe communication using programmable data planes". In: *IEEE Transactions on Network and Service Management* 18.1 (2020), pp. 415–428.

[Wan+22] Tao Wang et al. "Isolation Mechanisms for High-Speed Packet-Processing Pipelines". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1289–1305. ISBN: 978-1-939133-27-4. URL: https://www.usenix.org/conference/nsdi22/presentation/wang-tao.

[Wic+22] Matthias Wichtlhuber et al. "IXP Scrubber: Learning from Blackholing Traffic for ML-Driven DDoS Detection at Scale". In: *Proceedings of the ACM SIGCOMM 2022 Conference*. SIGCOMM '22. Amsterdam, Netherlands: Association for Computing Machinery, 2022, 707–722. ISBN: 9781450394208. DOI: 10.1145/3544216.3544268. URL: https://doi.org/10.1145/3544216.3544268.

[Wil94] Theodore J Williams. "The Purdue enterprise reference architecture". In: *Computers in industry* 24.2-3 (1994), pp. 141–158.

[Xie+18] Junfeng Xie et al. "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges". In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 393–430.

[Xin+22] Jiarong Xing et al. "Runtime programmable switches". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 651–665.

[XZ19] Zhaoqi Xiong and Noa Zilberman. "Do Switches Dream of Machine Learning? Toward In-Network Classification". In: *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. HotNets '19. Princeton, NJ, USA: Association for Computing Machinery, 2019, 25–33. ISBN: 9781450370202. DOI: 10.1145/3365609.3365864. URL: https://doi.org/10.1145/3365609.3365864.

[Yan24] Boyang Yan. "Workload Prediction in P4 Programmable Switches: Smart Resource Scheduling". In: *arXiv preprint arXiv:2405.11408* (2024).

[Yaz+20a] Abbas Yazdinejad et al. "An Energy-Efficient SDN Controller Architecture for IoT Networks With Blockchain-Based Security". In: *IEEE Transactions on Services Computing* 13.4 (2020), pp. 625–638. DOI: 10.1109/TSC.2020.2966970.

[Yaz+20b] Abbas Yazdinejad et al. "P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking". In: *Computers & Security* 88 (2020), p. 101629. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2019.101629. URL: https://www.sciencedirect.com/science/article/pii/S0167404819301762.

[YLL17] Xiaoyong Yuan, Chuanhuang Li, and Xiaolin Li. "DeepDefense: identifying DDoS attack via deep learning". In: *2017 IEEE international conference on smart computing (SMARTCOMP)*. IEEE. 2017, pp. 1–8.

[Yu+18] Wei Yu et al. "A Survey on the Edge Computing for the Internet of Things". In: *IEEE Access* 6 (2018), pp. 6900–6919. DOI: 10.1109/ACCESS.2017.2778504.

[YY15] Qiao Yan and F Richard Yu. "Distributed denial of service attacks in software-defined networking with cloud computing". In: *IEEE Communications Magazine* 53.4 (2015), pp. 52–59.

[Zan+22] Mingyuan Zang et al. "P4pir: in-network analysis for smart iot gateways". In: *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*. 2022, pp. 46–48.

[Zan+23] Mingyuan Zang et al. “Advanced threat defense with in-network traffic analysis for iot gateways”. In: *Proc. MobiUK* (2023), p. 15.

[ZBH18] Peng Zheng, Theophilus Benson, and Chengchen Hu. “P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs”. In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’18. Heraklion, Greece: Association for Computing Machinery, 2018, 98–111. ISBN: 9781450360807. DOI: 10.1145/3281411.3281436. URL: https://doi.org/10.1145/3281411.3281436.

[Zha+18] Menghao Zhang et al. “Control plane reflection attacks in SDNs: New attacks and countermeasures”. In: *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*. Springer. 2018, pp. 161–183.

[Zha+19] Yanling Zhao et al. “A survey of networking applications applying the software defined networking concept based on machine learning”. In: *IEEE Access* 7 (2019), pp. 95397–95417.

[Zha+20] Menghao Zhang et al. “Poseidon: Mitigating volumetric ddos attacks with programmable switches”. In: *the 27th Network and Distributed System Security Symposium (NDSS 2020)*. 2020.

[Zhe+22] Changgang Zheng et al. “IIsy: Practical in-network classification”. In: *arXiv preprint arXiv:2205.08243* (2022).