GRAPH-BASED FEATURE ENRICHMENT FOR ONLINE INTRUSION DETECTION IN VIRTUAL NETWORKS

Igor Jochem Sanz

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Otto Carlos Muniz Bandeira Duarte

Rio de Janeiro
Dezembro de 2018

GRAPH-BASED FEATURE ENRICHMENT FOR ONLINE INTRUSION DETECTION IN VIRTUAL NETWORKS

Igor Jochem Sanz

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Prof. Diego Leonel Cadette Dutra, Dr.

Prof. Altair Olivo Santin, Dr.

Dra. Cristine Hoepers, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2018

Sanz, Igor Jochem

Graph-based Feature Enrichment for Online Intrusion Detection in Virtual Networks/Igor Jochem Sanz. – Rio de Janeiro: UFRJ/COPPE, 2018.

XIII, 61 p.: il.; 29, 7cm.

Orientador: Otto Carlos Muniz Bandeira Duarte

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 52 – 61.

1. Intrusion Detection. 2. Machine Learning. 3. Network Function Virtualization. 4. Graph Processing. I. Duarte, Otto Carlos Muniz Bandeira. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família.

Acknowledgments

First, I would like to thank my family for always supporting me in all moments, especially to my Mom and my Dad, in which I own everything accomplished so far.

I would like to thank all friends I have made in GTA/UFRJ lab, especially to Martin Andreoni, Diogo Mattos, Gabriel Rebello, Antonio Lobato, Igor Alvarenga, Leopoldo Mauricio, and Hugo Sadok, in which I have been fortunate to meet and co-work with during all these years of research. Thanks for the good and fun moments and for the important discussions, which contributed to the construction of this manuscript.

I thank my advisor and friend, Professor Otto Carlos Duarte, for introducing me to academia and research world, for the motivation to always keep focusing, for the opportunities provided, for the challenging moments, for the patience, and for all good experiences obtained in these years of GTA.

I thank Professors Luís Henrique Costa, Miguel Elias Campista, Pedro Velloso, and Rodrigo Couto, for their friendship, the learning shared and for maintaining a good working environment in GTA/UFRJ. I also thank Professors Marcelo Rubinstein, Guy Pujolle, and Álvaro Cardenas for the contributions to this work.

I thank Professors Diego Dutra, Altair Santin, and Dra. Cristine Hoepers for accepting to be member of the examination jury.

For all friends and professors which I had the pleasure to meet during this journey and somehow contributed directly or indirectly to complete this cycle, a huge thanks.

A special thanks to the Program of Electrical Engineering and the institutions COPPE and UFRJ for the opportunity to obtain a M.Sc. degree.

Lastly, I thank CNPq, CAPES, FAPERJ, and FAPESP (2015/24514-9, 2015/24485-9, and 2014/50937-1) for financing this work.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ENRIQUECIMENTO DE CARACTERÍSTICAS BASEADO EM GRAFOS PARA DETECÇÃO DE INTRUSÃO EM LINHA EM REDES VIRTUAIS

Igor Jochem Sanz

Dezembro/2018

Orientador: Otto Carlos Muniz Bandeira Duarte

Programa: Engenharia Elétrica

O crescente número de dispositivos IoT conectados contribui para a ocorrência de ataques distribuídos de negação de serviço a uma escala sem precedentes. A Teoria de Grafos, reforçada por técnicas de aprendizado de máquina, melhora a descoberta automática de padrões de comportamento de grupos de ameaças de rede, muitas vezes omitidas pelos sistemas tradicionais de segurança. Nesse sentido, a virtualização da função de rede é uma tecnologia emergente que pode acelerar o provisionamento de cadeias de funções de segurança sob demanda para uma aplicação. Portanto, a repetição de testes de conformidade e a comparação de desempenho de tais cadeias de funções são obrigatórios. As contribuições desta dissertação são separadas em duas partes. Primeiro, é proposto um sistema de detecção de intrusão que utiliza um enriquecimento baseado em grafos para aprimorar a detecção de ameaças online. Um algoritmo de enriquecimento de características é desenvolvido e avaliado através de diferentes técnicas de aprendizado de máquina. Os resultados mostram que o enriquecimento baseado em grafos melhora a acurácia da detecção de ameaças até 15,7 % e reduz significativamente o número de falsos positivos. Em seguida, para avaliar sistemas de detecção de intrusões implantados como funções virtuais de rede, este trabalho propõe e desenvolve o SFCPerf, um framework para avaliação automática de desempenho do encadeamento de funções de rede. Para demonstrar a funcionalidade do SFCPerf, é implementado e avaliado um protótipo de uma cadeia de funções de rede de segurança, composta por um sistema de detecção de intrusão (IDS) e um firewall sobre a plataforma aberta para virtualização de função de rede (OPNFV).

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

GRAPH-BASED FEATURE ENRICHMENT FOR ONLINE INTRUSION DETECTION IN VIRTUAL NETWORKS

Igor Jochem Sanz

December/2018

Advisor: Otto Carlos Muniz Bandeira Duarte

Department: Electrical Engineering

The increasing number of connected devices to provide the required ubiquitousness of Internet of Things paves the way for distributed network attacks at an unprecedented scale. Graph theory, strengthened by machine learning techniques, improves an automatic discovery of group behavior patterns of network threats often omitted by traditional security systems. Furthermore, Network Function Virtualization is an emergent technology that accelerates the provisioning of on-demand security function chains tailored to an application. Therefore, repeatable compliance tests and performance comparison of such function chains are mandatory. The contributions of this dissertation are divided in two parts. First, we propose an intrusion detection system for online threat detection enriched by a graph-learning analysis. We develop a feature enrichment algorithm that infers metrics from a graph analysis. By using different machine learning techniques, we evaluated our algorithm for three network traffic datasets. We show that the proposed graph-based enrichment improves the threat detection accuracy up to 15.7% and significantly reduces the false positives rate. Second, we aim to evaluate intrusion detection systems deployed as virtual network functions. Therefore, we propose and develop SFCPerf, a framework for an automatic performance evaluation of service function chaining. To demonstrate SFCPerf functionality, we design and implement a prototype of a security service function chain, composed of our intrusion detection system and a firewall. We show the results of a SFCPerf experiment that evaluates the chain prototype on top of the open platform for network function virtualization (OPNFV).

Contents

List of Figures

List of Tables

List of Abbreviations

API - Application Programming Interface

CAIDA - Center for Applied Internet Data Analysis

CAPES - *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*

CATRACA - sCAlable TRAffic Classifier and Analyzer

CLI - Command Line Interface

CNPq - *Conselho Nacional de Desenvolvimento Científico e Tecnológico*

COPPE - *Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia*

COTS - Commercial off-the-shelf

CPU - Central Processing Unit

DARPA - Defense Advanced Research Projects Agency

DDoS - Distributed Denial of Service

DNS - Domain Name Service

DPDK - Data Plane Development Kit

DPI - Deep Package Inspection

DoS - Denial of Service

ETSI - European Telecommunications Standards Institute

FN - False Negatives

FP - False Positives

FW - Firewall

GTA - *Grupo de Teleinformática e Automação*

HTTP - Hypertext Transfer Protocol

IDPS - Intrusion Detection and Prevention System

IDS - Intrusion Detection System

IETF - Internet Engineering Task Force

IP - Internet Protocol

IoT - Internet of Things

JSON - JavaScript Object Notation

MAC - Media Access Control

MANO - Management and Orchestration

ML - Machine Learning

NFVI - Network Function Virtualization Infrastructure

NFV - Network Function Virtualization

NSH - Network Service Header

NoSQL - Not Only Structured Query Language

OPNFV - Open Platform for Network Function Virtualization

OvS - Open vSwitch

PCAP - Packet Capture file extension

RAM - Random Access Memory

REST - Representational State Transfer

RFC - Request for Comments

RTT - Round-trip time

SDN - Software-defined Networking

SFC - Service Function Chaining

SFP - Service Function Path

SIEM - Security Information and Event Management

SQL - Structured Query Language

SVM - Support Vector Machine

TCP - Transmission Control Protocol

TN - True Negatives

TOSCA - Topology and Orchestration Specification for Cloud Applications

TP - True Positives

UDP - User Datagram Protocol

UML - Unified Modeling Language

VLAN - Virtual Local Area Network

VM - Virtual Machine

VNF - Virtual Network Function

VXLAN - Virtual Extensible Local Area Network

YAML - YAML Ain't Markup Language

Chapter 1

Introduction

Network attacks are one of the main threats in a fully connected world. The growing increase in Internet-connected devices brings a range of unknown vulnerabilities. With the advent of the Internet of Things (IoT), vulnerabilities exploits affect millions of devices simultaneously. Denial-of-service attacks and probe scans on connected devices are critical points for large-scale vulnerability exploitation and attack execution. More voluminous and with a large number of devices involved, network attacks reached the mark of ten Distributed Denial-of-Service (DDoS) attacks above 300 Gb/s in 2016 [1]. Furthermore, zombie networks composed of infected IoT devices were responsible for a DDoS rate higher than 1 Tb/s [2]. In 2018, a massive DDoS attack against Github[1], which exploited a vulnerability on *memcached* servers that reflect an amplified packet between 10.000 and 50.000 times its original size, reached the peak of 1.35 Tb/s [3]. A week after, from the same exploit vector, the DDoS peak record was set to 1.7 Tb/s against an U.S. service provider [4]. Late analysis of those events showed a massive probe scan targeting the *memcached* protocol a few days before the attack [5]. Therefore, security mechanisms that accurately detect and prevent attacks are necessary to the Future Internet. Detecting threats on execution time and promptly reacting to attacks are essential to reduce the impact of security threats [6]. Currently, security threat detection takes weeks or months, and it is expected to reduce this time to minutes or seconds [7]. The scenario for the future is even more adverse due to the introduction of more than 80 billion connected devices by 2025 on a Internet of Things (IoT) world [8]. IoT devices produce a huge volume of data, which need to be managed, processed, transferred and stored in a safe and fast manner. Current detection methods, however, are not designed to operate in such conditions [9]. Security systems, such as the Security Information and Event Management (SIEM), fail in presenting satisfactory performance, since 85% of network intrusions are detected weeks after they occur [10].

[1] Github is a web-based hosting service for version control using Git and the largest source code repository in the world.

Another crucial challenge for security systems is zero-day attacks, in which no prior knowledge of attacks is available. It is important that the detection time of zero-day attacks becomes the shortest possible to reduce reaction time and to be effective. One possible solution to accomplish this goal is if the threat detection process becomes completely automatic. Moreover, the virtualization technology, which is the foundation of cloud infrastructure providers such as Amazon, introduces new possibilities for attacks and poses new challenges for security systems that protect the cloud infrastructure [11]. In this complex and challenging scenario, a promising alternative to automatically detect threats and efficiently classify network traffic flows relies on the use of machine-learning techniques.

Machine learning provides to security systems the capacity of learning and improving from prior experience without being explicitly programmed for it. Machine-learning techniques benefit from the huge amount of data generated by Big Data sources, to infer hidden patterns which are extremely difficult to be inferred by humans [12]. In terms of security, machine-learning techniques allow the automation of threat signature generation, which is a key aspect for reducing the period of time zero-day threats remain effective [13]. Traditional machine-learning techniques, however, rely on classifying stored historical datasets, which restricts real-time response due to the high latency associated to the vast consumption of computational resources [14]. One approach to outcome this drawback is classifying flows as soon as they are generated. When analyzing traffic flows, simple packet filtering analysis based on TCP/IP headers are inefficient as attackers attempt to hide themselves from security tools by spoofing source IP addresses, dynamically changing TCP ports and constantly changing attack patterns. Moreover, network threats such as denial of service and port scans are disguised as benign traffic if the security system is designed to independently classify each flow. Solutions that can correlate different sources of flows and identify group behavior patterns of attacks in execution time are mandatory to efficiently detect threats. Machine-learning algorithms for online threat detection are becoming widely used by security systems as the high capacity of distributed processing from a cluster of machines, assisted by stream processing frameworks, allow the construction of agile and real-time algorithms to treat huge amount of data. Therefore, there is a need for machine-learning solutions that enhance security system detection capabilities adapted to online traffic classification.

Besides proposals that enhance detection capabilities, a fundamental aspect of maintaining the network secure is the placement of the security system. The possibility of deploying traffic monitors and intrusion detection systems anywhere in the network is important to reduce the zero-day detection time and to accelerate reaction to threats. Furthermore, multiple security systems allow to define secu-

rity levels and to enrich the historical information of attacks from different sources. The feasibility of this challenging task is acquiring maturity each year thanks to the advent of Network function virtualization (NFV) technology. NFV achieved notable prominence in telecommunications and security as it reduces hardware expenditures and network operational complexity. By deploying network services as software, NFV improves network flexibility and allow the development of optimized resource allocation schemes [15]. In this paradigm, network functions such as firewall and intrusion detection systems migrate from dedicated hardware middle-boxes to software functions executed as virtual machines on top of commercial off-the-shelf (COTS) hardware. Therefore, network services can be extended and controlled in a centralized manner, dynamically migrated and deployed in the network, and tailored for each application. Concerning security features, the NFV flexibility allows deploying new policies, promoting fast updates, defining security zones, steering traffic and isolating compromised network components [16–19]. Furthermore, service function chaining (SFC) is a key enabler for flexible traffic management of a service or application [20]. When deploying a network security function as part of a chain of VNFs, the high latency or incorrect ordering of the VNFs imply failure of packet handling policies, increase of vulnerabilities or occurrence of security incidents [21]. Therefore, performing repeatable and comparable experiments through an infrastructure-agnostic framework is essential to identify and avoid performance bottlenecks on NFV and SFC platforms, as well as to correctly define resource constraints [22].

This manuscript proposes a machine-learning solution for network threat detection adapted to real-time security systems and evaluates its implementation on a NFV scenario. Therefore, we organize our proposal in two parts. First, we propose a feature enrichment algorithm that applies concepts of Graph Theory to online intrusion detection systems. Our algorithm represents a group of network flows comprised in a time window as a graph to infer characteristics based on complex networks. The algorithm infers different metrics from the snapshots of time windows, separated in three classes: vertex metrics, edge metrics and component metrics. The algorithm is evaluated under different sets of extracted features and machine-learning techniques, including cases preceded by pre-processing methods such as feature selection and reduction. We study the detection performance improvements as inferred metrics are incorporated as new features to the original set of features from the TCP/IP header. The proposed enrichment method is evaluated for three traffic datasets: a real dataset from a Brazilian telecommunication operator, a synthetic dataset constructed at GTA/UFRJ lab, and a publicly available and realistic botnet dataset. Results show improvements on the detection capabilities of distributed denial-of-service and probe threats, as well as of botnet traces, without compromising the

online detection. In addition, feature selection and reduction techniques can be applied to reduce the amount of processing load without significantly impacting the detection accuracy. For most analyzed scenarios, the evaluation demonstrates an increase in classification accuracy when our enrichment algorithm precedes classification. It is important to note that our algorithm is not restricted to the scenarios and features obtained, hence it can be extended to different set of features and machine-learning techniques available in the literature. Then, we design an architecture that incorporates the online enrichment process for online intrusion detection systems.

In the second part of this proposal, we propose SFCPerf, a framework to automate the performance evaluation of virtual network functions, such as virtual IDS and virtual firewall, deployed over different scenarios and conditions. Not only restricted to security functions, SFCPerf is a framework for automating experimentation of service function chaining. The framework generalizes the automation for any virtual network function and service function chain orchestrated in a NFV environment. The main goal of SFCPerf is to provide repeatability to experimentation through the definition of a testing workflow. Thus, results obtained by the framework allow comparison to any other service function chain configuration, as the scenario and the experiments are strictly defined by a workflow description file. The SFCPerf workflow is divided into three phases: setup phase, experimental phase, and post-experiment phase. SFCPerf automates environment creation and network configuration during the setup phase. Then, during the experimental phase, SFCPerf configures data measurement, performs data collection and controls the experiment. In the post-experiment phase, the acquired data is pre-processed and sent to preliminary analysis. To demonstrate the functionality of our framework in a real use case, we develop and evaluate the performance of a security service prototype based on service function chaining. The prototype is composed of two security network functions: an intrusion detection system based on machine-learning techniques and stream processing; and an adjustable firewall with a RESTful interface. We build our prototype on top of the European Telecommunications Standards Institute (ETSI) NFV MANO architecture [23]. In addition, we analyze network function chaining in compliance with RFC 7665 [24], provided by the Internet Engineering Task Force (IETF). The prototype meets the specifications of Network Service Header (NSH) for the SFC encapsulation. Furthermore, we use SFCPerf to evaluate the performance of VNF chaining over different topologies and the current development level of NSH to identify major bottlenecks. We adopt the Open Platform for Network Function Virtualization (OPNFV) as an NFV infrastructure for our evaluation experiments. Results from NFV experimentation show that chaining multiple functions incurs a throughput decrease and a linear end-to-end delay increase, which are independent

of the deployed topology.

1.1 Contributions and Publications

The main contributions of this work are summarized as following.

- An algorithm for online enrichment of machine-learning features based on a graph-based approach.
- An architecture for online intrusion detection systems that incorporates the online enrichment process
- A framework for automating the performance evaluation of Service Function Chaining.
- The identification of major bottlenecks of VNF deployment in a NFV-SFC environment using the OPNFV platform
- A proof-of-concept and prototype for an intelligent security chain composed of a VNF IDS with a VNF firewall. The combination of both VNF provides the automatic reaction to threats when a threat is detected by IDS, i.e., automatic and real-time insertion of firewall rules and malicious flow blocking.

The following publications are highlighted as direct contributions from this work.

- Sanz, I. J., Alvarenga, I. D., Andreoni Lopez, M. E., Mauricio, L. A. F., Mattos, D. M. F., Rubistein, M. G. and Duarte, O. C. M. B. - "Uma Avaliação de Desempenho de Segurança Definida por Software através de Cadeias de Funções de Rede", in Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'2017 [11].
- Sanz, I. J., Mattos, D. M. F., and Duarte, O. C. M. B. - "SFCPerf: An Automatic Performance Evaluation Framework for Service Function Chaining", in IEEE/IFIP Network Operations and Management Symposium - NOMS'2018 [25].
- Sanz, I. J., Andreoni Lopez, M., Rebello, G. A. F., and Duarte, O. C. M. B. - "Um Sistema de Detecção de Ameaças Distribuídas de Rede baseado em Aprendizagem por Grafos", in Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2018 [26].
- Sanz, I. J., Rebello, G. A. F., and Duarte, O. C. M. B. - "GRAFFITO-IDS: A Graph-based Algorithm for Feature Enrichment on Online Intrusion Detection Systems", submitted to Computers & Security.

Moreover, the following publications were derived as indirect results from this research.

- Sanz, I. J., Andreoni Lopez, M., Mattos, D. M. F., and Duarte, O. C. M. B. - "A Cooperation-Aware Virtual Network Function for Proactive Detection of Distributed Port Scanning", in 1st Cyber Security in Networking Conference IEEE - CSNet'2017. [19]

- Andreoni Lopez, M., Silva, S. R., Alvarenga, D. I., Rebello, G. A. F., Sanz, I. J., Lobato, G. P. A., Mattos, D. M. F., Duarte, O. C. M. B., Pujolle, G. - "Collecting and Characterizing a Real Broadband Access Network Traffic Dataset", in 1st Cyber Security in Networking Conference IEEE - CSNet'2017. (Best paper award) [27]

- Andreoni Lopez, M., Sanz, I. J., Mattos, D. M. F., Duarte, O. C. M. B and Pujolle G. - "CATRACA: uma Ferramenta para Classificação e Análise de Tráfego Escalável Baseada em Processamento por Fluxo", in Salão de Ferramentas do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'2017. (Best tool award) [28]

- Rebello, G. A. F., Alvarenga, I. D., Sanz, I. J., and Duarte, O. C. M. B. - "SINFONIA: Gerenciamento Seguro de Funções Virtualizadas de Rede através de Corrente de Blocos", in WBlockchain'2018: Workshop em Blockchain: Teoria, Tecnologias e Aplicações - SBRC'2018. (Best paper award) [29]

- Andreoni Lopez, M., Sanz, I. J., Lobato, A. Mattos, D. M. F., and Duarte, O. C. M. B. - "Aprendizado de Máquina em Plataformas de Processamento Distribuído de Fluxo: Análise e Detecção de Ameaças em Tempo Real", in Minicursos do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2018. [30]

- Lobato, A. P., Andreoni Lopez, M., Sanz, I. J., Cárdenas, A. A., Duarte, O. C. M. B., Pujolle, G. - "An Adaptive Real-Time Architecture for Zero-Day Threat Detection" - in IEEE International Conference on Communications - ICC'2018. [31]

1.2 Organization

The remainder of this work is organized as follows. Chapter 2 discusses related work. Chapter 3 presents the graph-based enrichment proposal, the performance evaluation results and its discussion. Chapter 4 presents the intrusion detection system prototype for a Network Function Virtualization scenario, proposes SFCPerf

framework and discusses results obtained from SFCPerf evaluation. Chapter 5 concludes this manuscript and provides directions for future work.

Chapter 2

Related Work

This chapter provides a summary on the state-of-art of graph theory approaches applied to intrusion detection systems, as well as the security concern in the network function virtualization field. We highlight the main differences of the contributions of this manuscript to related work available in the literature.

2.1 Graph Theory applied to Intrusion Detection

Considering graph-theory approaches for threat classification, [32] proposed GrIDS that is a graph-based intrusion detection system, which models a computer environment as an activity graph. The system generates a directed graph and associates extra information from events as attributes for the edges, vertices, and the global graph, which shows effectiveness in tracking the propagation of a worm over different hosts. The authors, however, do not focus on the online detection of a threat, instead, they depend on the report of suspicious connections from multiple hosts. This dependence implies that multiple hosts need to be previously infected to generate a worm propagation alert, which do not prevent such attacks. Liu *et al.* proposed an approach for detecting threats on HTTP communication using graph-based techniques to analyze data [33]. The authors restrict their proposal for the HTTP protocol and for the identification of malicious ISP clients. Alternatively, our proposal relies on network threat identification by combining TCP/IP stack features with features generated through graph analysis. Iliofotou *et al.* proposed traffic dispersion graphs as a network monitoring tool [34]. The proposed tool, names Graption, infers the nature of applications and identifies patterns in the network by combining information from a network-wide behavior with flow-level characteristics [35]. A key difference of their work to our proposal is that authors focus in classifying network traffic among different applications. In contrast, our proposal aims to detect network threats that are not necessarily associated to a specific application.

Concerning anomaly detection using graph modeling, Eswaran *et al.* proposed

an algorithm that analyzes the sudden appearance or disappearance of large dense directed subgraphs to detect anomalies in IP-IP communication, such as port scan and denial-of-service attacks [36]. Many researches propose graph clustering and partitioning techniques to leverage the efficiency and reduce the processing load of graph analysis for large-scale datasets [37–39]. Anomaly-detection proposals often abstract graph streaming problems, such as the replacement of the dynamism of the Internet data traffic into consecutive static graph snapshots [40, 41]. While all the above proposals focus on anomaly detection, which identifies deviation patterns from benign behavior, our system focuses in the feature enrichment for classification techniques. The combination of graph theory with machine-learning-based approaches are also found in the literature to reduce false-positives [42, 43], and by extracting graph features to perform online learning to detect botnets [44, 45].

In our security scenario, we aim to classify network flows between normal or potentially malicious, henceforth defined as threat. Thus, to investigate the performance of our graph-based enrichment proposal, this work focuses on the evaluation of supervised learning techniques for threat detection. We employ three supervised techniques frequently used in the intrusion detection literature, decision trees, naive Bayes, neural networks [46, 47], and we include the comparison of an ensemble learning of trees. Different set of features can be considered when a real problem is modeled as a machine-learning problem. The choice for specific features vary according to the level of abstraction in which the threat is modeled. In the intrusion detection literature, researchers parse the network traffic into 5-tuple flows to detect application threats [48, 49], between 2-tuple of IP addresses to identify network threats [31], or into group of packets defined by the IP source address to identify malicious hosts [50, 51]. In our work, we aim to detect network threats that can be performed in a distributed manner without the need for deep inspecting the payload data. Thus, we aggregate 5-tuples with the same source IP and destination IP addresses and define a flow at the network layer, also called network flow. Considering the network flow abstraction, a 26-feature set based on TCP/IP header information is proposed, when capturing raw network traffic between IP-IP communication [31]. In our work, we adopt the 26-feature set as baseline for the enrichment process, with slight adaptations and enhancements, such as the addition of four new features, which is fully described in Section 3.2.1. Furthermore, the enrichment increases this number by 39 new features inferred through the graph analysis. Unlike the aforementioned work, our work proposes a graph-based enrichment to support machine-learning techniques for online intrusion detection systems. It is important to note that the enrichment proposal is independent of the employed machine-learning technique, the selected feature set, or the flow abstraction, and is unrestricted to the ones chosen to evaluate the proposal.

2.2 Network Function Virtualization and Security

Intrusion detection systems for cloud computing became a necessary defense for threat detection in virtualized environments. These systems can benefit from emerging technologies to enhance detection capabilities, such as the flexibility of NFV and the global view of the network from SDN. The deployment of intrusion detection systems as virtual network functions gained attention of the community and different techniques were proposed to aid detection performance [52, 53]. Besides, another key aspect of deploying security systems as virtual network functions is the impact on the network performance. Concerning this aspect, the second part of this work focuses on assessing the performance of security systems deployed as virtual network functions and the impact on the network for different scenarios of NFV deployment.

Different network function virtualization architectures have been proposed with their own service function chaining approaches [54–59]. Likewise, middle-box chaining are performed by using software-defined networking techniques. FlowTags is a SFC proposal for middle-boxes that is capable of tagging headers and passing context information to the subsequent middle-box while enforcing traffic policies [54]. The StEERING proposal uses multiple tables, presented in the OpenFlow 1.0 standard, instead of adding tags [55]. StEERING creates hierarchical forwarding rules while adding metadata to the packet handling on each step and defining the next hop in the chain.

Most proposals consider an infrastructure that combines software-defined networking with network function virtualization [56, 57]. The ESCAPE tool is built upon network function virtualization standardized by ETSI [56]. The key idea is to use ClickOS as the basis for the development of virtual network function prototypes [58]. On the other hand, Cloud4NFV presents an architecture for network function virtualization based on four planes: infrastructure, virtual infrastructure management, orchestration, and service [57]. Though ESCAPE and Cloud4NFV are closely related to ETSI architecture, they do not comply with the IETF service function chaining [24].

The NetBricks proposal develops network-function packet forwarding with zero copy [59]. Hence, the chaining of network functions uses shared memory with reference passing in the memory area. NetBricks brings considerable gains in performance of virtual functions in terms of bandwidth and latency. Performance of virtual functions chaining is also evaluated analytically and is considered as a constraint for optimization problems, such as the placement of VNFs over the physical infrastructure [60]. Lopez *et al.*, in turn, argue that the location of virtual functions on the physical infrastructure follows a trade-off between accepting a larger number

of VNFs and the delay of chaining on more distant physical nodes [61].

Emmerich *et al.* evaluate the performance of virtual switches during VNF chaining. The authors conclude that optimization in core operating system configurations and dedicated CPU utilization for the network interfaces are essential to increase the performance of virtual switches [62]. Callegati *et al.* present a performance comparison of network virtualization and the main components of the OpenStack cloud operating system [63]. Bonafiglia *et al.* compare the performance of different network function virtualization technologies [64]. They consider configurations of virtual switches with and without Data Plane Development Kit (DPDK). The authors also compare the performance of network chaining executed in virtual machines versus Docker[1] containers. The results show the performance achieved by virtual machines is superior to the performance of Docker containers when using switches with DPDK support and dedicated processing cores.

Although most works focus on evaluating performance of NFV and SFC on a given scenario, there is a need for solutions to evaluate performance of NFV use cases, regarding the interoperability problem of the early stage of NFV [65]. In this sense, DETER is a testbed proposal for network testing and monitoring that focuses on security experiments [66]. Similarly, RIO is an experimentation platform to emulate denial-of-service (DoS) attacks using NFV technology [67]. The objective is to investigate DoS attack patterns and potential mitigation mechanisms. RIO also automates the configuration and setup of network elements and proposes a language to describe a given test scenario. Riggio *et al.* propose Scylla, a descriptive language to describe virtual network functions orchestration regardless the underlying infrastructure [68].

Unlike all aforementioned works, in this manuscript we propose SFCPerf, a framework for automating the experimentation of performance evaluation of service function chaining. The framework aims to be used for performance comparison of service function chains composed of virtual network functions from different manufacturers and running on distinct NFV-SFC platforms. To demonstrate SFCPerf functionality, we study NFV scenarios that comply with the Network Service Header (NSH) protocol to chain network functions [69] and to the most mature standard for SFC architecture [24]. Finally, we implement a prototype of our IDS as a virtual network function. We evaluate our VNF under SFCPerf framework to compare to other VNF performances and NFV scenarios.

[1] Available at `https://www.docker.com/`.

Chapter 3

Graph-based Feature Enrichment for Online Intrusion Detection Systems

This chapter presents and evaluates a graph-based enrichment proposal to enhance online intrusion detection systems based in machine learning. First, we detail all modules of the architecture for intrusion detection system (IDS) that incorporates our proposal. Then, the algorithm for graph-based feature enrichment is presented. Moreover, we evaluate the enrichment for different scenarios, including different network traffic datasets and machine-learning classification algorithms. Finally, we evaluate the case when the classification is preceded by feature selection methods.

3.1 The Proposed System Architecture

Five modules compose our proposed intrusion detection system architecture: data capture module, enrichment module, processing module, historical database, and visualization module. On the data capture module, distributed sensors collect data over the network, while all other modules run in a cluster for distributed data processing. Figure 3.1 depicts the proposed system architecture.

The data capture module consists of tools for monitoring and capturing data traffic packets executed on distributed sensors on the network. The sensors are instantiated as virtual machines in virtualized environments or through physical machines with traffic mirroring from a network link. First, during the online traffic capture, the data capture module abstracts captured packets into flows identified by the quintuples (source IP, source port, destination IP, destination port, transport protocol). Second, the data capture module groups flow quintuples with the same source and destination IP addresses, abstracting in IP–IP flows to detect network

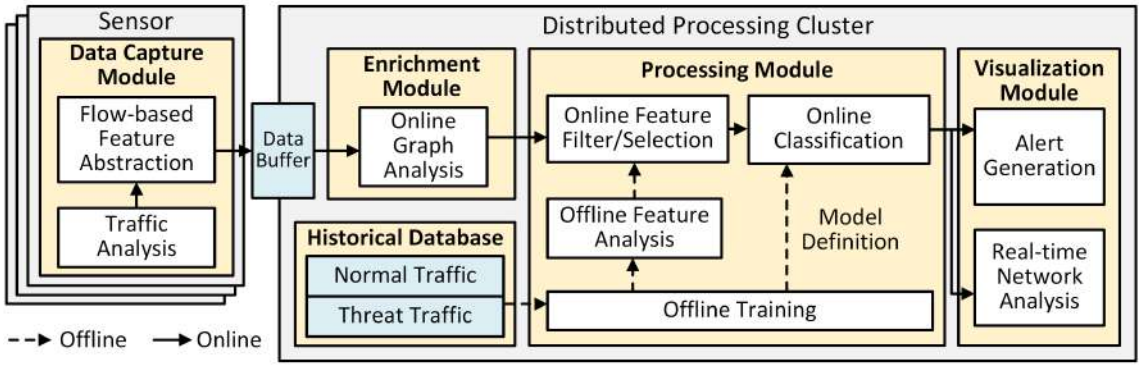


Figure 3.1: Architecture of the proposed classification system. The architecture is divided into five modules, data collection module, enrichment module, processing module, visualization module and historical database.

threats, such as port scans and denial-of-service attacks. This process leads to the extraction of 26 numerical features from the packet header and a new network-layer flow defined by two IP addresses. A detailed description of these features is presented in Section 3.2.1. The system publishes the abstracted features in a queue manager before deploying in a distributed processing cluster. This system receives data from multiple sensors placed in different locations in the network. Then, a data buffer stores the collected features to be requested by the enrichment module.

The online enrichment module is the main contribution of this work. The enrichment process performs a static graph analysis of the set of instances, i.e original feature vectors, in a flow time window and infers a new graph-based feature vector. The graph-based analysis detects coordinated and distributed threats from group behavior analysis, which are impossible to be detected by traditional techniques individually on each flow. We detail the process of enrichment, the algorithm, and all new extracted features in Section 3.2.

The processing module executes the detection algorithms and classifies the enriched samples. Our system also employs filter methods for features selection to the input sample to reduce the complexity of the online processing. The filter is defined towards predefined feature selection and dimensionality algorithms, executed through offline and batch processing. The online classification is performed through different machine-learning algorithms, sample per sample, implemented through distributed processing frameworks. The learning model is generated during the offline training phase from a labeled dataset stored in the historical database. To evaluate the performance of this proposal, we use three different datasets for the historical database, a synthetic dataset constructed in our lab, a real traffic from a Brazilian network operator and realistic and publicly available traces of botnet.

Lastly, the visualization module comprises an interface between the system and

the user through the Internet. This module generates traffic alarms for suspicious activities and a detailed analysis delivered in real time.

3.2 The Proposed Graph-based Online Enrichment

The main idea of our proposed online enrichment is to model a graph-based structure from samples retrieved from a time window. In essence, a graph is an ordered pair $G = (E, V)$ composed of a set of V vertices and E edges, which each edge is necessary associated to two vertices. To generate graphs from a group of flow samples, we use the concept of snapshots. We define a snapshot as a static set of samples collected during a time window of captured packets. The snapshot model of a directed graph considers the IP addresses as vertices and the data transmitted between two IP addresses as directed edges. The direction of the edge is defined as the source pointing to the destination of the flow. It is important to note that distributed network attacks, such as port scan and DDoS, have malicious characteristics that can be dissimulated when flows are individually analyzed. Thus, the graph-based enrichment aims to infer patterns from the group behavior of such attacks. Furthermore, port scan and DDoS network attacks have the particular characteristic that all malicious IP addresses involved are interconnected in the same graph component, i.e., vertices connected through a finite distance, frequently interconnected through the victim node. Figure 3.2 depicts an example of graph generated from a snapshot of a 2-second time window.

In this example of snapshot, which comprises 30 components, there are two threats highlighted with a circle, a distributed port scan originated from ten different malicious hosts and a distributed denial-of-service attack. The bigger component that comprises most of the snapshot vertices contains only benign nodes in this example, however, it may include malicious nodes altogether with non-malicious nodes. Therefore, characterizing only the component that comprises the vertices is not enough to detect threats and identify malicious hosts. Thus, we aim to characterize the attack components but also the vertex and edge behavior by generating features that are potentially correlated to a malicious activity.

In the proposed graph model, a vector of features of the TCP/IP header is assigned to each edge. To detect distributed threats, TCP/IP features that are related to the occurrence of these threats are selected as weights to generate new graph features. Algorithm 1 describes the extraction and enrichment process.

We consider as input the IP-IP flows abstracted from network traffic and defined in a time window. The first step is to create a directed and weighted graph model in

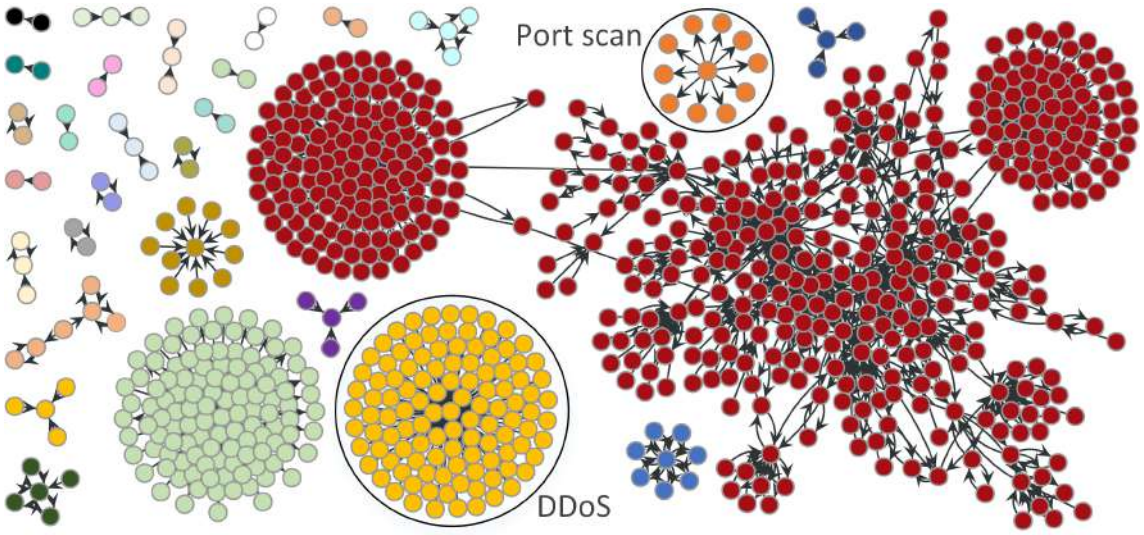


Figure 3.2: Graph of a snapshot from a 2-second time window containing a DDoS and a port scan. The graph is separated into 30 connected components, i.e., subgraphs with no finite distance among them.

Algorithm 1: Feature enrichment based in graph analysis of a set of instances comprised in a static snapshot of a time window.

Input : X: Matrix of the original feature set
Output: Y: Matrix of the enriched feature set

$G = constructGraph(X)$
$Components = extractComponents(G)$
foreach $Subgraph \in Components$ **do**
  $LocalFeatures = extractLocalMetrics(Subgraph)$
  foreach $Vertex \in Subgraph$ **do**
    $VertexFeatures = extractVertexMetrics(Vertex)$
  end
  foreach $Edge \in Subgraph$ **do**
    $EdgeFeatures = extractEdgeMetrics(Edge)$
  end
end
foreach $Edge \in G$ **do**
  $V1 = Edge.Source$
  $V2 = Edge.Destination$
  $Y[Edge] = X[Edge] + LocalFeatures +$
  $VertexFeatures(V1) + VertexFeatures(V2) + EdgeFeatures[Edge]$
end

which a vertex V represents an IP address and an edge E represents an IP-IP flow with an initial feature vector as weight. A sample with the initial 26-feature vector is represented as $[IP_{src}, IP_{dst}, feat_1, feat_2..., feat_{26}]$. When a set of samples of a time window arrives, a graph $G(E, V)$ of the sample set is constructed. Then, the algorithm divides the global graph into small connected components C, through the

function *extractComponents()*, which has a complexity of $O(V + E)$, defining all the connected components comprehended in G. Processing each extracted component individually allows to perform a graph analysis locally in each component, circumventing a centralized processing load and allowing the distribution and parallelism of the extraction tasks. The graph analysis infers new features based on graph metrics. Each evaluated metric becomes a new feature. The graph metrics are classified in three categories: i) local metrics; ii) vertex metrics; and iii) edge metrics. For each component C, the algorithm extracts local metrics from the component, such as the total number of vertices, total number of edges, total number of bytes transmitted or even the total number of distinct TCP ports occurred in that component. Then, the algorithm infers the metrics from vertices, such as the input or output degree, which can be weighted or non-weighted. It is important to note that any feature from TCP/IP header can be assigned to the edge weight to calculate a weighted degree, which expands the range of possible features. Lastly, the algorithm generates metrics from edges, such as the amount of flows, bytes and packets transmitted in those edges compared to the totals of the component. Complex features can also be inferred from the graph, such as centrality measures of vertices and edges. For instance, the coefficient of a betweenness centrality measure can be incorporated to the enriched feature vector as an edge metric. After the feature extraction process from all three categories, each initial flow sample E is enriched with the local features from the component C, in which the flow is part of, the features from both vertices V_{src} V_{dst} that compose the flow edge, and the features from the edge E which represents the flow. The final enriched sample set Y is sum of original sample set X with all features generated during the enriched process. It is important to note that a fourth category of features could be used in our algorithm, composed of metrics from the global graph in the entire time window. This category endorses metrics, such as total number of vertices or edges or the total bytes transmitted during that period of time, which can produce useful information for machine-learning techniques regarding the volume of traffic captured in the network at the moment of a threat.

3.2.1 Features

In our modelling problem, we aim to detect network threats without reading the application content of the packet and, therefore, our focus is not on application layer attacks. Hence, network packets are abstracted into IP-IP flows at the network layer, in which each flow is defined as a sequence of packets from a source IP address to the same destination IP during a time window. Thus, we define an input sample, to be classified between normal and threat, as an IP-IP flow comprising the initial 26

features inferred directly from the network traffic. Table 3.1 details all 26 features used as baseline for the enrichment evaluation.

Table 3.1: Original features abstracted from the packet header of IP-IP network flows.

#	**Label**	**Description**
1	n_pkt_tcp	Nb. of TCP packets
2	n_src_port	Nb. of distinct source ports
3	n_dst_port	Nb. of distinct destination ports
4	n_fin_flag	Nb. of packets with FIN flag set
5	n_syn_flag	Nb. of packets with SYN flag set
6	n_psh_flag	Nb. of packets with PUSH flag set
7	n_ack_flag	Nb. of packets with ACK flag set
8	n_urg_flag	Nb. of packets with URG flag set
9	n_pkt_udp	Nb. of UDP packets
10	n_pkt_icmp	Nb. of ICMP packets
11	n_pkt_ip	Nb. of IP packets
12	n_tos	Nb. of types of service (ToS)
13	mn_ttl	Mean time to leave (TTL)
14	mn_head_len	Mean length of packet header
15	mn_pkt_len	Mean length of packet
16	n_do_not_frag	Nb. of packets with *Don't fragment*
17	n_more_frag	Nb. of fragmented packets
18	n_rst_flag	Nb. of packets with RST flag set
19	n_ece_flag	Nb. of packets with ECE flag set
20	n_cwr_flag	Nb. of packets with CWR flag set
21	n_types_icmp	Nb. of distinct ICMP types
22	n_codes_icmp	Nb. of distinct ICMP codes
23	n_flows	Nb. of 5-tuples
24	n_flows_tcp	Nb. of TCP 5-tuples
25	n_flows_udp	Nb. of UDP 5-tuples
26	n_bytes_flow	Nb. of total bytes transmitted

The 26-feature vector contains numerical variables representing information gathered from all generated 5-tuples and transmitted packets between two IP addresses, for instance, the mean or variance of the quantity of flags from TCP/IP header. Thus, in our study case, each IP-IP flow has the following 26 initial features: number of each TCP flags (8); number of packets TCP, UDP, ICMP and IP (4); number of destination and source ports (2); mean size of packet header and content (2); number of fragmented packets and do-not-fragment flags (2); number of distinct ICMP types and codes (2); number of distinct service types (1); TTL mean (1); number of bytes transmitted (1); number of established TCP connections (1); number of unique UDP transmissions (1); and number of unique flow quintuples (1) between the two IP addresses that defines the flow.

Once we generate the graph model for each snapshot in a time window, as detailed in Algorithm 1, the 26-feature vector from each sample is enriched with 39 metrics inferred for the graph analysis. The new features are divided in three categories and, in our study case, represents the following metrics: a) 7 local metrics: total number of vertices (distinct IP addresses) and edges (IP-IP flows) of the component (2), total number of bytes, flows and packets transmitted in the component (3), total number of distinct destination and source ports occurred in the component (2); b) 4 edge metrics: fraction of bytes, flows and packets transmitted in the IP-IP flow in comparison to the total transmitted in the component (3), betweenness centrality [70] of the edge (1); and c) 14 vertex metrics: simple input and output degree (2), input and output degree of TCP, UDP, ICMP and IP packets (8), and input and output degree of source and destination ports from the source and destination vertices of the edge (4). Since one edge is strictly defined by two vertices, the initial feature vector is enriched with vertex metrics from both source and destination vertices of the edge. When considering the enrichment with generated local and edge metrics, this process produces 39 new features for each edge and a resultant feature vector composed of 65 features. Table 3.2 details all features obtained in the enrichment process.

3.3 Experimental Setup and Proposal Evaluation

To evaluate the performance of the proposed approach for intrusion detection, we perform experiments using three different datasets. The first dataset is a synthetic traffic elaborated in our lab, Grupo de Telefinformática e Automação (GTA/UFRJ), composed of real normal user behavior from desktop applications and threats executed in a controlled way. We introduce threats through the Kali linux distribution and the Nmap tool[1], comprising 36 types of threats divided into three categories: a) 7 types of DoS: ICMP flood, land, nestea, punk, smurf, SYN flood, and UDP flood; b) 8 types of DDoS: spoofed and non-spoofed SYN flood, teardrop, smurf, and nestea; and c) 20 types of port scans, scan of FIN, SYN, XMAS, NULL and ACK flags, executed in a horizontal and vertical manner, and in a distributed and non-distributed way [11]. This process yields a dataset composed of 19,149 network flows.

The second dataset is a real data traffic collected from a major Brazilian telecommunications operator [71]. The dataset contains real fixed Asymmetric Digital Subscriber Line (ADSL) access information of 373 residential broadband users from the city of Rio de Janeiro during a period of one week, from February 27th to March 5th 2017. For privacy concerns, the data is anonymized. We decapsulate the Point-

[1]Available at `https://nmap.org/`.

Table 3.2: Set of features obtained from the graph-based feature enrichment algorithm.

No.	Type	Description
27	Component	Number of vertices of component
28	Component	Number of edges of component
29	Component	Total bytes of component
30	Component	Total flows of component
31	Component	Total packets of component
32	Component	Total source ports
33	Component	Total destination ports
34	Edge	Fraction of bytes of the component
35	Edge	Fraction of flows of the component
36	Edge	Fraction of packets of the component
37	Edge	Edge betweenness
38	Src. Vertex	In degree
39	Src. Vertex	Out degree
40	Dst. Vertex	In degree
41	Dst. Vertex	Out degree
42	Src. Vertex	In degree weighted by source ports
43	Src. Vertex	Out degree weighted by source ports
44	Dst. Vertex	In degree weighted by source ports
45	Dst. Vertex	Out degree weighted by source ports
46	Src. Vertex	In degree weighted by destination ports
47	Src. Vertex	Out degree weighted by destination ports
48	Dst. Vertex	In degree weighted by destination ports
49	Dst. Vertex	Out degree weighted by destination ports
50	Src. Vertex	In degree weighted by TCP packets
51	Src. Vertex	Out degree weighted by TCP packets
52	Dst. Vertex	In degree weighted by TCP packets
53	Dst. Vertex	Out degree weighted by TCP packets
54	Src. Vertex	In degree weighted by UDP packets
55	Src. Vertex	Out degree weighted by UDP packets
56	Dst. Vertex	In degree weighted by UDP packets
57	Dst. Vertex	Out degree weighted by UDP packets
58	Src. Vertex	In degree weighted by ICMP packets
59	Src. Vertex	Out degree weighted by ICMP packets
60	Dst. Vertex	In degree weighted by ICMP packets
61	Dst. Vertex	Out degree weighted by ICMP packets
62	Src. Vertex	In degree weighted by IP packets
63	Src. Vertex	Out degree weighted by IP packets
64	Dst. Vertex	In degree weighted by IP packets
65	Dst. Vertex	Out degree weighted by IP packets

to-Point Protocol over Ethernet (PPPoE) sessions of the ADSL residential clients

with the Stripe tool[2]. Since it is not possible to assure that real traffic is benign or malicious, we apply the signature-based IDS Suricata[3] to classify normal traffic and different type of detected threats. Finally, we insert the 36 types of attacks described above, merging the IPs of the synthetic malicious traffic with the real IPs from operator residential users. For the sake of evaluation fairness, we balance the dataset as 50% malicious and 50% benign traffic, by randomly filtering flows from the dominant class after the enrichment process. This process yields a balanced dataset composed of 715,181 network flows.

The third dataset is the ISCX botnet dataset[4], the most realistic and publicly available dataset of botnet traffic [72]. We use the training data, which contains 5.26 GB of benign and botnet traffic from 7 different types of botnets, Neris, Rbot, Virut, NSIS, SMTP Spam, Zeus, and Zeus C&C (command and control). We discard IPV6 traffic and use a 2-second time-window to generate the final dataset. This process results in a dataset composed of 273,797 network flows, in which 20,1% represents malicious botnet traffic.

For the threat classification, we select three of the most common classifiers with large utilization and well known behavior in the literature of intrusion detection [46, 73, 74]: decision tree, neural network, and naive Bayes. The decision tree, despite having a high cost of model construction due to the complexity of creating well-adjusted models, in general, presents better results over the classification performance metrics and also create an easily-understandable model. The cost for decision tree updates is high because slight data changes may result in a completely different tree. Neural networks often represent an accurate black box where neuron performs a activation function over the data and neuron weights are adjusted to minimize the error. In case of data arriving as streams, neural network model adjustment is feasible using stochastic gradient descent. Neural networks, however, are slower for training due to iterative process of weight adjustments. The naive Bayes, differently, uses a probabilistic model to estimate the efficiency of simple classifiers with low processing load. Naive Bayes classifiers are capable of quick training the classification model, and therefore, present a faster reaction for detecting new threats as the model constantly needs updates.

We employ the decision tree algorithm (DT), which is a supervised technique, to construct a tree model that each leaf node is responsible for testing a particular attribute of the system. Figure 3.3 illustrates the generation of a tree-based model from a set of instances defined by network-based features. The model construction, i.e., the training phase, occurs in an offline manner and we store the probabilities

[2]Available at `https://github.com/theclam/stripe`.
[3]Available at `https://suricata-ids.org/`.
[4]Available at `https://www.unb.ca/cic/datasets/`.

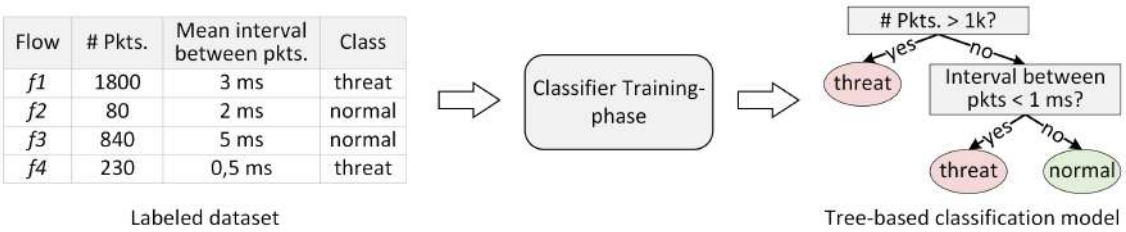


Figure 3.3: The decision tree learning technique, which generates a tree-based model from a set of training instances.

values of each class in the leaf nodes to serve as parameters for the decision-making. During the training phase, on each new input sample, the algorithm runs through the nodes evaluating the respective attributes to estimate the probability of that sample belonging to a given class. Thus, it is not always necessary to run through all nodes to perform the classification, which considerably reduces the processing and classification time compared to other models. We use the C4.5 algorithm for tree construction with the information gain ratio for tree splitting with no prepruning [75]. The tree construction starts from the definition of a root node, which has the probability of each class. Then, the root node is successively split, in which each child represents a new sample attribute that has a new set of probabilities for each class. This process is repeated for all nodes until the nodes reach probabilities of 100% for one class, configuring it as a leaf node. We repeat this process ten times with different fractions of the datasets complying with the 10-fold cross-validation.

The second employed algorithm is the multilayer perceptron (MLP), which uses a back-propagation training method to generate the coefficients of a neural network and adjust the weight of each neuron according to the gradient of an arbitrary loss function. Figure 3.4 shows the contents of a neural network model.

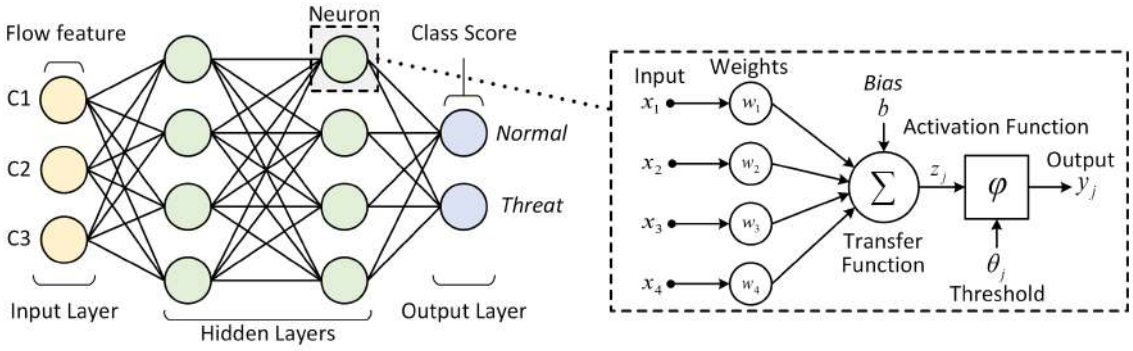


Figure 3.4: The multilayer perceptron technique, which generates a neural-network based model that contains the appropriate neuron weights adjusted according to a set of training instances.

We adopted as loss function the average of the Euclidean distance between the real and the predicted output expressed by

$$E = \frac{1}{2n} \sum_{\vec{x}} ||y(\vec{x}) - y'(\vec{x})||^2, \tag{3.1}$$

where n is the number of inputs for the training dataset, $\vec{x}$ is the evaluated input, and $y(\vec{x})$ and $y'(\vec{x})$ are the real and predicted output values, respectively. The weight adjustment of a neuron j from the layer i is given by the gradient descent method

$$\Delta w_{ij} = -\eta \frac{\partial \vec{E}}{\partial w_{ij}}, \tag{3.2}$$

where η is a parameter for model adjustment that controls the velocity of weight updates. The errors E_i from each layer are propagated and the neuron weights are adjusted according to the partial derivative related to each neuron. The back-propagation allows the separation of classifications that are not linearly or trivially divisible, which makes it suitable to adjust complex curves for the model. We use a MLP model with an input layer where each neuron represents a feature of the problem, an output layer with two neurons representing the output classes, threat and normal, and one hidden layer composed of 10 neurons. The hidden layer associates the output data from the previous layer to an activation function for each neuron and produces the output data for the next layer.

The last employed algorithm is the naive Bayes (NB), which assumes the strong premise that all features from the system are independent, i.e., one feature does not influence other feature values. This premise reduces the complexity of the problem and simplifies the prediction model. When a new sample $\vec{x}$ arrives, the algorithm calculates for each feature the *a priori* probability for it to belong to each class. From Bayes Theorem, the *a posteriori* probability of a feature vector $\vec{x}$ belonging to the class C_k is

$$P[C_k|\vec{x}] = \frac{P[\vec{x}|C_k] * P[C_k]}{P[\vec{x}]}, \tag{3.3}$$

where $P[\vec{x}|C_k]$ is the conditional probability *a priori* for feature vector $\vec{x}$ if class C_k is given, $P[C]$ the probability for a sample belonging to class C since class probability distribution is known from the training set, and $P[\vec{x}]$ a normalization constant for each sample. The objective is to maximize the numerator to find the class that better fits into the set of features of the unknown sample. Due to the independence assumption, the product of all probabilities *a priori* for a given class results in the probability *a posteriori* of the sample belonging to that class. Hence, the algorithm chooses for the class with higher estimated probability $P[C_k, \vec{x}]$ among all classes

given by

$$P[C_k|\vec{x}] = \prod_{i=1}^{n} P[x_i|C_k]. \tag{3.4}$$

After the enrichment process, the feature vector comprehends 65 features, which is relatively high to obtain online and also increases the chances for over-fitting the data. Therefore, we evaluate two algorithms that aim to reduce the number of features: a feature selection algorithm and a dimensionality reduction algorithm. The feature selection algorithm reduces the number of characteristics without significant information loss. In practice, it is a three-phase process: i) to normalize the dataset; ii) to eliminate features with zero variance; and iii) to calculate the correlation matrix among the remainder variables. Since all variables considered in this manuscript are numerical, we used the Pearson correlation, defined as

$$\rho_{X,Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}, \tag{3.5}$$

where σ_X and σ_Y are the standard deviation of each variable and μ_X and μ_Y their respective mean to define the value of correlation between each pair of variables. Figures 3.5, 3.6, and 3.7 show the correlation matrix for each dataset. Evaluating the most correlated features allows a quick assessment to features that do not aggregate useful information and can be dismissed, reducing the classifier processing load without loss of classification performance. Greater correlation between two features indicates that is highly probable that increasing the value of one of them will also increase the other. In the other hand, negative correlation indicates that one feature will likely increase if the other decreases and vice versa. Values near zero represents no relation between them.

We use Principal Component Analysis (PCA) to reduce the number of feature, finding orthogonal combinations s_i of input features that maximize the total variance of the projected data. The input features p are a linear combination of the k principal components and the direction that maximizes the variance while also minimizing the mean squared error. Therefore, the PCA not only reduces the dimensionality but it eradicates the redundancy caused by the correlation between features x_i. In this manuscript, we define the criteria for dimensionality reduction by specifying the minimal amount of information to be preserved as 100% for each feature. Then, by using the most important components without loss of information, we build a good approximation from the original data reducing the number of features from 65 to 51 in the network operator dataset, to 46 in the GTA/UFRJ dataset and to 27 in the ISCX botnet dataset. We define these set of features as PCA-reduction set for further evaluation.

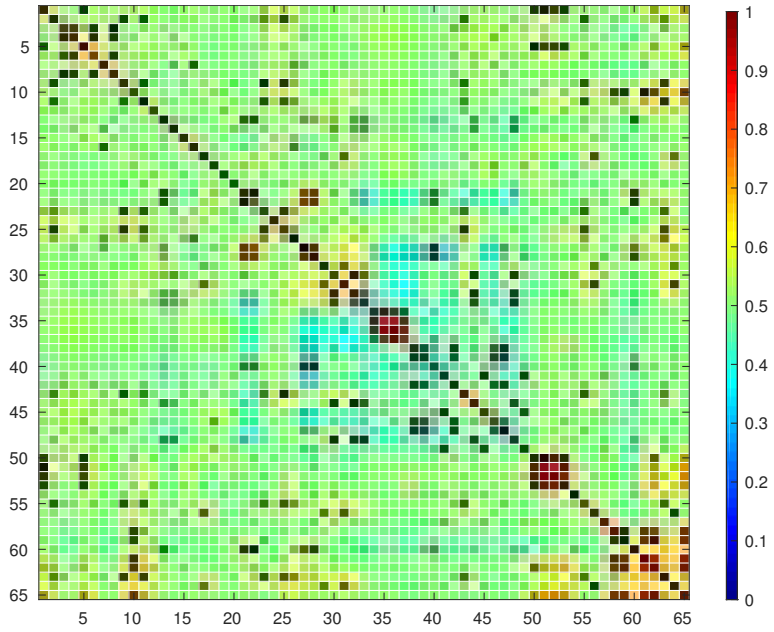


Figure 3.5: Correlation matrix of all features used in the Network Operator dataset. Features from 1 to 26 are the initial features inferred from the packet headers of IP-IP flow, and features 37 to 65 are obtained through the graph-based enrichment.

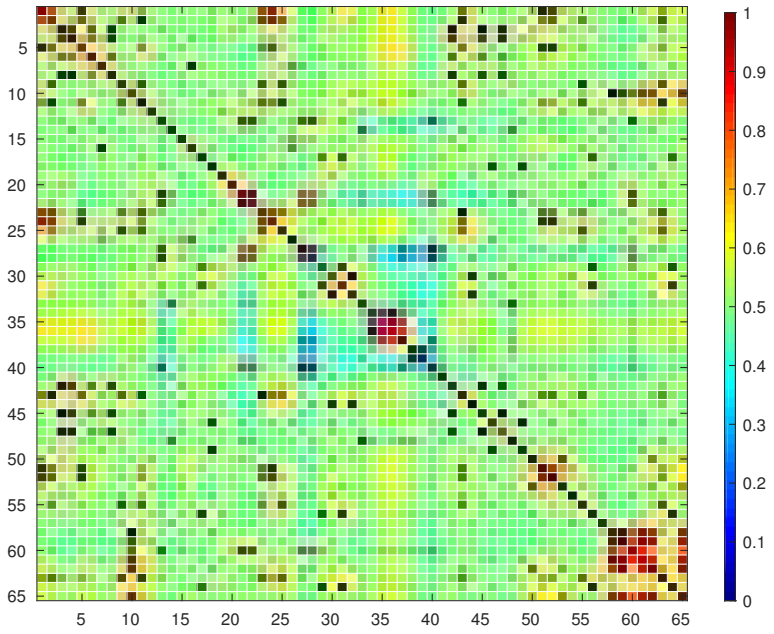


Figure 3.6: Correlation matrix of all features used in the GTA/Lab dataset. Features from 1 to 26 are the initial features inferred from the packet headers of IP-IP flow, and features 37 to 65 are obtained through the graph-based enrichment.

To evaluate the performance of our classifier for the various datasets, detection algorithms and feature sets, we select standard metrics for comparison fairness. Table 3.3 shows the metrics derived from the confusion matrix, where positive in-

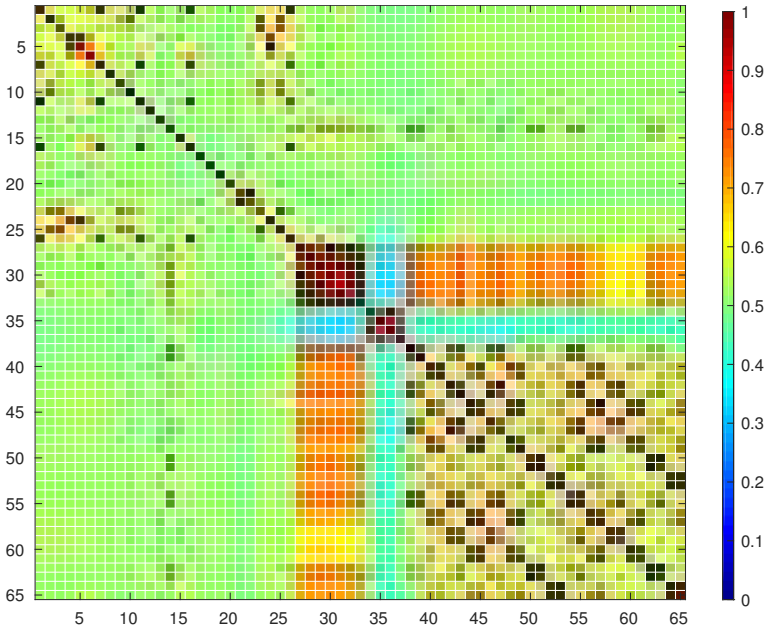


Figure 3.7: Correlation matrix of all features used in the ISCX botnet dataset. Features from 1 to 26 are the initial features inferred from the packet headers of IP-IP flow, and features 37 to 65 are obtained through the graph-based enrichment.

stances, i.e., feature vectors, represent benign or normal flows, and negative instances represent malicious or threat flows.

Table 3.3: The employed metrics for performance evaluation.

Metric	Description
False Positives (FP)	Number of threat flows misclassified.
False Negatives (FN)	Number of normal flows misclassified.
True Positives (TP)	Number of threat flows classified correctly.
True Negatives (TN)	The number of normal flows that are correctly classified.
True positive rate ($TPR = \frac{TP}{TP+FN}$)	Proportion of normal flows classified correctly.
False positive rate ($FPR = \frac{FP}{FP+TN}$)	Proportion of threat flows incorrectly predicted as normal.
Accuracy $= \frac{TP+TN}{TP+FP+FN+TN}$	The ratio of correctly classified flows to the total flows.
Area under the ROC curve $AUC = \int_{\infty}^{-\infty} TPR(\gamma)FPR'(\gamma)d\gamma$	The ROC curve represents graphically the trade-off between the FPR and TPR for every possible γ that defines the detection cut-off. The AUC quantifies the ROC curve for numerically comparison.

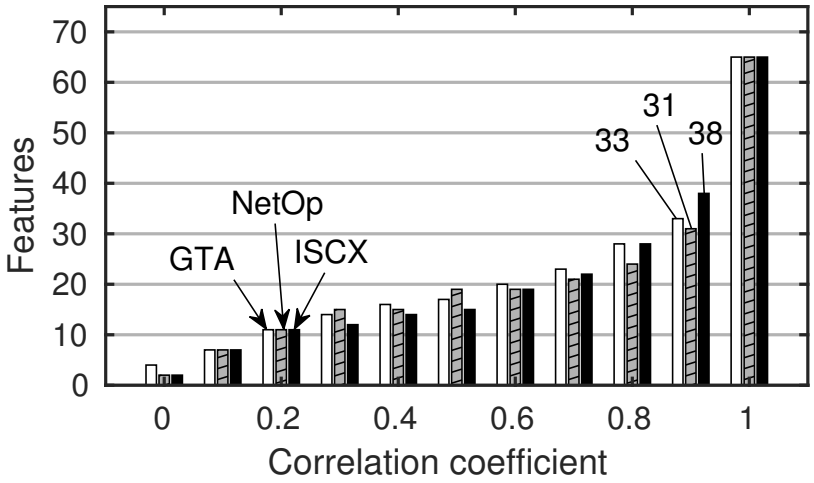


Figure 3.8: Number of features after the implementation of a linear correlation filter for all GTA, NetOp and ISCX datasets.

3.4 Numerical Results

We implement a prototype and evaluate the gain introduced by our proposal into classification performance. We use an Intel Xeon E5-2650 @ 2.00GHz 16-core (32) with 512 GB memory and 21 TB hard disk. We employ a two-second time window due to a better trade-off between time consumed and high accuracy obtained with machine-learning offline algorithms [31]. We define five sets of features for evaluation: i) the original set of 26 features from the TCP/IP header inferred online directly from the data traffic abstraction into IP-IP network flows; ii) the set of 39 features inferred from the online enrichment from the graph-based analysis; iii) the enriched set, which merges both aforementioned feature sets together, totaling 65 features; iv) the set of features obtained through the dimensionality reduction from PCA method, which resulted in 51 features for the operator dataset, 46 features for the GTA/UFRJ dataset, and 27 for the ISCX botnet dataset; and v) the set of features obtained after the application of linear correlation filter.

To select a parameter for the linear correlation filter, we evaluate the system performance when we apply a correlation filter prior to the classification. Figure 3.8 illustrates the amount of features that are eliminated when we decrease the correlation coefficient ρ. By varying the correlation parameter ρ, we set the threshold $\rho = 0.9$, as it already reduces by half the number of features to be processed by only filtering the features near 100% correlated to others. Using this parameter selection, we attain a set composed of 32 features for the operator dataset, 34 features for the GTA/UFRJ dataset, and 38 for the ISCX botnet dataset.

We define different set of features to classify both datasets with the three pre-

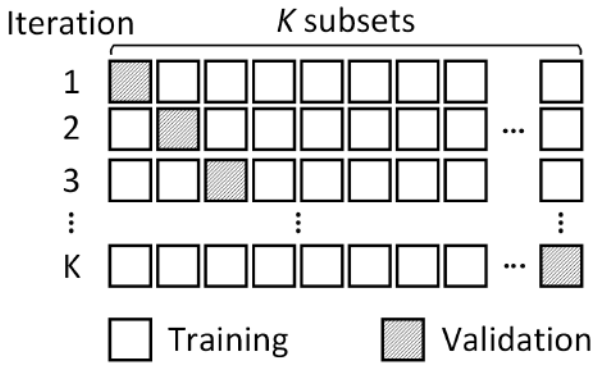


Figure 3.9: The k-fold cross-validation technique, which splits the dataset into k mutually exclusive parts.

sented classification methods: decision tree, naive Bayes, and multilayer perceptron. We perform the training phase for all algorithms using a k-fold, with $k = 10$ cross-validation to better generalize the model and avoid over-fitting. Figure 3.9 illustrates this technique, which consists of successively separating the data set in 10 random partitions mutually exclusive, in which 9 partitions are separated for the training phase and the remainder for the online test phase in each iteration. Thus, after 10 iterations, the mean of all training phases are considered as the final model, which grants more confidence on the performance results.

Furthermore, we balance the classes of both datasets using a Gaussian sampling filter on the normal traffic, obtaining a final dataset composed of 50% of normal traffic and 50% of malicious traffic, in order to avoid a biased classification towards the dominant class.

We consider accuracy and the area under the receiver operating characteristic curve (AUC) as our classifier performance estimators. Tables 3.4, 3.5 and 3.6 show the results of the evaluated scenarios for all datasets. The results show that our feature enrichment proposal improves both accuracy and AUC on the TCP/IP feature set for all algorithms using the NetOp dataset, and for two algorithms on the GTA/UFRJ and the ISCX dataset.

The accuracy for the naive Bayes algorithm in the ISCX dataset, however, declined significantly. This is due to the overall high correlation between the graph features for ISCX, found in Figure 3.7, which contradicts the feature independence premise assumed by the algorithm.

Furthermore, when using a feature set composed of only the 39 graph-based features, the classification presents higher accuracy than if using the complete enriched set of 65 features. This behavior suggests that, in this case, the inclusion of particular TCP/IP features introduces noise information into the classification. Moreover, we observe that the graph-based enrichment produces minor changes on

the accuracy of the threat detection for the decision tree algorithm because it already presents an accuracy near to 100% with the 26 original features TCP/IP.

Table 3.4: Classification accuracy and area under the ROC curve (AUC) for the network operator dataset.

	Decision Tree		Naive Bayes		ML Perceptron	
Feature set (# feat.)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)
TCP/IP features (26)	99.95	99.99	81.46	84.10	99.12	99.03
Graph features (39)	99.96	99.99	95.38	99.65	99.96	99.98
Enriched features (65)	99.98	99.99	94.61	99.88	99.99	99.99
PCA reduction (51)	99.96	99.99	96.03	99.84	99.19	99.71
Linear corr. filter (31)	99.94	99.99	94.60	99.86	99.99	99.99

Table 3.5: Classification accuracy and area under the ROC curve (AUC) for the GTA/UFRJ dataset.

	Decision Tree		Naive Bayes		ML Perceptron	
Feature set (# feat.)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)
TCP/IP features (26)	99.98	99.97	75.13	86.65	99.29	99.80
Graph features (39)	99.96	99.97	82.60	97.67	99.70	99.86
Enriched features (65)	99.96	99.97	80.32	98.30	99.96	99.98
PCA reduction (46)	99.96	99.94	94.65	96.47	96.53	97.52
Linear corr. filter (33)	99.96	99.98	90.24	98.93	99.95	99.87

Table 3.6: Classification accuracy and area under the ROC curve (AUC) for the ISCX dataset.

	Decision Tree		Naive Bayes		ML Perceptron	
Feature set (# feat.)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)	Acc. (%)	AUC (%)
TCP/IP features (26)	99.03	99.08	85.39	89.74	95.17	96.63
Graph features (39)	99.92	99.91	82.61	84.91	98.98	99.65
Enriched features (65)	99.79	99.77	84.19	89.12	98.88	99.61
PCA reduction (27)	99.77	99.78	79.30	93.36	93.06	98.41
Linear corr. filter (38)	99.69	99.56	79.97	81.93	98.34	99.36

Considering all evaluated algorithms, naive Bayes presents the higher accuracy gain after the enrichment process, 15.7% for the operator dataset and 9.9% for the GTA/UFRJ dataset. Furthermore, when we introduce a linear correlation filter, we reduce by half the number of features to be processed while the accuracy remains stable. The exception for this behavior is the simple Bayesian algorithm in the GTA/UFRJ dataset, which presented a 12% gain after the introduction of a linear correlation filter. Therefore, the results indicate that the enrichment process adds valuable information to the detection and is more effective for algorithms with simpler training, such as the naive Bayes.

Table 3.7 shows the full classification statistics for the three datasets before and after the enrichment process. We also present the difference and the ratio from

Table 3.7: The confusion matrix containing the full statistics of all 9 enriched scenarios.

		DT				NB				MLP			
		TP	FP	TN	FN	TP	FP	TN	FN	TP	FP	TN	FN
NetOp	Ori.	357570	83	357507	20	225071	316	357274	132519	351869	301	357289	5721
	Enr.	357585	0	357590	5	316902	360	357230	40688	357586	62	357528	4
	Diff.	15	-83	83	-15	91831	44	-44	-91831	5717	-239	239	-5717
	Ratio	1.00	0.00	1.00	0.25	1.41	1.14	1.00	0.31	1.02	0.21	1.00	0.00
GTA/Lab	Ori.	9572	0	9574	2	7161	2348	7226	2413	9505	66	9508	69
	Enr.	9570	2	9572	4	5944	138	9436	3630	9571	3	9571	3
	Diff.	-2	2	-2	2	66	-63	63	-66	66	-63	63	-66
	Ratio	1.00	-	1.00	2.00	0.83	0.06	1.31	1.50	1.01	0.05	1.01	0.04
ISCX	Ori.	216138	1598	55015	1046	185444	8275	48338	31740	211319	7359	49254	5865
	Enr.	216933	333	56280	251	198410	24511	32102	18774	216155	2035	54578	1029
	Diff.	795	-1265	1265	-795	12966	16236	-16236	-12966	4836	-5324	5324	-4836
	Ratio	1.00	0.21	1.02	0.24	1.07	2.96	0.66	0.59	1.02	0.28	1.11	0.18

Table 3.8: True positive rate and false positive rate comparison for all 9 enriched scenarios.

		DT		NB		MLP	
		TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)
NetOp	Ori.	99.99	0.02	62.94	0.09	98.40	0.08
	Enr.	100.00	0.00	88.62	0.10	100.00	0.02
GTA/Lab	Ori.	99.98	0.00	74.80	24.52	99.28	0.69
	Enr.	99.96	0.02	62.08	1.44	99.97	0.03
ISCX	Ori.	99.52	2.82	85.39	14.62	97.30	13.00
	Enr.	99.88	0.59	91.36	43.30	99.53	3.59

the enriched (E) set of features to the original (O) set, respectively. These metrics serve as indicators for the overall performance improvement. We aim to reduce false positives (FP) and false negatives (FN), then, we expect negative values for the differential $E - O$ metric or values above 1.0 for the E/O ratio. In seven of nine scenarios, the enrichment process reduced misclassified samples, except for the decision tree algorithm in the GTA/Lab dataset and for the naive Bayes algorithm in the ISCX dataset. The decision tree case is easily explained, since the original set of features already presented a classification accuracy near 100%, it prevented any type of performance improvement. The naive Bayes algorithm, in the other hand, assumes strong independence among variables and from Figure 3.7, we observe that variables from graph analysis were strongly correlated in the ISCX botnet dataset. Then, we conclude that Naive Bayes is not a good approach for botnet detection regarding graph-based features.

Overall, in 5 of the 7 improved scenarios, the gain in the classification performance occurred with no trade-off. The two exceptions for this rule are the naive Bayes algorithm for the network operator dataset, which significantly reduces the number of false negatives by a small increase in the number of false positives, and for the GTA/LAB dataset, which increases the number of false negative but reduces false positives twice more. To compare the relative gain, after the enriched process,

Table 3.8 shows the true positive rate (TPR) and false positive rate (FPR) for all 9 scenarios. Despite decision tree algorithm presents the best overall classification performance among all algorithms, the higher performance gain occurred for the naive Bayes algorithm in the network operator dataset, which increased the true positive rate on 25.7% while the false positive rate remained stable. Finally, we highlight the significant gain in the classification performance achieved for the MLP algorithm over all evaluated datasets. The enrichment process using the MLP algorithm reduced the number of false negatives for the real traffic of operator dataset by 1430 times, by 22 times for the GTA/UFRJ dataset, and by 5.69 times for the ISCX botnet dataset, while the number of false positive is significantly reduced in all cases. Indeed, concerning botnet traffic detection, the enrichment with MLP algorithm reduce the false positive rate in 9.4%.

Despite an overall gain on the majority of the evaluated scenarios, we highlight some of the limitations faced during the experiments. First, the generation of a realistic dataset is challenging, and there is still much effort to be done in order to generate an up-to-date network threat dataset. The procedure of the synthetic dataset generation aimed to use as many available attacks as possible at the moment of the dataset construction. Therefore, some of the attacks included in our dataset may not represent a current threat in the real world scenario. For instance, the smurf Denial-of-service is a well-known attack that is inoffensive for modern systems nowadays. Indeed, due to the lack of availability of recent attacks in open source penetration tools, results obtained might not represent the reality in current network threat scenario. To overcome this limitation, a possible solution is to build our own botnet based on the source code of recent botnets and malware, e.g. Mirai source code[5]. Second, in the real dataset, which we use real data from a network operator to evaluate our proposal, we can never assure the real label of the network data. To overcome this, we make a strong assumption that an open source IDS correctly labels normal traffic as benign to conduct our experiments. Finally, we emphasize that the experiments performed in this chapter aim to prove the concept of a graph-based enrichment for Intrusion Detection Systems and to evaluate the performance gain when we implement our proposal to detect network threats. Our proposal, however, is not restricted to the attacks and datasets evaluated and the enrichment algorithm is feasible to be implemented in other network security scenarios.

[5]Available at `https://github.com/jgamblin/Mirai-Source-Code`.

Chapter 4

Intrusion Detection Systems in Virtual Networks

This chapter introduces the basic concepts of network function virtualization (NFV) and demonstrates how an intrusion detection system could benefit from this technology to acquire flexibility on deployment and to enhance security capabilities in the cloud. It also evaluates a prototype of a service function chaining (SFC) composed of an intrusion detection system and a firewall. For the evaluation, we develop a framework for automated performance evaluation of SFC named SFCPerf. Therefore, this chapter also presents SFCperf architecture, and provide the evaluation of some use case scenarios of NFV to identify major bottlenecks in this paradigm.

4.1 Network Function Virtualization

In the softwarization era, virtual networks became the foundation of the service operation and management across the network infrastructure. Virtual networks use network elements as software to virtualize applications and facilitate their migration with less service degradation over the Internet. Virtual networks paved the way for emerging cloud architectures. To this end, the network function virtualization (NFV) is an emergent technology which applies cloud computing concepts into the domain of telecommunications operator [20]. The main benefits of NFV are to reduce capital and operational expenditures, since hardware network functions are expensive, need physical space allocation, require proper cooling and high power consumption, and demand human resource training. The NFV also accelerates the time-to-market for network function since its conception till the delivery to the network operator. To reach similar performance of the hardware middleboxes, recent advances in cloud computing, as multiple hypervisors, hardware-assisted virtualization, cloud operating systems, containerization, and efficient software switches have

contributed to advances in the software implementation of network functions. This implementation is called virtual network function (VNF). Software-defining networking (SDN), in addition, decouples the control plane from the underlying data plane and consolidates control functions in a logically centralized controller [76].

NFV and SDN are complementary technologies, once the network function management benefits from the logically centralized controller to configure data plane, to chain coherent network functions, and even to balance the network load among several virtual machine clusters [77]. Figure 4.1 shows how the network function virtualization infrastructure (NFVI), according to the architecture of management and orchestration of NFV (NFV-MANO) [23], can compose end-to-end microservices tailored for each application. The NFVI provides the necessary abstractions for processing, storage and networking to the virtual network functions. Furthermore, packet forwarding and the consequent infrastructure abstraction into a graph service function chains can be performed in a flexible manner by the SDN controller [78].

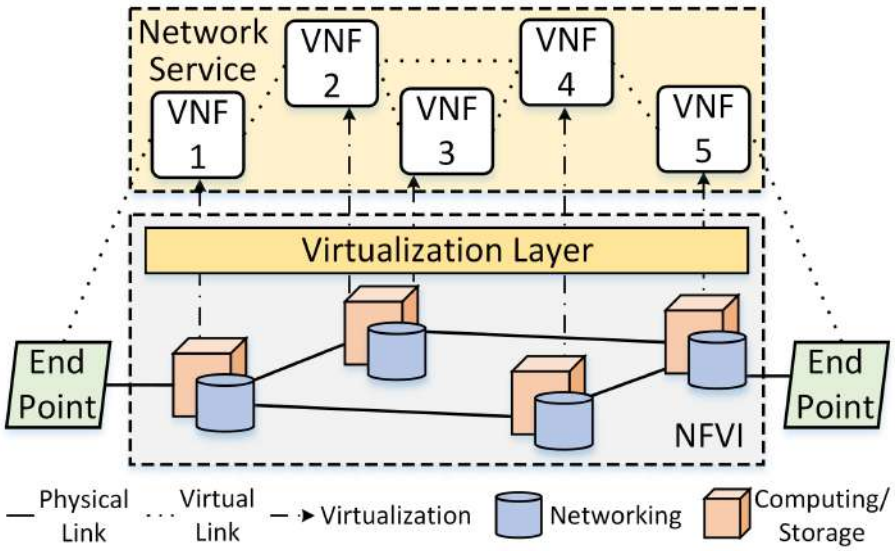


Figure 4.1: The network function virtualization infrastructure, which allows the creation of virtual links for end-to-end microservices through packet forwarding among virtual network functions (VNF).

For the sake of simplicity, we define both network function and service function as synonyms in this manuscript. Some researchers, however, consider a service function as a composition of one or more chained network functions [79].

4.2 Virtual Network Function Chaining

A key aspect of service function chaining (SFC), also referred as network function chaining (NFC), is to extend the functionality of the network service by chaining

specific functions into the network layer. Virtual network functions (VNF) are virtual machines that perform functions on the network layer to replace the numerous hardware-specific middleboxes that currently exist. Examples of such virtual functions are intrusion detection systems, firewall, proxy, NAT, switches etc.

The basic architecture of service function chaining, according to RFC 7665 [24], is shown in Figure 4.2. Considering a multi-service environment, a classifier is configured with specific rules to identify and specify the chain of VNFs that each service flow must traverse. The chain is defined by a ordered and logical sequence of VNFs called service function path (SFP). Hence, flows from different micro-services can simultaneously traverse the same VNF, but each coursing its own service function path. Virtual network functions may also be hosted on different physical nodes. Therefore, the service function forwarder (SFF) is a mandatory element in each network function virtualization infrastructure (NFVI) node to provide the virtual links to its hosted VNFs.

In the SFC architecture, isolation between micro-service flows in the same VNF is performed through packet encapsulation. VNFs may be aware or unaware of the SFC encapsulation. VNFs that are unaware of encapsulation require a precedent element to decapsulate SFC packets, called SFC Proxy, whereas SFC-aware VNFs need either a kernel module or a software switch compatible with the SFC encapsulation. The most consolidated proposal of encapsulation that meets the specification of SFC architecture is the network service header (NSH) [69]. NSH performs correct packet forwarding to the next VNF and isolation between micro-service flows thanks to two main fields, the service index (SI) and the service path identifier (SPI). The SI is an 8-bit index representing the relative position of the current VNF in the chain. The SPI is a 24-bit identifier associated to a specific service function path described in a software switch. All forwarding decisions on packets tagged with NSH header are defined by rules in the software switch and pro-actively orchestrated by the SDN controller, without resulting in delay for the packet forwarding. Other benefit that network operators can achieve from NSH encapsulation is the possibility to exchange meta-data between VNFs using the context fields. This aspect is positive for the case of mobile Internet which may contain an user identifier to execute tailored contract-based policies [80].

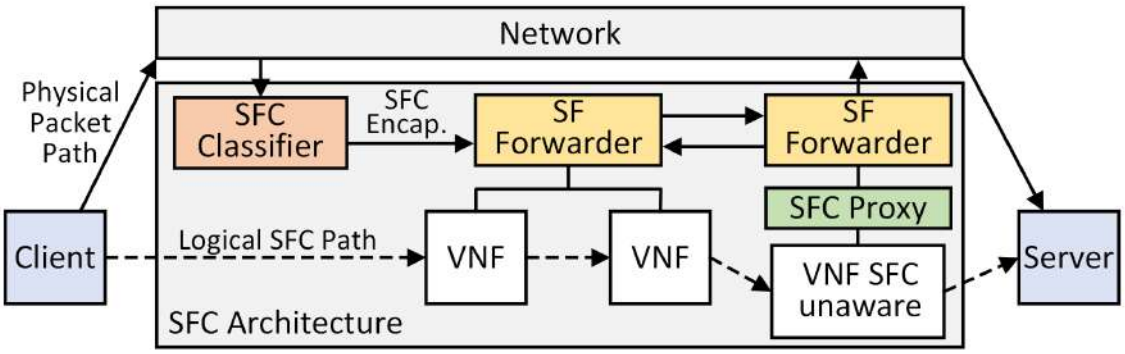


Figure 4.2: Basic elements of the service function chaining architecture. The SFC classifier encapsulates the incoming traffic and the SF forwarder redirects the traffic to the correct chain based on the encapsulation headers. An SFC proxy enables the implementation of functions unaware of the SFC encapsulation.

4.3 SFCPerf: An Automatic Performance Evaluation Framework for Service Function Chaining

Different open source NFV platforms are currently being proposed, such as OPNFV [81], OpenMANO [82], ClickOS [58]. There are also other approaches for service function chaining, such as container-based chaining [83], NetBricks [59], Dysco [84], etc. We modularly design the SFCPerf framework to allow automated SFC testing, agnostic to the underlying NFV-SFC infrastructure. Our testing framework provides experiment repeatability for all scenarios, which is essential to compare different approaches for VNF. When testing different SFC infrastructure providers, we are able to perform exactly the same experiment over the different infrastructures and, thus, we assure repeatability required to compare the performance. The SFCPerf framework is illustrated in Figure 4.3 and comprises the following main components: control, management, driver, passive and active data collection, analysis, and visualization modules.

Control module is the main component of the architecture. It configures and coordinates the other SFCPerf modules. The central goals of the control module are to handle all configuration parameters setup and calibration knobs, to check the experiment health, and to deliver the testing results to user. The control module receives a unique document as input data containing a detailed JSON description of the experiment. The document contains all parameters necessary to perform an automated testing of a service function chain. The control module also assigns the correct driver regarding compatibility into different NFV infrastructures.

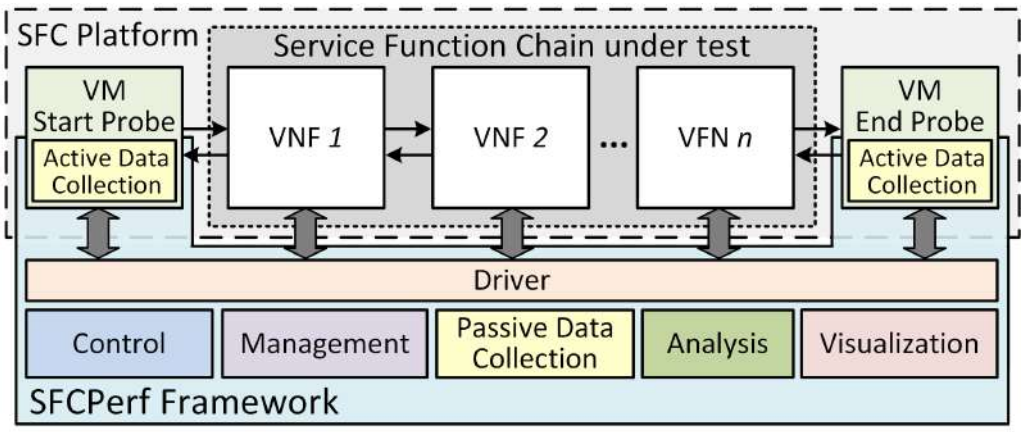


Figure 4.3: The proposed SFCPerf framework. The driver module provides an adaptive interface between the framework and different SFC platforms. The control module coordinates other modules to provide the environment setup, data collection and data delivery to user.

Management module governs the NFV-SFC platform and is responsible for orchestrating operations, such as instantiating and resizing VMs or VNFs. These operations are instructions to the management module of the NFV-SFC platform intermediated by the driver module. The management module also manages sensing tools in the NFV-SFC platform by coordinating the passive data collection module. The main difference between management and control modules is that the management module handles NFV-SFC related operations, while the control handles the experiment workflow.

Passive data collection module senses physical and virtual resource usage when performing experimental evaluations of VNFs. The data include information from telemetry modules of the NFV platform, such as disk, processing and memory usage of virtual resources, and from a set of data gathering applications executed on top of VNFs, e.g., tpcdump, top, iotop, free, etc.

Active data collection module consists in a set of probe applications, such as iperf, ping and httperf that actively measures the network performance. The applications are hosted in the endpoint VMs and generate traffic that traverses the chain of VNFs. Thus, traffic is processed in the other termination to obtain the evaluation metric.

Analysis module performs operations over experimental data, such as data preprocessing, merging, correlating, and enriching. The module allows to combine multiple sources of data in a customized manner and to enrich data with additional information, such as geo-tagging.

Visualization module provides a user-readable interface for data visualization. The module plots graphs and compares experimentation approaches. We conceive

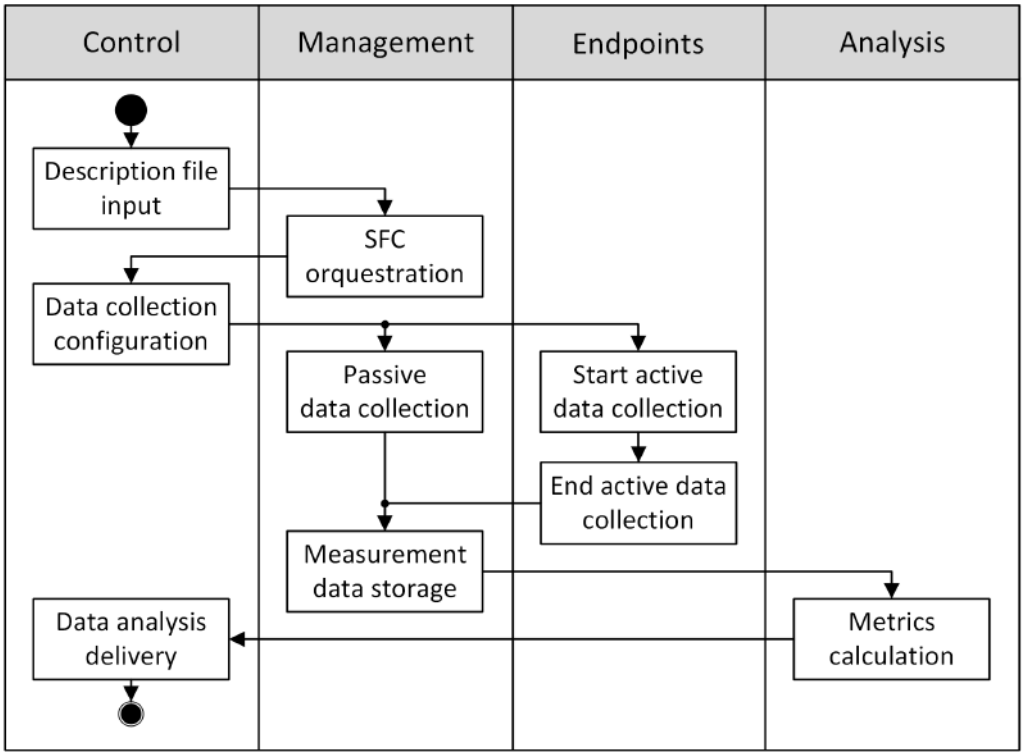


Figure 4.4: UML activity diagram of the proposed SFCPerf framework. The framework user interacts with the control module, which sets up the environment, controls the experimentation, and delivers the obtained data.

the visualization module with Kibana[1], a third-party data visualization application which is a component of the Elastic stack. Kibana implements a web interface that allows the visualization of data and the design of new observation scenarios in a simple and fast way. Moreover, Kibana benefits from a high-performance execution of queries over large volumes of data to provide visualization with low-latency.

Driver module is an essential component to provide compatibility of the framework to any NFV-SFC infrastructure. It provides an RPC interface between framework modules and orchestration components of the NFVI. In our prototype, we use OPNFV as NFVI platform, thus, the driver sends requests directly to Openstack, OpenDaylight, and Tacker APIs. For OPNFV, we implement this communication via RESTful HTTP requests.

The execution of an experiment over SFCPerf is exemplified on the activity diagram, Figure 4.4. We highlight the relationship among modules and their functions. First, the framework user loads the workflow description file into the control module, which parses and interprets the test description and calls the appropriate functions on the management module. All modules deal with the SFC platform through the driver module, omitted from Figure 4.4 for the sake of simplicity. The control mod-

[1]Available at `https://www.elastic.co/products/kibana`.

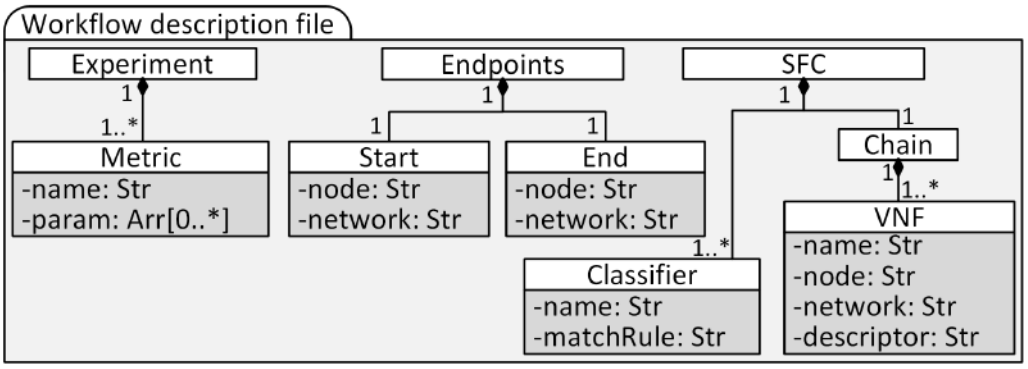


Figure 4.5: UML class diagram of the JSON structure of the workflow description file. The description file contains all parameters to set up the environment, configure parameters knobs and obtain the desired metrics.

ule also instantiates all data collection modules. The SFC platform hosts tools that monitor the resource usage and feed the passive data collection module. Moreover, active data collection takes place between source and destination of the SFC. Depending on the tested SFC platform, active data collection is implemented as active measuring tools instantiated in virtual machines, or on physical machines that behave as SFC endpoints. After the accomplishment of measures, the data collection modules report the gathered data to the management module. The management module also keeps a local data repository, which is implemented as a non-structured database (NoSQL) based on the open source Elasticsearch[2]. The analysis module computes the result and reports the final metrics to the control module. Finally, control module exports experimental results to the framework user and publishes it on the visualization module.

We define a JSON-based description language to represent the detailed configuration of an environment and the workflow of an arbitrary experiment. The workflow description file is shown in Figure 4.5 as a UML class diagram. The diagram represents the JSON structure of the parameters defined in the workflow description file. The parameters include information about endpoints, resource allocation, resource placement, VNF descriptors, VNF types, VNF chaining order, and the measurements to evaluate the SFC. The VNF descriptor parameter is the path to a VNF description file based on TOSCA standards and YAML language. Moreover, the parameters list is adaptable to comprise adjustable knobs of different VNFs.

[2]Available at `https://www.elastic.co/products/elasticsearch`.

4.3.1 The SFCPerf Framework Implementation[3]

We conceive the framework as a modular object-oriented software project in which new features are achieved by adding new modules to the framework. Thus, the SFCPerf is built upon some basic parent classes that provides the API (Application Programming Interface) for any new module. The driver module, for instance, assures the infrastructure-agnostic property. We conceive this module as a translation module from the experiments provided by SFCPerf and the NFV platform. For each different platform, we develop a new driver module as an API between SFCPerf and the CLI (command line interface) of the new platform. The new driver module inherits the main functions from the parent driver module. In our prototype, we develop a driver for the Tacker VNF manager used in OPNFV. Moreover, automating the experimentation is achieved by implementing a workflow control mechanism, which follows the experimentation definition on a JSON file. The experiments are also implemented as separated independent modules, which are imported and executed in run-time according to the parameters and the sequence in the JSON file. The automation also assures the repeatability property of the conducted experiments, since the JSON input file fully describes the topology and conditions necessary to orchestrate the SFC and obtain the desired metrics. Furthermore, performance comparison between different tests and scenarios are obtained through the execution of a new round of experiments while inputting the same JSON description file with slightly changes on the test or scenario configuration. It assures equality of the experiment conditions as the workflow of the SFCPerf configures the new experimentation scenario and keeps the same experiment parameters. Thus, comparing the results of different SFC platform resides into running the same JSON file calling the correspondent driver module of the SFC platform. Repeatability property is achieved by a new execution of a experiment-definition JSON file. Besides, experiment reproducibility is achieved by providing a copy of the experiment-definition JSON file with all modules used during the experimentation. It is worth noting that visualization of the results is also handled by an other module that is imported and executed in run-time. The default visualization module of the SFCPerf exports the experiment results to the ElasticSearch database and enables visualization through Kibana.

4.4 The Service Chain Security Prototype

To demonstrate the functionalities of our proposed SFCPerf framework, we develop and evaluate a service function chain prototype considering different scenarios. We

[3]The SFCPerf framework is written in Python language and its source code is available at `https://github.com/ijochem/SFCPerf`.

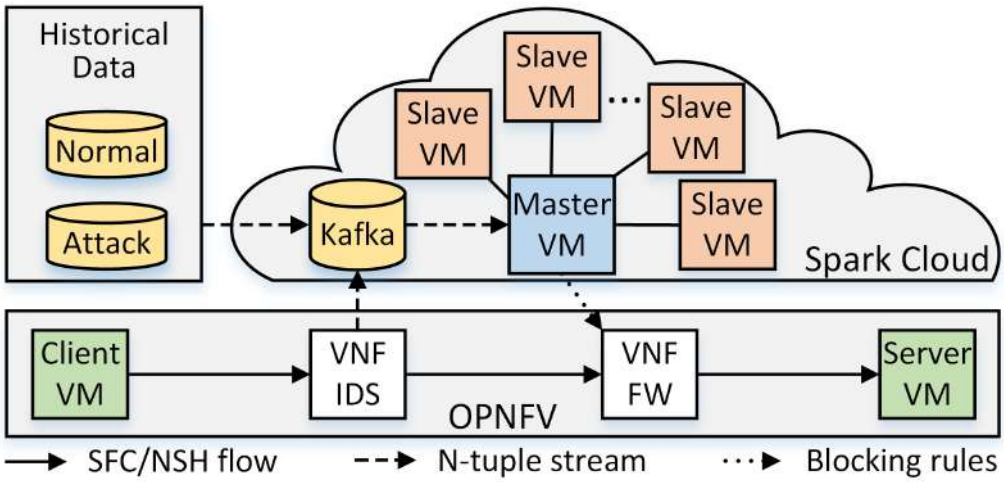


Figure 4.6: The architecture of a service function chain use case, composed of a virtual intrusion detection system and a firewall, evaluated with the proposed SFCPerf framework. The intrusion detection system VNF uses a distributed stream processing cloud to analyze data traffic and set blocking rules in real time on the subsequent firewall VNF.

use the open platform for network function virtualization (OPNFV) with an SDN and NFV hybrid architecture to implement service function chaining. The chaining is built upon rules in the software switch Open vSwitch, using the OpenFlow protocol. These rules are managed by the SDN OpenDaylight controller coupled with a VNF manager and orchestrator, named Tacker.

One of the main uses for network function virtualization is correspondent to ensure the correct, coherent and consistent application of policies to network traffic [55]. The prototype to be evaluated from SFCPerf is a chain composed of two security functions, the proposed intrusion detection system and a firewall. The intrusion detection system analyzes packets, generates alarms, and defines packet filtering rules, while the firewall implements the rules, creating security perimeters on the network. Intelligent chaining of these virtual functions provides the flexibility to instantiate complex real-time security mechanisms at any point of the network. The key idea of our prototype is to associate the processing scalability provided by a streaming processing cloud, realized by Spark[4], with the flexibility of security functions chaining, in order to enable a real-time response to malicious traffic. Figure 4.6 shows the architecture of our service function chaining prototype. One advantage of such architecture is allowing to scale the stream processing jobs along the mirroring and flow handling in the cloud, and therefore, independent from the network function virtualization infrastructure.

The main element of the architecture is the intrusion detection system that em-

[4]Available at `https://spark.apache.org/`.

ploys stream processing techniques to perform real-time traffic analysis [85]. The packets are captured through traffic mirroring by the IDS module, which acts directly in the chain. These packets are abstracted into quintuple flows, which are defined as a sequence of packets with the same source IP, destination IP, source port, destination port, and transport protocol, during a time window. In total 46 quintuple-flow characteristics from the open source tool Flowtbag[5] are extracted and published in a publish/subscribe message service of Apache Kafka [27]. The features extracted at this part differ from the approach presented in Section 3.2 which we define flow as IP-IP to detect network-layer attacks. Hence, in this scenario our focus is in detecting application-layer attacks. After the features are published in the message service, the service operates as a filtering and data flow manipulation system at low latency, in which flow statistics are queued and, then, consumed by the classification module. The classification module, in turn, is instantiated in a dedicated cloud for classification and contains Apache Spark as its main processing core. We select Apache Spark stream processing framework as the core of stream processing due to the lowest loss rate among other stream processing frameworks [86] and to the highest number of open-source contributors considering number of Github commits and users, which also implies in a mature documentation publicly available. Spark is implemented in a cluster of virtual machines on a master/slave model, which slaves have a capacity for expansion and reduction of resources. Other stream processing frameworks, such as Storm[6] and Flink[7], could also be implemented as the processing core without loss of generalization. Indeed, Flink has been widely adopted in the literature of intrusion detection due to the native stream processing capabilities since its conception [87].

The implemented IDS uses the lambda architecture [88], composed of three layers: a data processing layer, a batch processing layer, and a service layer. The data processing layer handles real-time flow data, such as feature selection techniques to online classification algorithms [89]. A batch processing layer analyzes a large amount of data stored through distributed computing techniques, such as map-reduce. Finally, a service layer aggregates information obtained from the two previous layers to create outputs of the analyzed data. The IDS classifies malicious or benign flows through machine-learning algorithms based on a decision tree classification algorithm.

A historical database of network traffic labeled as attack or as normal is used to train the machine-learning algorithm in an off-line manner [27]. The parameters calculated during the off-line data processing adjust the classification model in real-

[5]Available at `https://github.com/DanielArndt/flowtbag`.
[6]Available at `http://storm.apache.org/`.
[7]Available at `https://flink.apache.org/`.

time. Thus, the system is adaptive, as parameters can be updated and adjusted to new network behaviors. After training, the IDS performs real-time traffic classification and, by detecting malicious flows, is able to send blocking rules to the firewall.

To evaluate and compare the performance of different virtual network functions, we develop a VNF firewall, a prototype of an IDS, and a SFC Proxy. Our VNF firewall implementation contains a module capable of encapsulating and decapsulating NSH packets, built upon the open-source Python application *vxlan_tool*[8]. This application is extended to support a JSON object that stores blocking rules based on the 5-tuple of a packet. The 5-tuple stores the source and destination IP addresses, the source and destination ports and the transport protocol type. Before forwarding the packet to the subsequent VNF in the chain, the 5-tuple is checked over the existing set of rules. A RESTful interface is implemented to enable rule insertions and deletions in the set.

The second evaluated security function is the IDS, which consists of the two modules described in Section 4.4. The first module acts directly in the chain and performs two basic functions over the traffic, to extract characteristics of the flows in a two-second time window, and, at the same time, to forward the packets to the next VNF. In addition, the VNF publishes the extracted characteristics in the queuing and data-flow manipulation system Kafka to be read by the second processing module, running outside the chain.

Last, we develop an SFC proxy, shown in Figure 4.7. The main goal of the proxy is to enable the deployment of VNFs unaware of the SFC encapsulation. Hence, we hide the NSH encapsulation from the VNF application by instantiating a pair of user-space interfaces. The VNF application accesses the user-space interfaces like a common network interface. It is worth noting that OPNFV implementation of NSH also applies a VXLAN encapsulation to forward packets between the service function forwarder and the VNF. Our proxy enhances the VXLAN packet handling by using a software switch in kernel space, which only forwards NSH-encapsulated packets to the user-space network interfaces. Our proxy deployment is based on Open vSwitch version 2.8.9 and the NSH encapsulation and decapsulation are performed by OpenFlow 1.4 rules implemented on user-space Open vSwitch data paths.

4.5 Evaluation and Results

We use the SFCPerf framework to evaluate the service function chaining prototype shown in Figure 4.6. The performance and efficiency are evaluated for different

[8]Available at `https://github.com/opendaylight/sfc/blob/master/sfc-test/nsh-tools/vxlan_tool.py`.

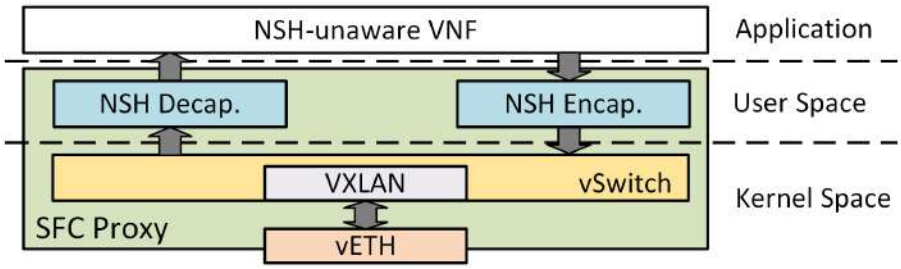


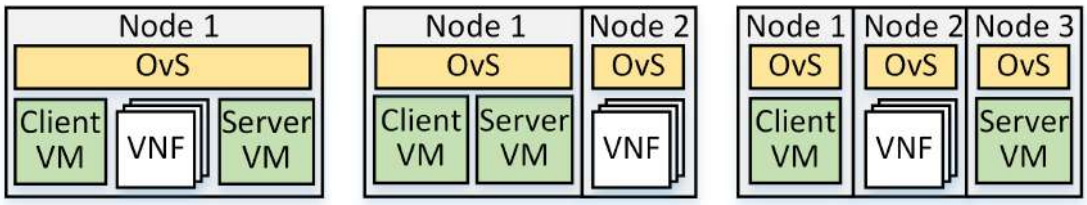
Figure 4.7: The architecure of our SFC proxy implementation, in which an NSH-unaware VNF runs on top of a pair of user-space network interfaces that decapsulate incoming traffic and encapsulate outgoing traffic with NSH.

topologies and configurations. We implement the experiments over OPNFV Danube 2.0[9]. The OPNFV platform deploys the NFV-MANO reference architecture based on the cloud operating system OpenStack[10]. The OPNFV is built through the Fuel installer in a scenario that provides the service functions chaining architecture referenced in RFC 7665. This scenario also deploys the SDN controller OpenDaylight, which manages the data link layer, the VNF manager Tacker, and the software switch Open vSwitch compatible with NSH. The aforementioned software versions of Openstack, OpenDaylight, Open vSwitch, and Tacker, used in this work are branches that have been adapted for integration to the OPNFV platform and are publicly available in the Danube 2.0 version of OPNFV. It is important to emphasize that the OPNFV platform is still in development phase, with poor documentation, and many errors and code bugs had to be debugged and fixed. The network function chain prototype using NSH is a pioneering implementation in Brazil.

The hardware environment consists of a controller node Intel 8-Core i7-4770 CPU 3.40 GHz processor with 32 GB RAM, and three compute nodes, Intel Xeon 8-core X5570 2.93 GHz processor with 96 GB RAM (node 1), Intel 8-Core i7-6700 CPU, 3.40 GHz with 64 GB RAM (node 2) and Intel 8-Core i7-2600 CPU, 3.40 GHz with 32 GB RAM (node 3). All machines are interconnected through a top-of-the-rack switch over 1 Gb/s network interfaces that comprise the five VLANs needed for OPNFV cloud: public, private, management, storage and Preboot Execution Environment.

The performance evaluation of the service function chaining is based on the RFC 7665 architecture using the NSH protocol, as shown in Figure 4.2. Therefore, SFCPerf analyzes several topologies for different parameters that affect the performance of the chaining. The evaluated metrics include the placement of the endpoint VMs and of the chain of VNFs, defined by each given topology, the overhead that

[9]Available at `https://www.opnfv.org`.
[10]Available at `https://www.openstack.org`.



(a) Topology 1. (b) Topology 2. (c) Topology 3.

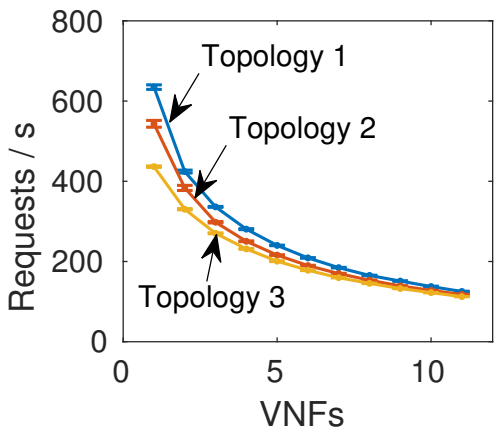
Figure 4.8: Topologies of the performance evaluation scenarios of the service function chaining: a) client, server, and chain of VNFs in the same node; b) client and server on a separated node from the node that hosts the chain; c) client, server, and chain on three separated nodes.

each network function introduces in the chaining, and the number of virtual processing cores requested by a VNF. In addition, the implemented service function chaining, as well as each developed VNF, are evaluated by SFCPerf. The evaluation considers latency, throughput and number of responded HTTP requests. Finally, we evaluate the implemented chaining within a real scenario. We inject real network traffic, which is analyzed, classified and filtered in real time. This evaluation compares an on-line classification with the off-line classification out of the chain for accuracy, and for the efficiency of real-time detection and automatic blocking of malicious flows.

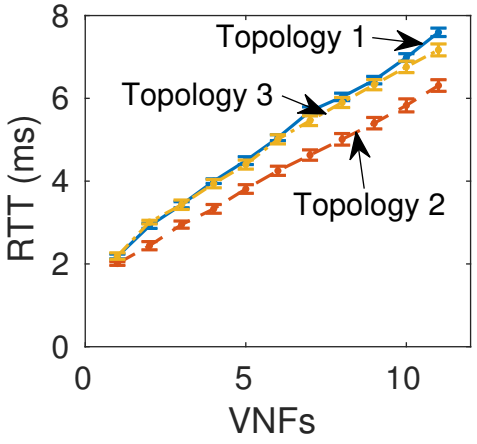
The chosen topologies for evaluation are illustrated in Figure 4.8. The first topology (Topology 1), illustrated in Figure 4.8(a), uses only one compute node to instantiate the virtual function chain and the endpoint VMs. Thus, the endpoint VMs and the VNFs compete for network resources of the same physical node, and there are no physical link hops between nodes. Figure 4.8(b) shows Topology 2, in which the endpoint VMs are instantiated on the same node, and the chain of VNFs on another node. In the third topology (Topology 3), shown in Figure 4.8(c), the server VM, the client VM, and the chain of VNFs are instantiated in three distinct nodes. Thus, Topologies 2 and 3 require the use of more software switches, since each node has a local software switch (Open vSwitch – OvS) to control its network resources.

Figure 4.9 shows the impact on performance results obtained in each topology relative to the number of VNFs in the chain and considering a 95% of confidence interval. It is worth mentioning that the VNFs in this specific scenario are extremely simple network functions that only forward incoming packets to the next VNF in the chain. The forwarding process consists in decrementing the service index field by 1 in the NSH protocol header, and then the packet is returned to the Open vSwitch, which sends it to the next virtual function. Thus, the processing and the delay

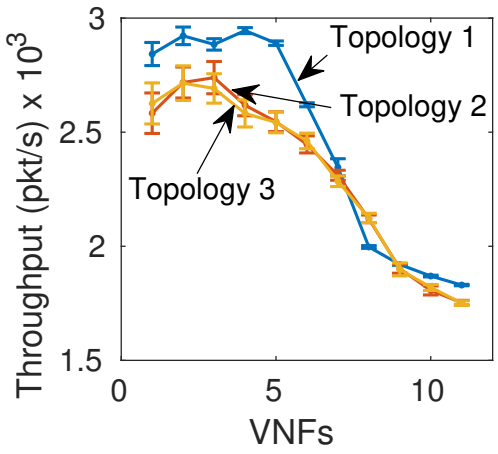
are practically only due to the the overhead of handling the NSH protocol. The application responsible for these operations is a tool written in Python (*vxlan_tool*) for tests with the NSH header.



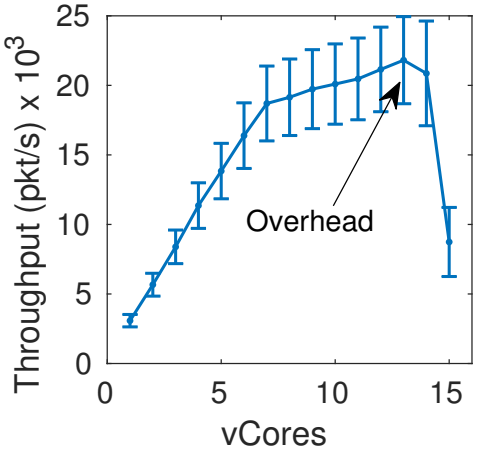
(a) Maximum HTTP requests rate.



(b) Round trip time.



(c) Maximum throughput.



(d) Maximum throughput according to the number of virtual cores.

Figure 4.9: Impact on the performance of network function chains considering: a) HTTP request rate supported by the chain; b) chain round-trip time; c) chain throughput; and d) throughput of a single VNF on Topology 1 as a function of the number of assigned virtual cores.

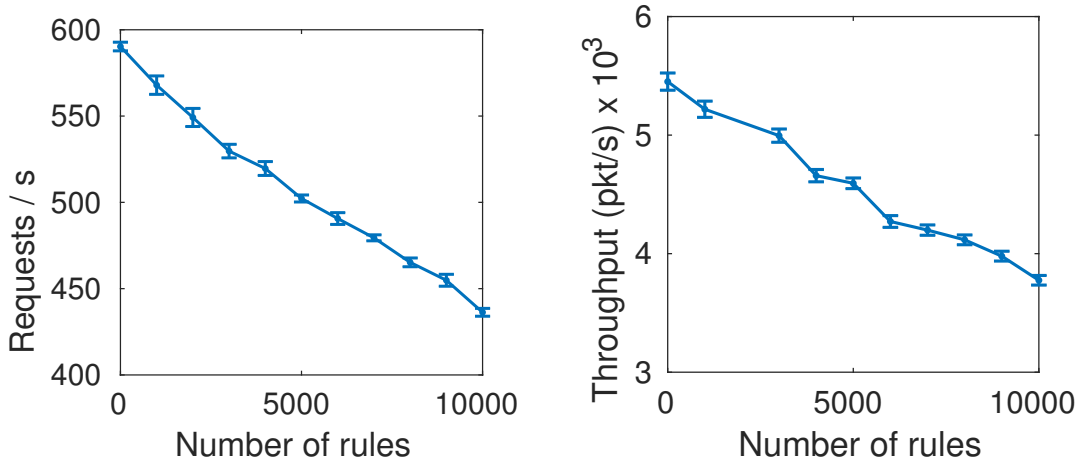
Figure 4.9(a) compares the three topologies in relation to the rate of HTTP requests performed from a client VM to a server VM that traverses the chain. Note that Topology 1 provides the best rate of HTTP requests for short chains of VNFs. The difference, however, becomes negligible when the chain length exceeds 8 VNFs. It demonstrates that for short chains, the overhead introduced by spreading the client, server and VNFs on different nodes is the performance limiting factor. Nevertheless, longer chains introduce an overhead that exceeds this factor. Figure 4.9(b)

shows that the round-trip time in all topologies grows linearly with the increase of the chain length. Topology 2 presented a significantly lower latency increase than the other two, because the client and server VMs are on the same physical node, which decreases the packet round-trip time. The node does not share resources with other VNFs. We conclude that the increase of physical link hops, as well as the sharing of resources on the same node are factors that compromise the end-to-end delay. Hence, Topology 2 presents a fair trade-off of these factors. Figure 4.9(c), in turn, shows the maximum throughput in packets per second for each topology as a function of the chain length. The packet size used in the experiments is 1334 bytes, which represents the maximum transmission unit for the Ethernet packet sent by the client, which suffers no IP fragmentation with NSH encapsulation. Topology 1 presents a better throughput in relation to the others when chaining few VNFs, due to lower number of physical link hops between nodes. It is important to notice that the increase of the number of VNFs in all the topologies implies resources competition on the node that hosts the chain, which considerably compromises throughput.

The major limiting factor for throughput is the *vxlan_tool* application that decapsulates the NSH packets. This application, by default, operates sequentially in only one processing core, and we extended it for parallel execution on multiple cores. The effect of this change is observed in Figure 4.9(d), which shows the increased throughput of a VNF in a unit-length chain in which we allocate more dedicated virtual processing cores (vCPUs). In this way, VNF retains more processing power and is able to perform more packet operations per second until it reaches the hypervisor processing limit. It is expected, however, that in the next versions of OPNFV platform, the NSH encapsulation and decapsulation will be implemented as a kernel module of the operating system of virtual machines to assure performance gain.

Figures 4.10(a) and 4.10(b) characterize the maximum rate of HTTP requests and the maximum throughput from the VNF firewall when varying the number of blocking rules configured into the firewall. We observe a linear reduction for both metrics when we increase the number of rules. This reduction, however, impacts the performance of the VNF only by 1% for a considerable usage of 500 rules configured into the firewall. The increase on number of rules also did not significantly impact the latency for the packets that traverse the firewall VNF, instead the latency kept a mean value near 2 ms.

As SFCPerf assures repeatability, we designed a scenario and performed a comparison experiment of four VNF approaches. Figure 4.11 shows the performance of one-length chains of two different simple VNFs (a forwarder, and an SFC proxy), two virtual security functions (a firewall, and a IDS), and of the composition of the two virtual security functions. In all chains, Topology 1 is used as a reference, providing only one virtual core for each VNF. It can be observed that the performance



(a) Maximum rate of HTTP requests regarding the number of rules from the firewall VNF.

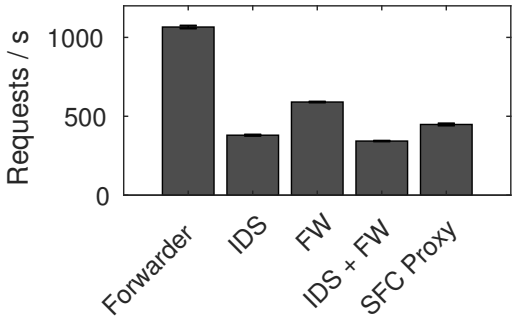
(b) Maximum throughput regarding the number of rules from the firewall VNF

Figure 4.10: Impact on performance of (a) maximum HTTP request rate (b) maximum throughput regarding the number of rules from the firewall VNF.

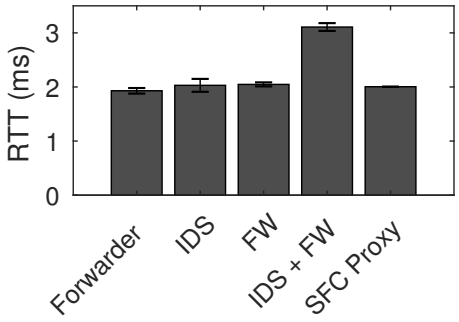
of the VNF that only forwards packets is superior to the other virtual functions and, thus, it is used as a baseline. Figure 4.11(a) and Figure 4.11(c) show similar results regarding the maximum rate of HTTP requests and the supported throughput of each chain. The firewall VNF presents better results than IDS VNF in both metrics, thus the chain of both VNFs has the performance limited by the IDS, with a small overhead due to the extra hop between the two VNFs. Figure 4.11(b), however, shows that the latency overload introduced by each virtual security function is very low, remaining at a similar time to the baseline threshold. The chain of firewall and IDS increases by 50% the packet round-trip time, which was already predicted from Figure 4.9(b). Nevertheless, this value is 33% lower in the case where the firewall and IDS functions separately operate over the traffic. It is worth to highlight that the forwarder, which is implemented with the Python `vxlan_tool`, performed better than the SFC proxy VNF. Although we deploy the NSH-unaware VNF, preceded by the SFC proxy, as an Open vSwitch associated with a Linux Bridge, the overhead of copying packets between user-space and kernel-space contexts slows down the packet handling.

Finally, we use the SFCPerf framework to evaluate the chain for real-time online classification of network traffic dataset of a Brazilian telecommunications operator [27]. We implement the decision tree machine-learning algorithm for the traffic classification [27]. First, a fraction of the traffic (20%), labeled as attack or normal by Suricata IDS[11], was used to train the classifier in an off-line manner. The remainder fraction is injected by the client VM in direction to the server and traverses

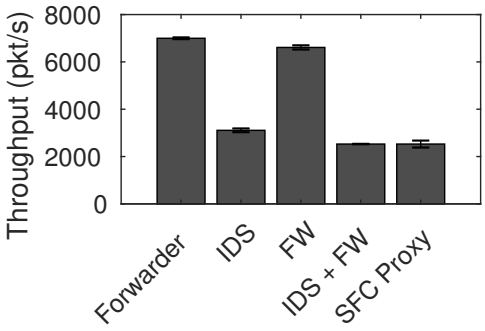
[11] Available at `https://suricata-ids.org`.



(a) Maximum rate of HTTP requests.



(b) Average round-trip time.



(c) Maximum throughput.

Figure 4.11: Impact on the performance introduced by each virtual security function and by the chaining of the two VNFs in relation to: (a) maximum rate of HTTP requests; (b) packet round trip time; and (c) maximum throughput.

the two virtual security functions. Table 4.1 shows the results of the on-line classification[12], comparing the flows that reach the server with those that were blocked in the core of the network by the virtual firewall.

Table 4.1: Confusion matrix of real-time flow classification and blocking. The flows that reach the server are represented as Normal, while those blocked by our VNF as Attack.

	TP	**FP**	**TN**	**FN**
Attack	430	2277	4412795	1658973
Normal	4412795	1658973	430	2277

The results show an accuracy of 72.7% for the classification and that 0.02% of the malicious flows were blocked before reaching the server. However, since the proposed firewall is a reactive software defense, there is a real reduction of 15% of the total volume of malicious traffic that reaches the server. Due to the processing time of a machine-learning classification task, flows that have short duration time are not feasible to be blocked in real time, since they finish before the detection

[12]TP: True Positives; FP: False Positives; TN: True Negatives; FN: False Negatives.

occurs. In the other hand, results show that the chaining of IDS with a firewall was effective on blocking malicious flows with long duration time. One example of this malicious flows with such characteristics are connections established by attackers when having the control of hacked devices to perform DDoS.

Chapter 5

Conclusion

Distributed threats and coordinated malicious actions are a major concern in the current network scenario, since most of attacks are not detected when flows are analyzed separately. Graph Theory combined with machine-learning techniques enable the automatic identification of threat patterns that are not feasible to be detected when social interactions among IP addresses are not analyzed. As the first part of this manuscript, we propose a graph-based algorithm for feature enrichment in online intrusion detection systems. The proposal enhances the detection of distributed network threats, such as distributed denial of service, port scans or botnet traces. We propose an online intrusion detection architecture that employs an enrichment module containing the proposed enriched algorithm to infer graph-based features from traffic samples collected in a fixed time window. The architecture of the IDS is composed of four other modules related to data collection, processing of machine-learning techniques, visualization, and historical database. In the proposed enriched process, each set of instances arrived in a time window generates a static and directed graph. The graph model considers IP addresses as vertices and data transmission between two IP addresses as edges. The graph of a time window is parsed into sub-graphs composed of connected components. An algorithm, designed to run with parallelism over connected components, infers a set of graph-based features separated in three categories: local metrics, vertex metrics and edge metrics. For the algorithm evaluation, we use three different datasets, a synthetic dataset produced in GTA/UFRJ lab, a network operator dataset containing real data from broadband users, and a realistic and publicly available botnet dataset. To assess the impact of our proposal into different machine-learning classifiers, we employ three distinct learning techniques, decision tree, naive Bayes, and neural network. Additionally, we evaluate the scenario in which a feature selection module precedes the classification. Results with a linear correlation filter and a dimensionality reduction with Principal Component Analysis showed that both techniques reduce the number of features to be processed at a small cost in the classification performance. The

exception for this, however, is the PCA reduction employed with the naive Bayes algorithm, which reduced the amount of features to be processed at the same time it improved the classification performance. For seven of nine evaluated scenarios, our enrichment proposal showed an improvement in traffic classification compared to the original set of features inferred by TCP/IP packet header analysis. The only exceptions were the decision tree in the GTA/UFRJ lab dataset, which the performance remained stable, and for the naive Bayes in the ISCX botnet, where the assumption of independent features do not represent the actually feature correlation for the dataset. For simpler algorithms, such as naive Bayes, the proposed method improved the accuracy up to 15.7%. We conclude the naive Bayes algorithm, indeed, is more effective in detecting network threats by using only graph-based features than if combining it with TCP/IP header features. Finally, comparing the obtained gain among all algorithms, our proposal was better coupled with neural network based techniques, which showed improvements in the classification performance for all evaluated datasets. Indeed, our proposed enrichment reduced up to 9.4% times the false negative rate for the multilayer perceptron technique. Therefore, our graph-based feature enrichment proposal is an efficient complementary feature to detect dissimulated network attacks that are not feasible to be detected when group behavior is not analyzed. We enhanced the overall detection capabilities of coordinated attacks, from distributed network-layer attacks to botnet C&C traces.

Network security relies not only on the threat detection scheme, but also on the correct positioning of security solutions in the network. Network function virtualization technology paves the way for flexible and on-demand security network function deployment. Chaining virtual security functions enables the network administrator to provide complex network services comprised of features developed by different VNF manufacturers. Hence, mechanisms that automate the chaining of such network functions, perform test benchmarking, and accelerate the development of VNFs are essential. To this end, in the second part of this work we proposed SFCPerf, an automatic performance evaluation framework for service function chaining. SFCPerf assures repeatability for testing and comparison of different virtual network function chains. SFCPerf is agnostic of the underlying NFV infrastructure, which is mandatory for evaluating different NFV proposals in the early stage of NFV technology. We developed an automatic SFC testing workflow composed of three phases: setup phase, experimental phase; and post-processing phase. During the setup phase, SFCPerf orchestrates all the virtual resources and network configuration to prepare the environment for the test execution. In the experimental phase, our framework executes and collects the performance measurements from a set of well-defined tests and metrics defined previously in a test configuration file. For the post-processing phase, the framework analyzes data from all experiments from a set of desired met-

rics. We also presented a performance evaluation of a service function chain prototype, described by coherent chaining of virtual security functions. We employed the SFCPerf framework to evaluate the prototype, composed of an intrusion detection system based on stream data processing in the cloud chained to a reactive firewall. We construct our prototype based on the European Telecommunications Standards Institute (ETSI) NFV MANO architecture, the current most mature standard for NFV architecture. The employed service function chain complies to the RFC 7665, standardized by the Internet Engineering Task Force (IETF) and to the specifications of the Network Service Header (NSH) protocol. The results provided by our proposed SFCPerf framework showed the impact in terms of throughput, round-trip time, and HTTP request rate of several VNF chaining scenarios. These performance measures were obtained for different chain implementation topologies, chain lengths, and varying number of virtual cores offered to VNFs. Our framework allowed the comparison of the overhead introduced by each virtual function individually. The results showed that the main impact factors on the performance were the number of physical link hops between nodes and the competition for resources at shared physical nodes. Furthermore, the SFC throughput is directly related to the number of cores assigned to the virtual functions, which determines the number of packets that each VNF is able to process. Finally, we evaluate our prototype during the execution of security virtual functions. We tested our security prototype regarding real-time malicious traffic detection and blocking. The evaluated scenario consisted of a virtual IDS function automatically setting block rules to the subsequent virtual firewall function. Results showed that the prototype was able to block 0.02% of the malicious traffic before they reach the end server, which represented 15% of the total malicious traffic.

5.1 Future Work

As medium-term future work, we will evaluate the impact of graph-based enrichment for detecting network traffic anomalies as well as evaluating new set of features inferred from graph analysis. As long term, we will port our proposed algorithm to real production state, assisted by stream processing frameworks and distributed processing. Furthermore, concerning the automation of NFV benchmarking, we will evaluate the performance of the network function chaining on new topologies and over different NFV platforms, to assess the performance bottlenecks of other network function virtualization approaches.

Bibliography

[1] AKAMAI. *Q4 2016 State of the Internet / Security Report*. Technical Report 4, Akamai, 2017. Available at: <https://content.akamai.com/pg7967-q4-soti-security-report.html>.

[2] KOLIAS, C., KAMBOURAKIS, G., STAVROU, A., et al. "DDoS in the IoT: Mirai and Other Botnets", *Computer*, v. 50, n. 7, pp. 80–84, 2017. ISSN: 0018-9162. doi: 10.1109/MC.2017.201.

[3] MAJKOWSKI, M. "Memcrashed - Major amplification attacks from UDP port 11211". 2018. Available at: <https://blog.cloudflare.com/memcrashed-major-amplification-attacks-from-port-11211/>. Accessed on: 31-08-2018.

[4] MORALES, C. "NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us". 2018. Available at: <https://asert.arbornetworks.com/netscout-arbor-confirms-1-7-tbps-ddos-attack-terabit-attack-era-upon-us/>. Accessed on: 31-08-2018.

[5] RUDIS, B. "The Flip Side of memcrashed". 2018. Available at: <https://blog.rapid7.com/2018/02/27/the-flip-side-of-memcrashed/>. Accessed on: 31-08-2018.

[6] ANDREONI LOPEZ, M., MATTOS, D. M. F., DUARTE, O. C. M. B. "An elastic intrusion detection system for software networks", *Annales des Telecommunications/Annals of Telecommunications*, v. 71, n. 11-12, pp. 595–605, dec 2016. ISSN: 0003-4347. doi: 10.1007/s12243-016-0506-y.

[7] VERIZON. *2018 Data Breach Investigations Report*. Technical report, Verizon, 2018. Available at: <https://enterprise.verizon.com/resources/reports/DBIR_2018_Report_execsummary.pdf>.

[8] GLUHAK, A., KRCO, S., NATI, M., et al. "A survey on facilities for experimental Internet of things research", *IEEE Communications Magazine*, v. 49, n. 11, pp. 58–67, 2011.

[9] CÁRDENAS, A. A., MANADHATA, P. K., RAJAN, S. "Big data analytics for security intelligence", *University of Texas at Dallas@ Cloud Security Alliance*, 2013.

[10] CLAY, P. "A modern threat response framework", *Network Security*, v. 2015, n. 4, pp. 5–10, 2015.

[11] SANZ, I. J., LOPEZ, M. A., MATTOS, D. M. F., et al. "A cooperation-aware virtual network function for proactive detection of distributed port scanning". In: *2017 1st Cyber Security in Networking Conference (CSNet)*, pp. 1–8, Oct 2017. doi: 10.1109/CSNET.2017.8242000.

[12] MAYHEW, M., ATIGHETCHI, M., ADLER, A., et al. "Use of machine learning in big data analytics for insider threat detection". In: *IEEE Military Communications Conference, MILCOM*, pp. 915–922, 10 2015.

[13] LOBATO, A. G. P., ANDREONI LOPEZ, M., REBELLO, G. A. F., et al. "Um Sistema Adaptativo de Detecção e Reação a Ameaças". In: *Anais do XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg'17*, pp. 400–413, 2017.

[14] HENKE, M., SANTOS, C., NUNAN, E., et al. "Aprendizagem de máquina para segurança em redes de computadores: Métodos e aplicações", *Minicursos do XI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2011)*, v. 1, pp. 53–103, 2011.

[15] KHETTAB, Y., BAGAA, M., DUTRA, D. L. C., et al. "Virtual security as a service for 5G verticals". In: *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, April 2018. doi: 10.1109/WCNC.2018.8377298.

[16] PATTARANANTAKUL, M., HE, R., MEDDAHI, A., et al. "SecMANO: Towards Network Functions Virtualization (NFV) Based Security MANagement and Orchestration". In: *IEEE Trustcom/BigDataSE/ISPA*, pp. 598–605, ago. 2016.

[17] DA SILVA, A. S., WICKBOLDT, J. A., GRANVILLE, L. Z., et al. "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 27–35, abr. 2016.

[18] REYNAUD, F., AGUESSY, F. X., BETTAN, O., et al. "Attacks against Network Functions Virtualization and Software-Defined Networking: State-

of-the-art". In: *IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 471–476, jun. 2016.

[19] SANZ, I. J., ANDREONI LOPEZ, M., MATTOS, D. M. F., et al. "A Cooperation-Aware Virtual Network Function for Proactive Detection of Distributed Port Scanning". In: *2017 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil, out. 2017.

[20] MEDHAT, A. M., TALEB, T., ELMANGOUSH, A., et al. "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges", *IEEE Communications Magazine*, v. 55, n. 2, pp. 216–223, fev. 2017. ISSN: 0163-6804.

[21] SENDI, A. S., JARRAYA, Y., POURZANDI, M., et al. "Efficient provisioning of security service function chaining using network security defense patterns", *IEEE Transactions on Services Computing*, 2017.

[22] LUIZELLI, M. C., RAZ, D., SA'AR, Y., et al. "The actual cost of software switching for NFV chaining". In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 335–343, May 2017. doi: 10.23919/INM.2017.7987296.

[23] ETSI. *ETSI GS NFV-MAN 001: Network Functions Virtualisation; Management and Orchestration*. Technical report, ETSI, 2014.

[24] HALPERN, J., PIGNATARO, C. *Service Function Chaining (SFC) Architecture*. RFC 7665, RFC Editor, October 2015. Available at: <`http://www.rfc-editor.org/rfc/rfc7665.txt`>.

[25] SANZ, I. J., MATTOS, D. M. F., DUARTE, O. C. M. B. "SFCPerf: An automatic performance evaluation framework for service function chaining". In: *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, April 2018. doi: 10.1109/NOMS.2018.8406237.

[26] E MARTIN ANDREONI LOPEZ, I. J. S., REBELLO, G. A. F., DUARTE, O. C. M. B. "Um Sistema de Detecção de Ameaças Distribuídas de Rede baseado em Aprendizagem por Grafos", *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, v. 36, 2018. ISSN: 2177-9384.

[27] ANDREONI LOPEZ, M., SILVA, R. S., ALVARENGA, I., et al. "Collecting and Characterizing a Real Broadband Access Network Traffic Dataset". In: *2017 1st Cyber Security in Networking Conference (CSNet'17)*, Rio de Janeiro, Brazil, out. 2017.

[28] ANDREONI LOPEZ, M., SANZ, I. J., MENEZES, D. M., et al. "CATRACA: uma Ferramenta para Classificação e Análise Tráfego Escalável Baseada em Processamento por Fluxo". In: *Salão de Ferramentas do XVII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais - SBSeg'2017*, pp. 788–795, 2017.

[29] REBELLO, G. A. F., ALVARENGA, I. D., SANZ, I. J., et al. "SINFONIA: Gerenciamento Seguro de Funções Virtualizadas de Rede através de Corrente de Blocos", *Workshop em Blockchain: Teoria, Tecnologias e Aplicações (WBlockchain - SBRC)*, v. 1, n. 1/2018, 2018.

[30] LOPEZ, M. E. A., SANZ, I. J., LOBATO, A. G. P., et al. "Aprendizado de Máquina em Plataformas de Processamento Distribuído de Fluxo: Análise e Detecção de Ameaças em Tempo Real", *Minicursos do Simp{ó}sio Brasileiro de Redes de Computadores-SBRC 2018*, pp. 150–206, 2018.

[31] LOBATO, A. G. P., LOPEZ, M. A., SANZ, I. J., et al. "An Adaptive Real-Time Architecture for Zero-Day Threat Detection". In: *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, May 2018. doi: 10.1109/ICC.2018.8422622.

[32] STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., et al. "GrIDS - A Graph-Based Intrusion Detection System for Large Networks". In: *In Proceedings of the 19th National Information Systems Security Conference*, pp. 361–370, 1996.

[33] LIU, L., SAHA, S., TORRES, R., et al. "Detecting malicious clients in ISP networks using HTTP connectivity graph and flow information". In: *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 150–157. IEEE, 2014.

[34] ILIOFOTOU, M., PAPPU, P., FALOUTSOS, M., et al. "Network Monitoring Using Traffic Dispersion Graphs (Tdgs)". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pp. 315–320, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-908-1. doi: 10.1145/1298306.1298349.

[35] ILIOFOTOU, M., KIM, H.-C., FALOUTSOS, M., et al. "Graption: A graph-based P2P traffic classification framework for the internet backbone", *Computer Networks*, v. 55, n. 8, pp. 1909–1920, 2011.

[36] ESWARAN, D., FALOUTSOS, C., GUHA, S., et al. "SpotLight: Detecting Anomalies in Streaming Graphs". In: *Proceedings of the 24th ACM*

SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD'18, pp. 1378–1386, New York, NY, USA, 2018. ACM. ISBN: 978-1-4503-5552-0. doi: 10.1145/3219819.3220040.

[37] MINGQIANG, Z., HUI, H., QIAN, W. "A graph-based clustering algorithm for anomaly intrusion detection". In: *2012 7th International Conference on Computer Science Education (ICCSE)*, pp. 1311–1314, July 2012.

[38] CHOWDHURY, S., KHANZADEH, M., AKULA, R., et al. "Botnet detection using graph-based feature clustering", *Journal of Big Data*, v. 4, n. 1, pp. 14, May 2017. ISSN: 2196-1115. doi: 10.1186/s40537-017-0074-7.

[39] MUTTIPATI, A. S., PADMAJA, D. P. "Analysis of Large Graph Partitioning and Frequent Subgraph Mining on Graph Data", *International Journal of Advanced Research in Computer Science*, v. 6, n. 7, pp. 29–40, 2017. ISSN: 0976-5697. doi: 10.26483/ijarcs.v6i7.2579.

[40] MANZOOR, E., MILAJERDI, S. M., AKOGLU, L. "Fast memory-efficient anomaly detection in streaming heterogeneous graphs". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1035–1044. ACM, 2016.

[41] SRICHARAN, K., DAS, K. "Localizing anomalous changes in time-evolving graphs". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1347–1358. ACM, 2014.

[42] PIETRASZEK, T., TANNER, A. "Data mining and machine learning—Towards reducing false positives in intrusion detection", *Information Security Technical Report*, v. 10, n. 3, pp. 169 – 183, 2005. ISSN: 1363-4127. doi: 10.1016/j.istr.2005.07.001.

[43] LI, W., MENG, W., LUO, X., et al. "MVPSys: Toward practical multi-view based false alarm reduction system in network intrusion detection", *Computers & Security*, v. 60, pp. 177 – 192, 2016. ISSN: 0167-4048. doi: 10.1016/j.cose.2016.04.007.

[44] CHEN, F., RANJAN, S., TAN, P.-N. "Detecting Bots via Incremental LS-SVM Learning with Dynamic Feature Adaptation". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pp. 386–394, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-0813-7. doi: 10.1145/2020408.2020471.

[45] SUPRANAMAYA, R., JOSHUA, R., FEILONG, C. "Machine Learning Based Botnet Detection Using Real-time Connectivity Graph-Based Traffic Features". 2014. US Patent 8762298 B1.

[46] BUCZAK, A., GUVEN, E. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection", *IEEE Communications Surveys Tutorials*, v. 99, pp. 1–26, 2015.

[47] NGUYEN, T. T., ARMITAGE, G. "A Survey of Techniques for Internet Traffic Classification Using Machine Learning", *Commun. Surveys Tuts.*, v. 10, n. 4, pp. 56–76, out. 2008. ISSN: 1553-877X. doi: 10.1109/SURV.2008.080406.

[48] WILLIAMS, N., ZANDER, S., ARMITAGE, G. "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification", *SIGCOMM Comput. Commun. Rev.*, v. 36, n. 5, pp. 5–16, out. 2006. ISSN: 0146-4833. doi: 10.1145/1163593.1163596.

[49] BERNAILLE, L., TEIXEIRA, R., AKODKENOU, I., et al. "Traffic Classification on the Fly", *SIGCOMM Comput. Commun. Rev.*, v. 36, n. 2, pp. 23–26, abr. 2006. ISSN: 0146-4833. doi: 10.1145/1129582.1129589.

[50] GARCÍA, S., GRILL, M., STIBOREK, J., et al. "An empirical comparison of botnet detection methods", *Computers & Security*, v. 45, pp. 100 – 123, 2014. ISSN: 0167-4048. doi: 10.1016/j.cose.2014.05.011.

[51] ÁLVAREZ CID-FUENTES, J., SZABO, C., FALKNER, K. "An adaptive framework for the detection of novel botnets", *Computers & Security*, v. 79, pp. 148 – 161, 2018. ISSN: 0167-4048. doi: 10.1016/j.cose.2018.07.019.

[52] BAKSHI, A., DUJODWALA, Y. B. "Securing Cloud from DDOS Attacks Using Intrusion Detection System in Virtual Machine". In: *2010 Second International Conference on Communication Software and Networks*, pp. 260–264, Feb 2010. doi: 10.1109/ICCSN.2010.56.

[53] MODI, C., PATEL, D., BORISANIYA, B., et al. "A survey of intrusion detection techniques in Cloud", *Journal of Network and Computer Applications*, v. 36, n. 1, pp. 42 – 57, 2013. ISSN: 1084-8045. doi: 10.1016/j.jnca.2012.05.003.

[54] FAYAZBAKHSH, S. K., SEKAR, V., YU, M., et al. "FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions".

In: *II ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 19–24, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2178-5.

[55] ZHANG, Y., BEHESHTI, N., BELIVEAU, L., et al. "StEERING: A software-defined networking for inline service chaining". In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, out. 2013.

[56] CSOMA, A., SONKOLY, B., CSIKOR, L., et al. "ESCAPE: Extensible Service Chain Prototyping Environment Using Mininet, Click, NETCONF and POX". In: *ACM Conference on SIGCOMM*, SIGCOMM '14, pp. 125–126, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2836-4.

[57] CALLEGATI, F., CERRONI, W., CONTOLI, C., et al. "Dynamic chaining of Virtual Network Functions in cloud-based edge networks". In: *1st IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, abr. 2015.

[58] MARTINS, J., AHMED, M., RAICIU, C., et al. "ClickOS and the Art of Network Function Virtualization". In: *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pp. 459–473, Berkeley, CA, USA, 2014. USENIX Association. ISBN: 978-1-931971-09-6.

[59] PANDA, A., HAN, S., JANG, K., et al. "NetBricks: Taking the V out of NFV". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, pp. 203–216. USENIX Association, 2016.

[60] BOUET, M., LEGUAY, J., COMBE, T., et al. "Cost-based placement of vDPI functions in NFV infrastructures", *International Journal of Network Management*, v. 25, n. 6, pp. 490–506, 2015.

[61] ANDREONI LOPEZ, M., MATTOS, D. M. F., DUARTE, O. C. M. B. "Evaluating allocation heuristics for an efficient virtual Network Function chaining". In: *7th International Conference on the Network of the Future (NoF)*, pp. 1–5, nov. 2016. doi: 10.1109/NOF.2016.7810141.

[62] EMMERICH, P., RAUMER, D., WOHLFART, F., et al. "Performance characteristics of virtual switching". In: *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pp. 120–125, out. 2014.

[63] CALLEGATI, F., CERRONI, W., CONTOLI, C., et al. "Performance of Network Virtualization in cloud computing infrastructures: The OpenStack case". In: *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pp. 132–137, out. 2014.

[64] BONAFIGLIA, R., CERRATO, I., CIACCIA, F., et al. "Assessing the Performance of Virtualization Technologies for NFV: A Preliminary Benchmarking". In: *2015 Fourth European Workshop on Software Defined Networks*, pp. 67–72, set. 2015.

[65] MIJUMBI, R., SERRAT, J., GORRICHO, J. L., et al. "Network Function Virtualization: State-of-the-Art and Research Challenges", *IEEE Communications Surveys Tutorials*, v. 18, n. 1, pp. 236–262, 2016. ISSN: 1553-877X.

[66] MIRKOVIC, J., BENZEL, T. V., FABER, T., et al. "The DETER project: Advancing the science of cyber security experimentation and test". In: *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pp. 1–7. IEEE, 2010.

[67] ALVARENGA, I. D., DUARTE, O. C. M. B. "RIO: A denial of service experimentation platform in a Future Internet Testbed". In: *2016 7th International Conference on the Network of the Future (NOF)*, pp. 1–5, nov. 2016.

[68] RIGGIO, R., YAHIA, I. G. B., LATRÉ, S., et al. "Scylla: A language for virtual network functions orchestration in enterprise WLANs". In: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 401–409, abr. 2016.

[69] QUINN, P., ELZUR, U., PIGNATARO, C. *Network Service Header (NSH)*. RFC 8300, RFC Editor, January 2018. Available at: <`http://www.rfc-editor.org/rfc/rfc8300.txt`>.

[70] BRANDES, U. "A faster algorithm for betweenness centrality", *The Journal of Mathematical Sociology*, v. 25, n. 2, pp. 163–177, 2001. doi: 10.1080/0022250X.2001.9990249.

[71] LOPEZ, M. A., SILVA, R. S., ALVARENGA, I. D., et al. "Collecting and characterizing a real broadband access network traffic dataset". In: *2017 1st Cyber Security in Networking Conference (CSNet)*, pp. 1–8, Oct 2017. doi: 10.1109/CSNET.2017.8241999.

[72] BEIGI, E. B., JAZI, H. H., STAKHANOVA, N., et al. "Towards effective feature selection in machine learning-based botnet detection approaches". In: *2014 IEEE Conference on Communications and Network Security*, pp. 247–255, Oct 2014. doi: 10.1109/CNS.2014.6997492.

[73] AMOR, N. B., BENFERHAT, S., ELOUEDI, Z. "Naive Bayes vs Decision Trees in Intrusion Detection Systems". In: *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pp. 420–424, New York, NY, USA, 2004. ACM. ISBN: 1-58113-812-1. doi: 10.1145/967900.967989.

[74] UMER, M. F., SHER, M., BI, Y. "Flow-based intrusion detection: Techniques and challenges", *Computers & Security*, v. 70, pp. 238 – 254, 2017. ISSN: 0167-4048. doi: 10.1016/j.cose.2017.05.009.

[75] SHAFER, J. C., AGRAWAL, R., MEHTA, M. "SPRINT: A Scalable Parallel Classifier for Data Mining". In: *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pp. 544–555, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN: 1-55860-382-4.

[76] MATTOS, D. M. F., DUARTE, O. C. M. B. "AuthFlow: authentication and access control mechanism for software defined networking", *Annals of Telecommunications*, v. 71, n. 11, pp. 607–615, Dec 2016. ISSN: 1958-9395.

[77] VICENTINI, C., SANTIN, A., VIEGAS, E., et al. "SDN-based and multitenant-aware resource provisioning mechanism for cloud-based big data streaming", *Journal of Network and Computer Applications*, 2018. ISSN: 1084-8045. doi: 10.1016/j.jnca.2018.11.005.

[78] HAN, B., GOPALAKRISHNAN, V., JI, L., et al. "Network function virtualization: Challenges and opportunities for innovations", *IEEE Communications Magazine*, v. 53, n. 2, pp. 90–97, fev. 2015. ISSN: 0163-6804.

[79] LI, Y., CHEN, M. "Software-Defined Network Function Virtualization: A Survey", *IEEE Access*, v. 3, pp. 2542–2553, 2015. ISSN: 2169-3536. doi: 10.1109/ACCESS.2015.2499271.

[80] KULKARNI, S., ARUMAITHURAI, M., RAMAKRISHNAN, K. K., et al. "Neo-NSH: Towards scalable and efficient dynamic service function chaining of elastic network functions". In: *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pp. 308–312, mar. 2017.

[81] OPNFV. "Open Platform for NFV". 2017. Available at: <`https://www.opnfv.org/`>. Accessed on: 10-06-2017.

[82] ETSI. "Open Source Mano (OSM) Release Four Technical Overview". 2018. Available at: <`https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseFOUR-FINAL.pdf`>.

[83] SUN, C., BI, J., ZHENG, Z., et al. "NFP: Enabling Network Function Parallelism in NFV". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM'17, pp. 43–56, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4653-5.

[84] ZAVE, P., FERREIRA, R. A., ZOU, X. K., et al. "Dynamic Service Chaining with Dysco". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM'17, pp. 57–70, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4653-5.

[85] ANDREONI LOPEZ, M., LOBATO, A. G. P., MATTOS, D. M. F., et al. "Um Algoritmo Não Supervisionado e Rápido para Seleção de Características em Classificação de Tráfego". In: *SBRC'2017*, Belém/PA, 2017.

[86] LOPEZ, M. A., LOBATO, A. G. P., DUARTE, O. C. M. B. "A Performance Comparison of Open-Source Stream Processing Platforms". In: *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec 2016. doi: 10.1109/GLOCOM.2016.7841533.

[87] VIEGAS, E., SANTIN, A., BESSANI, A., et al. "BigFlow: Real-time and reliable anomaly-based intrusion detection for high-speed networks", *Future Generation Computer Systems*, v. 93, pp. 473 – 485, 2019. ISSN: 0167-739X. doi: 10.1016/j.future.2018.09.051.

[88] MARZ, N., WARREN, J. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. 1st ed. Greenwich, CT, USA, Manning Publications Co., 2013.

[89] LOPEZ, M. A., MATTOS, D. M. F., DUARTE, O. C. M. B., et al. "A fast unsupervised preprocessing method for network monitoring", *Annals of Telecommunications*, Aug 2018. ISSN: 1958-9395. doi: 10.1007/s12243-018-0663-2.