UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDERSON SANTOS DA SILVA

ATLANTIC: A Framework for Anomaly Traffic Detection, Classification, and Mitigation in SDN

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Alberto Egon
Schaeffer-Filho

Porto Alegre
December 2015

CIP – CATALOGING-IN-PUBLICATION

Silva, Anderson Santos da

ATLANTIC: A Framework for Anomaly Traffic Detection, Classification, and Mitigation in SDN / Anderson Santos da Silva. – Porto Alegre: PPGC da UFRGS, 2015.

127 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2015. Advisor: Alberto Egon Schaeffer-Filho.

1. Anomaly detection. 2. Traffic classification. 3. Traffic monitoring. 4. Management framework. 5. Machine learning. I. Schaeffer-Filho, Alberto Egon. II. Título.

ACKNOWLEDGMENTS

I would like to thank all who somehow helped me and contributed to the conclusion of this master thesis.

Firstly, I would like to thank God. Secondly, to my parents and brothers who always believed in me. They teach me every day that the most important thing is what we live. Thirdly, to all colleagues of the computer networks group. In particular, room 212 staff by the discussions, jokes and the best clan of UFRGS :)

Finally, to Professor Alberto Egon Schaeffer-Filho for guidance and patience during this period.

"I keep six honest serving men. They taught me all I knew. Their names are What and Why and When and How and Where and Who".

— RUDYARD KIPLING.

ABSTRACT

Software-Defined Networking (SDN) aims to alleviate the limitations imposed by traditional IP networks by decoupling network tasks performed on each device in particular planes. This approach offers several benefits, such as standard communication protocols, centralized network functions, and specific network elements, such as controller devices. Despite these benefits, there is still a lack of adequate support for performing tasks related to traffic classification, because (*i*) the native flow features available in OpenFlow, such as packet and byte counts, do not convey sufficient information to accurately distinguish between some types of flows; (*ii*) there is a lack of support to determine what is the optimal set of flow features to characterize different types of traffic profiles; (*iii*) there is a need for a flexible way of composing different mechanisms to detect, classify and mitigate network anomalies using software abstractions; (*iv*) there is a need of online traffic monitoring using lightweight/low-cost techniques; (*v*) there is no framework capable of managing anomaly detection, classification and mitigation in a coordinated manner and considering all these demands. Additionally, it is well-known that anomaly traffic detection and classification mechanisms need to be flexible and easy to manage in order to detect the ever growing spectrum of anomalies. Detection and classification are difficult tasks because of several reasons, including the need to obtain an accurate and comprehensive view of the network, the ability to detect the occurrence of new attack types, and the need to deal with misclassification. In this dissertation, we argue that Software-Defined Networking (SDN) form propitious environments for the design and implementation of more robust and extensible anomaly classification schemes. Different from other approaches from the literature, which individually tackle either anomaly detection or classification or mitigation, we present a management framework to perform these tasks jointly. Our proposed framework is called ATLANTIC and it combines the use of *lightweight techniques* for traffic monitoring and *heavyweight*, but accurate, techniques to classify traffic flows. As a result, ATLANTIC is a flexible framework capable of categorizing traffic anomalies and using the information collected to handle each traffic profile in a specific manner, *e.g.,* blocking malicious flows.

Keywords: Anomaly detection. Traffic classification. Traffic monitoring. Management framework. Machine learning.

ATLANTIC: Um Framework para Detecção, Classificação e Mitigação de Tráfego Anômalo em SDN

RESUMO

Software-Defined Networking (SDN) objetiva aliviar as limitações impostas por redes IP tradicionais dissociando tarefas de rede executadas em cada dispositivo em planos específicos. Esta abordagem oferece vários benefícios, tais como a possibilidade de uso de protocolos de comunicação padrão, funções de rede centralizadas, e elementos de rede mais específicos e modulares, tais como controladores de rede. Apesar destes benefícios, ainda há uma falta de apoio adequado para a realização de tarefas relacionadas à classificação de tráfego, pois (*i*) as características de fluxo nativas disponíveis no protocolo OpenFlow, tais como contadores de bytes e pacotes, não oferecem informação suficiente para distinguir de forma precisa fluxos específicos; (*ii*) existe uma falta de suporte para determinar qual é o conjunto ótimo de características de fluxo para caracterizar um dado perfil de tráfego; (*iii*) existe uma necessidade de estratégias flexíveis para compor diferentes mecanismos relacionados à detecção, classificação e mitigação de anomalias de rede usando abstrações de software; (*iv*) existe uma necessidade de monitoramento de tráfego em tempo real usando técnicas leves e de baixo custo; (*v*) não existe um framework capaz de gerenciar detecção, classificação e mitigação de anomalias de uma forma coordenada considerando todas as demandas acima. Adicionalmente, é sabido que mecanismos de detecção e classificação de anomalias de tráfego precisam ser flexíveis e fáceis de administrar, a fim de detectar o crescente espectro de anomalias. Detecção e classificação são tarefas difíceis por causa de várias razões, incluindo a necessidade de obter uma visão precisa e abrangente da rede, a capacidade de detectar a ocorrência de novos tipos de ataque, e a necessidade de lidar com erros de classificação. Nesta dissertação, argumentamos que SDN oferece ambientes propícios para a concepção e implementação de esquemas mais robustos e extensíveis para detecção e classificação de anomalias. Diferentemente de outras abordagens na literatura relacionada, que abordam individualmente detecção ou classificação ou mitigação de anomalias, apresentamos um framework para o gerenciamento e orquestração dessas tarefas em conjunto. O framework proposto é denominado ATLANTIC e combina o uso de técnicas com baixo custo computacional para monitorar tráfego e técnicas mais computacionalmente intensivas, porém precisas, para classificar os fluxos de tráfego. Como resultado, ATLANTIC é um framework flexível capaz de categorizar anomalias de tráfego utilizando informações coletadas da rede para lidar com cada perfil de tráfego de um modo específico, como por exemplo, bloqueando fluxos maliciosos.

Palavras-chave: Detecção de anomalias, classificação de tráfego, monitoramento de tráfego, framework de gerenciamento, aprendizagem de máquina.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

DHCP	*Dynamic Host Configuration Protocol*
ICMP	*Internet Control Message Protocol*
SVM	*Support Vector Machine*
IETF	*Internet Engineering Task Force*
SDN	*Software-Defined Networking*
HTTP	*Hypertext Transfer Protocol*
DDoS	*Distributed Denial of Service*
API	*Application Programming Interface*
IP	*Internet Protocol*
REST	*Representational State Transfer*
JSON	*JavaScript Object Notation*
PCA	*Principal Component Analysis*
GA	*Genetic Algorithm*
DFT	*Discrete Fourirer Transform*
MAC	*Media Access Control*
TCP	*Transport Control Protocol*
QoS	*Quality of Service*
RFC	*Request For Comments*

CONTENTS

1 INTRODUCTION

Resilience is the ability of the network to maintain an acceptable level of service when confronted with operational challenges (STERBENZ et al., 2010). A challenge is an atypical event that hinders the expected normal network operation (CETINKAYA; STERBENZ, 2013) (SCHAEFFER-FILHO et al., 2014). When a network challenge arises, resilience mechanisms should be activated, ideally without human intervention, to rapidly protect a network and the services it supports. However, the broad range of potential challenges that could befall a network requires sophisticated network (resilience) management systems that can detect and mitigate their effects (SCHAEFFER-FILHO et al., 2014). Existing management systems have limitations, including a lack of flexibility with respect to challenge identification and classification, which has encouraged research that considers this problem in the context of new network architectures (A. LANDA R.; GEORGE, 2014).

In particular, traffic classification assists network resilience mechanisms in guaranteeing quality of service (QoS), detecting malicious attacks, reallocating network resources, and performing traffic modeling (NGUYEN; MINH; YAMADA, 2013a). Traffic classification techniques can be broken down into several domains, including Internet application protocol classification (i.e., classifying transport flows according to their corresponding application layer protocol), packet classification (i.e., categorizing packets into transport flows), and traffic classification for anomaly detection (i.e., separating malicious and non-malicious flows). In order to achieve more accurate traffic classification, it is important to retrieve precise information about individual traffic flows features, which include, for example, the average packet transmission time.

Typically, systems for anomaly traffic detection and classification need to be flexible and easy to manage in order to be able to detect a growing spectrum of anomalies. However, the majority of network functions in traditional IP networks are coupled with network forwarding devices, thus creating a distributed control plane. This approach presents limitations when we consider management requirements, such as programmability, flexibility and evolution, because even simple changes would require several network devices to be updated. These limitations result in a significant impact on the performance, accuracy and management of anomaly traffic classification solutions (KIM et al., 2008). Due to their critical importance, anomaly traffic detection and classification have demanded considerable attention in the past few years (STERBENZ et al., 2010). Part of these previous research efforts have concentrated on mechanisms for monitoring and shaping network traffic. Others have proposed mechanisms to mitigate anomalies by restoring the network to normal operation (CHANDOLA; BANERJEE; KUMAR, 2009).

Software-Defined Networking (SDN) offers a reformulation of the network control logic and alleviate the limitations imposed by traditional IP networks (WICKBOLDT et al., 2015b). In SDN, the control plane is decoupled from the data plane, *i.e.*, part of the control logic is moved from the forwarding devices to a logically centralized device often referred to as the network controller. In this approach, every control decision is taken by the network controller, while network devices become simple packet forwarders, programmable through a standardized protocol, such as OpenFlow (MCKEOWN et al., 2008). SDN can facilitate the design of anomaly detection and traffic classification systems because of several reasons: (*i*) SDN offers a more comprehensive view of the network, (*ii*) SDN supports the easy collection of flow statistics, and (*iii*) SDN includes a dedicated management plane to coordinate dynamic reconfiguration actions. However, despite the benefits brought by SDN, there is still a lack of adequate research on traffic classification in OpenFlow-based networks, because: (*i*) the native flow features available in OpenFlow, such as packet and byte counts, do not convey sufficient information to accurately distinguish between some types of flows; (*ii*) there is a lack of support to determine what is the optimal set of flow features to characterize different types of traffic profiles; (*iii*) there is a need for a flexible way of composing different mechanisms to detect, classify and mitigate network anomalies using software abstractions; (*iv*) there is a need of online traffic monitoring using lightweight/low-cost techniques; (*v*) there is no framework capable of managing anomaly detection, classification and mitigation in a coordinated manner and considering all these demands.

To address these issues, we introduce the ATLANTIC (*Anomaly deTection and machine LeArNing Traffic classifICation for software-defined networking*) framework for detection, classification, and mitigation of traffic anomalies in SDN-based networks. We perform anomaly detection and classification in two complementary phases: (*i*) a *lightweight phase*, in which low computation cost methods are executed more frequently to quickly spot potentially malicious flows, and (*ii*) a *heavyweight phase*, where such spotted flows are analyzed and classified according to their abnormal behavior. To instantiate our framework, we employ an information theory method based on entropy analysis (GIOTIS et al., 2014a) in the *lightweight phase*. In the *heavyweight phase*, a set of machine learning algorithms, based on Support Vector Machines (SVM), Naïve Bayes and Neural Networks (YUAN et al., 2010a), are used to leverage historical knowledge about past anomalies and to classify the abnormal traffic. To the best of our knowledge, there is no framework capable of managing anomaly detection, classification and mitigation in a coordinated manner in SDN environments. We advocate that such a framework should perform these tasks jointly, be fully extensible to accommodate different types of anomalies, and rely on modular software abstractions.

Our main contributions are: (*i*) a strategy that obtains global network information without additional costs to network administrators, such as additional sensors; (*ii*) an architecture to combine several types of anomaly detection, classification, and mitigation techniques in a flexible manner, while avoiding high resource utilization; (*iii*) a publicly available application of how SDN can provide sophisticated software-based management solutions (represented by the lightweight and heavyweight phases) to tackle legacy network problems, such as managing classification techniques. We also have developed a prototype system as a proof-of-concept. Our prototype has been implemented in Python and is publicly available in GitHub[1]. We evaluate the instantiation of the ATLANTIC framework to manage an SDN-based environment consisting of 12 switches organized according to the topology of a campus network. In our experimental evaluation, we observed performance, accuracy, and overhead of ATLANTIC, considering distributed denial of service (DDoS) and port scanning attacks.

The remaining of this dissertation is organized as follows. In Chapter 2, we present the background and review the related work. We show basic concepts related to Software-Defined Networking, the OpenFlow protocol and issues related to network traffic classification. Additionally, we review some important related work in the anomaly detection field discussing techniques and lessons learned in the past. In Chapter 3, we introduce ATLANTIC and show a description and overview of its architectural components, as well as describe the operation of its lightweight and heavyweight phases. In Chapter 4, we outline the implementation of ATLANTIC showing how each component communicates with others and internal details of its architecture, and in Chapter 5 we present our evaluation and associated results, including a performance analysis of the framework. Finally, in Chapter 6, we conclude this dissertation presenting final remarks and future work.

[1]https://github.com/AndersonSanSilva/ATLANTIC

2 BACKGROUND AND RELATED WORK

This chapter presents some basic concepts for the understanding of this work. Software-Defined Networking, the OpenFlow protocol and the notion of Feature Selection are introduced, along with concepts related to traffic classification. After that, the related work is presented and discussed, exposing the major differences in comparison with the ATLANTIC framework.

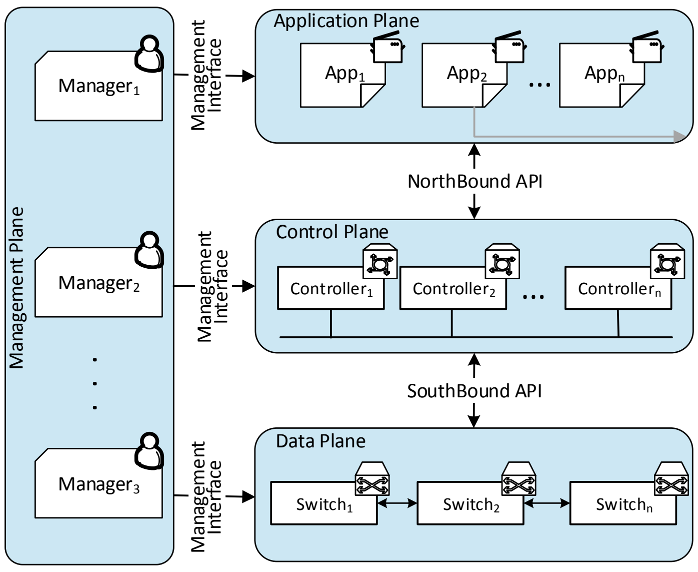
2.1 Software-Defined Networking

Software-Defined Networking (SDN) is an architecture for computer networks aimed at decoupling the network control functions (*control plane*) from the forwarding devices (*data plane*) (FEAMSTER; REXFORD; ZEGURA, 2013a). The *control plane* is responsible for determining the network control logic, such as implementing routing protocols. The aim of the SDN architecture is to simplify the deployment of new control plane functions, such as routing strategies, when compared to traditional networks (JARRAYA; MADI; DEBBABI, 2014; CUI et al., 2014), in which the control and data planes are more tightly coupled and typically operate in an entirely distributed fashion.

2.1.1 Architecture Overview

The SDN architecture defines four conceptual planes and communication interfaces as depicted in Figure 2.1.

Figure 2.1: SDN architecture: conceptual planes and communication interfaces



Source: by author (2015).

- The *application plane* is responsible for executing applications that run over the network infrastructure. Generally, these applications perform modifications regarding network aspects, such as network policies and routing behavior, with some degree of human intervention (JARRAYA; MADI; DEBBABI, 2014). Examples of network applications deployed in this plane are network visualization, path reservation and network provisioning;
- The *control plane* defines the control logic, such as routing schemes. Additionally, the control plane can manage the information collected by switches at the *data plane*, such as flow statistics, to orchestrate the traffic behavior. This plane has a global network view, being able to offer mechanisms for fault diagnosis, make decisions over current traffic distributions and enforce QoS policies. Usually, the *control plane* is physically distributed into *controller* devices, but logically centralized (BERDE et al., 2014);
- The *data plane* includes the devices that are responsible for forwarding data, which are generally referred to as *switches*. An OpenFlow switch offers the notion of programmable flow tables, *i.e.*tables that define an action for each packet associated with a specified flow. A flow table can be dynamically configured by the *control plane*. When a new packet arrives in a given switch it can be (*i*) dropped; (*ii*) flooded through all output ports; (*iii*) sent to a specific output port; or (*iv*) sent to the network controller (Open Networking Foundation, 2009). For every flow the switches involved in this communication store statistical information that can be accessed by the *control plane*.
- The *management plane* is responsible for monitoring, configuring and maintaining the behavior of network elements in each plane. The management focuses on the configuration of these network elements. Consequently, some human intervention may be necessary for the managed applications in this plane. Examples of applications are resource allocation, enforcing access control policies and VNF (Virtual Network Function) managers.

Furthermore, the communication between the different planes occurs through the following interfaces:

- *Northbound API*: Implements the communication interface between the *control plane* and the *application plane*. This API enables the programmability of the network controller by exposing network data abstractions to the *application plane*. Currently, the most used protocol for this communication is REST (REpresentational State Transfer);
- *Southbound API*: Implements the communication interface between the *control plane* and the *data plane*. Through this interface it is possible for the *control plane* to configure switches with forwarding actions according to received notifications of incoming packets from the *data plane* (FARHADI; DU; NAKAO, 2014). This is typically standardized and implemented by the OpenFlow protocol (MCKEOWN et al., 2008; AMAZONAS; SANTOS-BOADA; SOLÉ-PARETA, 2014).
- *Management Interface*: is responsible for providing information exchange between net-

work management solutions and the elements in all other planes. Examples of protocols to do this task are OF-Config and NETCONF (WICKBOLDT et al., 2015a).

It can be seen that through these interfaces the SDN architecture introduces a great deal of flexibility in flow management, impacting directly in areas such as security, traffic management and performability (NAKAYAMA et al., 2014; PANTUZA et al., 2014). Also, SDN has the potential to reduce the cost of network deployment, because simplified data plane switches are relatively inexpensive components, when compared to more complex routers (GREENBERG et al., 2008). Furthermore, OpenFlow has proven to be ideal for the development of prototype network applications (LARA; KOLASANI; RAMAMURTHY, 2014); based on this success, research in this field has increased (SCHIFF; BOROKHOVICH; SCHMID, 2014). These characteristics generated enthusiasm in both industry and academia. Many surveys covering historical aspects, architecture and challenges related to SDN have been published and further discussions can be found in (CASADO et al., 2012; CARAGUAY et al., 2013; HU; HAO; BAO, 2014; JARRAYA; MADI; DEBBABI, 2014).

2.1.2 OpenFlow Protocol

The OpenFlow protocol was proposed as a platform to enable innovation in computer networks. This protocol provides the communication between the control and the data plane (OpenFlow switch). An OpenFlow switch offers the notion of programmable flow tables, *i.e.*, tables that define an action for each packet associated with a specified flow. A flow table can be dynamically configured by the control plane. When a new packet arrives in a given switch it can be (*i*) dropped; (*ii*) flooded through all output ports; (*iii*) sent to a specific output port; or (*iv*) sent to the network controller (Open Networking Foundation, 2009). For every flow the switches involved in this communication store statistical information that can be accessed by the control plane. Consequently, the OpenFlow switch is a device that is responsible for forwarding data, however can do additional tasks such as the collection of flow counters. The basic set of flow counters defined by the OpenFlow specification (Open Networking Foundation, 2009) is represented by:

$$\langle packet\_count, duration, byte\_count\rangle^1$$

The Openflow protocol is the key implementation of SDN ideas and enables the handling of traffic flows with a massive use of wildcards to efficiently configure switches with the forwarding behavior for incoming packets. These wildcards are used to specify *don't care* packet header fields to filter or match a subset of network packets. A benefit of this practice is the low TCAM memory usage since several traffic flows can use the same switch forwarding policy, which enhances performance issues in SDN networks (CURTIS et al., 2011).

[1]OpenFlow protocol version 1.0. New flow counters can be added in future versions of this protocol.

However, traffic flow information, such as individual IP address, can be lost in this process, thus hindering the detection of malicious activities and facilitating its propagation. Still, despite all the benefits that the OpenFlow protocol represents, the reduced number of flow counters, the absence of conditional match rules, and the inability to force TLS encryption in its communication with the network controller are examples of current limitations that this protocol faces.

2.2 Traffic Classification

Traffic classification techniques are capable of identifying patterns in the sampled network traffic. Their purpose ranges from the identification of malicious traffic up to the categorization of Internet traffic for QoS support. However, with the increasing sophistication of applications, protocols and traffic profiles, strategies based on port numbers do not offer reliable classification. Further, strategies based on *payload* inspection can be very accurate if packets are not encrypted, but at a high processing cost.

2.2.1 Classification techniques

Because of the emergence of new traffic behaviors, strategies to detect and classify anomalous traffic are necessary so to protect the network against malicious attacks. In traditional networks, machine learning has been widely used for traffic classification and anomaly detection (NGUYEN; ARMITAGE, 2008). These techniques can be divided into two main classes:

- *Supervised learning:* supervised learning techniques, such as Naive Bayes (MOORE; ZUEV, 2005a) and Support Vector Machine (SVM) (YUAN et al., 2010b), are suitable for classifying data samples into a range of known attacks because these techniques use a data model that describes the known classes to classify data. However, supervised learning is unable to handle new types of attacks when the model used does not have a sample of these attacks. SVM typically achieves high classification accuracy (NGUYEN; ARMITAGE, 2008) and thus it is commonly chosen to compose anomaly detection systems.
- *Unsupervised learning:* unsupervised learning techniques, such as K-means (ERMAN; ARLITT; MAHANTI, 2006) and Expectation Maximization (NGUYEN; ARMITAGE, 2008), are suitable for detecting new types of attacks. This occurs because these techniques do not use historical information or a data model to produce the data clustering, but only the similarities observed in the data. In this way, a new data profile can be clustered with similar old data samples or be clustered in a single group. However, in general, they need human input to determine the classes of the clustered data.

Despite the high accuracy and performance obtained with these techniques, machine learning algorithms tend to suffer from several limitations: *(i)* the difficulty of determining the best set of discriminators to classify flows (MOORE; ZUEV, 2005a); *(ii)* the availability of labeled training data for classification (ERMAN; ARLITT; MAHANTI, 2006) (WILLIAMS; ZANDER; ARMITAGE, 2006); *(iii)* the trade-offs between different machine learning algorithms regarding accuracy and performance (WILLIAMS; ZANDER; ARMITAGE, 2006); *(iv)* the sheer amount of traffic data that makes it difficult to handle and to promptly detect malicious activities (LAKHINA; CROVELLA; DIOT, 2005) (WANG; GOMBAULT, 2008); *(v)* the need of a high amount of resources, such as management systems and middleboxes, to collect traffic information (YU et al., 2011). Further discussion on machine learning and its use for network traffic classification is presented by Nguyen and Armitage (NGUYEN; ARMITAGE, 2008).

Additionally, techniques based on Information Theory have also been used in traditional networks for anomaly traffic detection (LAKHINA; CROVELLA; DIOT, 2005) (GIOTIS et al., 2014b). These techniques use probability and statistic theory to model the entropy, *i.e.*, the mean information present in some set of traffic features, to detect when disturbances occur in the network. In particular, entropy can be used to model a high-level view of the flows observed in the network, and enables the monitoring of distributions of flow features with reduced computational cost. By analyzing the entropy information within a time interval it is possible to detect deviations that indicate an anomaly. Past research efforts indicate that entropy is a suitable, low-cost, and accurate technique to monitor traffic behavior changes (NYCHIS et al., 2008). Moreover, the combination of entropy and machine learning can be used for traffic classification (AGARWAL; MITTAL, 2012).

2.2.2 Feature Selection

In the context of machine learning, the problem of discovering the optimal set of features to describe input data has been studied for several years (KLOFT et al., 2008). This process is named feature selection and often represents a problem because (*i*) there is a lack of a priori information about the relevance of features collected to describe some data, thus leading to inaccurate classifications; (*ii*) the excessive number of collected features can lead to a classification with high computational cost; (*iii*) there is a lack of information on the combination between different data features, thus encouraging the study of their joint influence. Further, there is a wide range of applications that can benefit from this process, such as anomaly detection systems and QoS enforcement applications, which have boosted the study of strategies to solve this problem.

It is desirable that irrelevant or redundant features should be removed in the feature selection process and the remaining set should represent enough information for an accurate traffic classification. Strategies for selecting flow characteristics are the subject of intense study, as evidenced by (PASCOAL et al., 2012; MANTERE; SAILIO; NOPONEN, 2012). The work of Blum *et al.* (BLUM; LANGLEY, 1997) summarizes several solutions to the feature selection problem, such as heuristic search, filter-based strategies and solutions based on assigning weights to each feature.

The techniques for feature selection can be broken down in the following classes:

- **Filter methods:** use variable ranking techniques such as Principal Component Analysis (PCA) to determine a feature ordering. In this way, these methods can determine the set of less relevant features that can be ignored in the feature selection process.
- **Wrapper methods:** use a predictor, such as a machine learning classifier, to evaluate each subset of features generated. The task of evaluating all possible subsets of features is impracticable, thus this type of method in general generates a suboptimal feature set. Examples are Sequential Selection algorithms and Heuristic Search methods.
- **Embedded methods:** aim to reduce the computation time used in wrapper methods. The idea behind this is to incorporate feature selection in the training process of Wrapper methods. The strategies for this can use greedy algorithms or the assignment of weights for the classifier and its subset of features.

The work of Fahad *et al.* (FAHAD et al., 2013) studies algorithms based on mathematical concepts, such as eigenvalues and eigenvectors for selecting data features. For example, algorithms, such as *gain information (IF)*, *gain ratio (GR)*, *principal component analysis (PCA)*, *correlation based feature selection (CBFS)*, *Chi- square*, *consistency-based-search (CBS)* can be used to provide feature selection. The quality of each algorithm is generally studied using metrics such as stability (select the same set of characteristics even in the studied traffic variations) and similarity (how similar are the sets of optimal characteristics found by each algorithm). Fahad *et al.* still concludes that the use of these techniques together increases the quality of the final set of features achieved. In the class of feature selection strategies using heuristic methods, the work of Lanzi *et al.* (LANZI, 1997) is intended to resolve the feature selection problem using heuristic algorithms, such as the genetic algorithm, whose optimal solution is found in an indeterminate number of controlled iterations.

2.3 Related Work

Recent research efforts have indicated that SDN is suitable for the implementation of sophisticated software solutions and that anomaly detection schemes can benefit from the SDN architecture (BRAGA; MOTA; PASSITO, 2010) (ZHANG, 2013) (SHIN et al., 2013). However, some key issues still remain open, such as the limited number of traffic features offered by OpenFlow, the orchestration of classification mechanisms to avoid network performance degradation, and the need of an extensible anomaly detection mechanism that can evolve and be modified to treat new types of network anomalies. In the next subsections, we show the principal studies on these topics.

2.3.1 Existing Solutions for Anomaly Detection and Traffic Classification in SDN

According to Mehdi *et al.* (MEHDI; KHALID; KHAYAM, 2011), the deployment of an anomaly detection system in the traditional network core is difficult mainly due to the low detection rate that these systems can provide with limited network information. In SDN, however, the *control plane* has a comprehensive view of the network, which facilitates the implementation of detection mechanisms. Mehdi *et al.* (MEHDI; KHALID; KHAYAM, 2011) presents an overview of attack detection possibilities using SDN. The authors discuss four well-known algorithms: TRW-CB, MaxEnt, RateLimit and NETAD. Their study suggests that SDN is a platform suitable for the mitigation of DDoS attacks, mainly because of the use of standard protocols, services and interfaces, thus facilitating the deployment of new solutions.

Braga *et al.* (BRAGA; MOTA; PASSITO, 2010) investigate the mitigation of DDoS attacks using Self-Organizing-Maps (SOM), a machine learning algorithm already used in traditional networks but with limited effects due to the restrictions of that architecture. The authors propose a solution based on flow collection, feature extraction and flow classification. The traffic features used are the *average number of packets per flow*, *average bytes per flow*, *average duration per flow*, *percentage of pair-flows*, and *growth of single-flows*. However, the feature selection step is not performed. In the concluding remarks, the authors highlight that the detection rate obtained is remarkably good and that the flexible composition/communication of different detectors is an aspect needed in this context.

McHale *et al.* (MCHALE et al., 2014) propose a packet classification scheme based on prefix match and flow cache techniques to avoid the repeated classification of traffic flows. Its classification scheme is based on the idea of flow locality, *i.e.*, the idea that 35% of flows contain 95% of the incoming packets and that SDN is an environement that will increase the stress on packet classification. Using these assumptions, they take advantage of flow locality to provide an stochastic flow classification mechanism using flow caches and strategies for pre-classification. They conclude that the use of flow cache mininizes the effect of repeatedly classifying high-throughput flows and that the pre-classification prioritizes known traffic. This research represents an example of the need for flexible schemes to monitor traffic in real-time.

The use of information theory for packet classification has been investigated by Giotis *et al.* (GIOTIS et al., 2014a), who use entropy analysis for monitoring deviations in network behavior. The authors propose an sFlow-based mechanism to obtain information from the network. This mechanism is coupled with a modular architecture that separates data collection from other processing overloads that the centralized control-plane should perform, such as routing. The architecture is composed of a flow collector, an anomaly detection component responsible for analyzing flow statistics/anomaly identification, and an anomaly mitigation component responsible for changing flow rules in order to block malicious flows.

2.3.2 Discussion

As described in the previous section, there is a number of research efforts investigating anomaly detection and traffic classification in SDN environments. However, as pointed out in Chapter 1, SDN/Openflow lacks support to:

- An *extended set of flow features* to convey sufficient information to accurately distinguish between some types of flows. This extended set of flow features can be generated through traffic information collected from the data plane and the use of statistical techniques over primitive traffic counters collected;
- *Feature selection* schemes to determine what is the optimal set of flow features to characterize different types of traffic profiles;
- *Flexible composition of several mechanisms to detect, classify and mitigate network anomalies* using software abstractions;
- The easy implementation of *lightweight/low-cost techniques for online traffic monitoring* in order to alleviate the overhead imposed on the network;
- A *framework capable of managing anomaly detection, classification and mitigation* in a coordinated manner and considering all these demands;

Table 2.1 compares the related work discussed in the previous section and the criteria above. Note that the related work does not include research about Feature Selection.

Table 2.1: Anomaly detection requirements and related work

Feature	**Mehdi *et al.***	**Braga *et al.***	**McHale *et al.***	**Giotis *et al.***
Extended flow features	✓	✓	✓	x
Feature selection	x	x	x	x
Flexible composition of several mechanisms to detect, classify and mitigate network anomalies	✓	x	✓	✓
Lightweight/low-cost online traffic monitoring	x	✓	x	✓
Framework capable of managing anomaly detection, classification and mitigation	✓	✓	x	✓

Source: by author (2015).

Differently from the related work discussed in this section, we propose a framework that jointly coordinates anomaly detection, classification and mitigation tasks and assists the network administrator with traffic classification related tasks, such as feature selection. These issues are addressed by the ATLANTIC framework, which will be described in the next chapter.

3 ATLANTIC: CONCEPTUAL SOLUTION

We advocate that anomaly detection and traffic classification can take advantage of the programmability offered by SDN/OpenFlow. To demonstrate this, we introduce ATLANTIC, an anomaly detection, classification and mitigation framework that allows an administrator to flexibly reconfigure the operation of its building-block components and algorithms. In this chapter, we discuss the general principles behind our framework. In addition, we present ATLANTIC in details and describe its main components. In particular, we show an overview of our traffic classification process and discuss our proposed architecture.

3.1 Framework General Principles

A framework for anomaly detection and traffic classification should be capable of orchestrating several different modules, such as those responsible for traffic monitoring, classification, and mitigation. We argue that the required functionality for such a framework can be placed in the management plane of the SDN architecture, and take into account the following aspects:

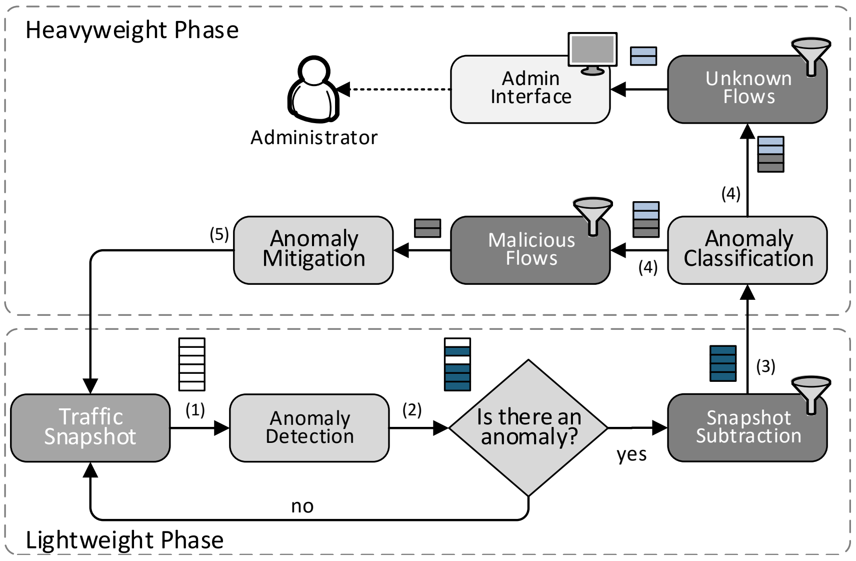
- **Comprehensive view of the network -** To perform traffic monitoring and analysis, the framework must be able to retrieve detailed and unrestricted information about the network and traffic flows. As opposed to applications sitting in the application plane of SDN – which makes use of the Northbound API to request network resources to the controller – our framework uses the Management Interface to gain access to information about flows from all applications, and uses this information to manage traffic anomalies.
- **Human intervention -** The network administrator must be able to interact and monitor the operation of the anomaly detection and traffic classification framework, observing logs and reconfiguring its operation whenever necessary. For example, an administrator might update parameters or replace some component functionality to increase performance or accuracy of classification.
- **Flexible network configuration -** Several types of configurations can assist the task of anomaly mitigation, such as the definition of proactive and reactive path instantiation, deployment of specific or generic flow rules in flow tables, and management of flow parameters such as timeout and data rate. Our framework must be able to instruct the network controller to change its behavior regarding certain events and flows as they are deemed anomalous.

We anticipate that our management framework must be modular and support customization, *i.e.*, it should be possible to update components with a more sophisticated algorithm or strategy whenever needed. For example, a *network driver* (see Section 3.3) may need to be customized to collect information from different types of individual network controllers or from a large set of distributed controllers. Using the management plane, these decisions can be taken by administrators to achieve more appropriate configurations.

3.2 Lightweight and Heavyweight Processing

ATLANTIC comprises two operational phases: a *lightweight processing phase*, responsible for traffic monitoring and anomaly detection; and a more *heavyweight processing phase*, consisting of anomaly classification and mitigation. Next, we explain these phases in details and how they are combined to support robust anomaly management. Figure 3.1 summarizes the interplay between the lightweight and heavyweight processing phases.

Figure 3.1: Framework management process



Source: by author (2015).

3.2.1 Lightweight Processing Phase

Several techniques can be used to extract network traffic profiles, for example, by performing packet sampling using sFlow (GIOTIS et al., 2014b). A limitation of this approach is the high memory consumption to obtain fine-grained traffic information and packet inspection. To rapidly perform lightweight anomaly detection, our framework benefits from the characteristics of SDN and uses the control plane to obtain a snapshot of existing traffic flows, including information about traffic counters and matched packet headers. Based on the traffic snapshot collected (arrow 1 in Figure 3.1), lightweight anomaly detection mechanisms are employed to detect deviations from the "normal" traffic pattern.

We apply entropy analysis to detect variations in the distribution of certain flow features observed along consecutive traffic snapshots. For example, consider two consecutive snapshots

t_1 and t_2. If the entropy of the flow features in t_1 is approximately equal to the entropy of the same features observed in t_2, then it is safe to assume that no significant traffic changes have occurred between the two consecutive snapshots. However, if there is a large difference in the entropy calculated for a given flow feature between two snapshots, this might indicate an anomaly (arrow 2). If by subtracting the flows in t_2 from the flows in t_1 we obtain a non-empty result, this indicates which flows are responsible for the entropy change. Flows that are responsible for the entropy change in this stage can only be considered suspicious, and are thus selected for further categorization using a proper classification scheme (arrow 3). It is important to emphasize that our framework is designed so that any snapshot-based anomaly detection scheme can be employed. We chose to use entropy analysis for the *Lightweight Processing Phase* because it can be executed very often and permits fast detection of disturbances in the network (GIOTIS et al., 2014b).

Considering that information is associated with a process of symbol selection, the amount of information associated with one symbol should be inversely proportional to the symbol occurrence (NYCHIS et al., 2008). As the logarithm function fits the restriction showed above, the equation to represent information is:

$$I(x_i) = log_2 \frac{1}{p_i} \tag{3.1}$$

If we consider the quantity of information Q associated with one message M with k symbols, this is represented by the following equation:

$$Q(M) = \sum_{i=1}^{k} -log_2 p_i \tag{3.2}$$

Where p_i represents the probability of occurrence of symbol i. One alphabet is a set of symbols, and the mean information associated with an alphabet X is called *entropy*, which is calculated as follow:

$$H(X) = -\sum_{i=1}^{N} p_i * log_2 p_i \tag{3.3}$$

The higher the entropy of an alphabet, the higher the average information generated. Thus the quantity of information associated with one source depends on the entropy of the alphabet used.

We use the following information from the traffic snapshot to calculate the entropy associated with a flow:

$$\langle srcip, dstip, srcport, dstport, protocol \rangle$$

We chose to calculate the entropy based on IP address and transport port features because they have been demonstrated to be accurate for the detection of DDoS attacks and worm propagations (BRAGA; MOTA; PASSITO, 2010).

3.2.2 Heavyweight Processing Phase

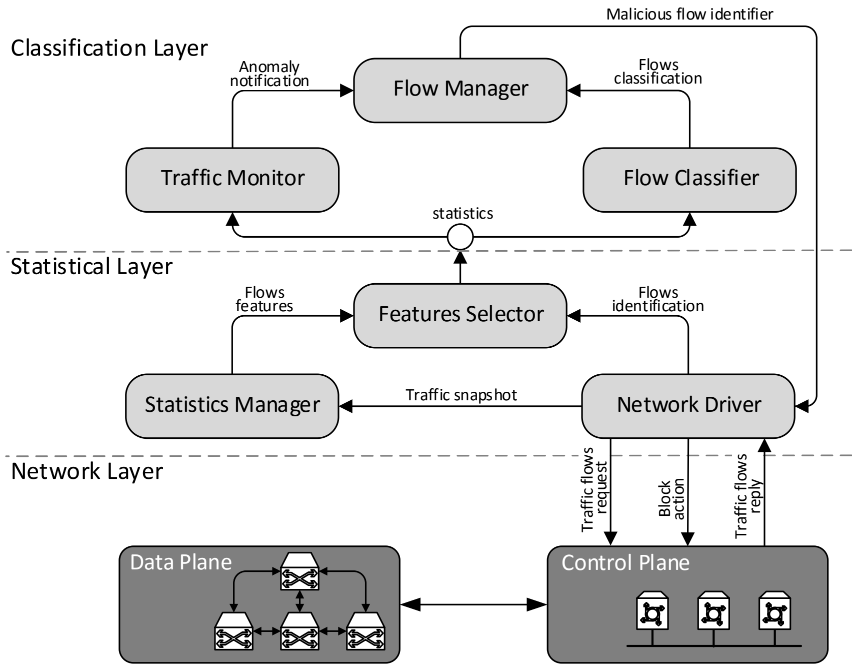
Our framework comprises the occasional need to execute complementary heavy processing classification mechanisms to categorize traffic flows. We consider traffic classification mechanisms based on machine learning, which indeed take a considerable amount of resources to execute but can produce very accurate results in terms of traffic classification (NGUYEN; ARMITAGE, 2008). Our framework allows both supervised and unsupervised machine learning mechanisms. With supervised mechanisms, a model is generated to internally organize data obtained from previous malicious activity to automatically categorize traffic flows as either malicious or benign. Unsupervised mechanisms, on the other hand, are interesting to be used to analyze and organize information about traffic features, even if they cannot identify whether there is a threat or not. As a result, the framework allows flows to be categorized into either malicious, benign, or unknown, according to their traffic profiles. Malicious flows are sent for mitigation, whereas unknown flows need to be manually analyzed by a human administrator (arrow 4, Figure 3.1).

For every flow that is signalized as malicious, an action must be taken so to avoid network disruption or performance degradation. For example, commands can be sent back to the network control plane to instruct the devices closer to the source of the malicious traffic to drop packets of flows deemed malicious. After mitigation actions are performed, the framework returns to its initial traffic monitoring and snapshot collection step (arrow 5, Figure 3.1). For flows signalized as unknown, the administrator can use the information obtained during classification, for example, to create new models for the supervised mechanisms to identify the new traffic pattern in a future round of anomaly detection. Note that this heavyweight phase is expected to be executed less frequently than the lightweight one. In addition, heavy classification mechanisms only need to deal with a subset of the full traffic snapshot, because of the subtraction performed in the lightweight phase. The more the administrator interacts with the framework inserting new information about traffic patterns to be automatically identified, the more efficient the detection will be.

3.3 ATLANTIC Architectural Components

The basic components of our anomaly traffic classification framework are depicted in Figure 3.2. The *Statistical Layer* is responsible for collecting traffic flow statistics and comprises the following components: *Statistics Manager*, *Features Selector*, and *Network Driver*. The information generated by the *Statistical Layer* is delivered to the *Classification Layer*, which comprises the following components: *Traffic Monitor*, *Flow Classifier*, and *Flow Manager*. Next, we describe these components in details. In particular, we highlight that the components traffic monitor and network driver have their operation related to the lightweight phase and the components features selector, statistics manager, flow classifier, and flow manager are related to the heavyweight phase.

Figure 3.2: Overview of the anomaly classification framework



Source: by author (2015).

3.3.1 Network Driver

We named this component as "driver" due to its similarity with existing drivers in the architectures of personal computers. Its primary function is to abstract the implementation details of a broad range of network controllers in order to allow their interoperation with the ATLANTIC framework. The network driver should perform all the communication between ATLANTIC and the network. Consequently, this component can (*i*) deal with several network controllers or a single one; (*ii*) communicate with the management plane or directly with the control plane; (*iii*) be extended with several kinds of functionalities, such as routines to convert network information into a specific format.

The communication between the network driver and the controller is based on the exchange of three types of messages. The first one (*Traffic flows request*) is responsible for collecting network traffic information, the second one (*Traffic flows reply*) is responsible for receiving the information about the traffic, and the third one (*Block action*) is responsible for installing policies on the network in order to treat flows that deserve attention.

Consequently, the Network Driver operates by sending a request to the network controller every time that other modules need to query the status of flows in the data plane. After receiving flow information from the network, the *Network Driver* parses and organizes relevant data, such as the flow identifier, packet headers, and flow counters. The result of one request produces a traffic snapshot, *i.e.*, a data structure summarizing all flows currently existing in the network. Note that we construct a *flow-id* using the following 5-tuple, which is also used as our flow definition:

$$\langle srcip, dstip, srcport, dstport, protocol \rangle$$

We consider this flow as a 5-tuple consisting of source IP address, destination IP address, TCP/UDP source port, TCP/UDP destination port, and transport protocol identification. It is important to emphasize that this tuple can be adjusted according to the needs of each network infrastructure. Thus, some approaches can ignore information such as destination TCP/UDP port when the network infrastructure performs only one type of service, *e.g.*, HTTP.

A traffic snapshot conveys information about flows as exemplified below:

Table 3.1: Traffic Snapshot main information

Flow	**Switch**
priority	0
duration nanoseconds	484000000
hard timeout	0
idle timeout	5
actions	drop or send to controller or ...
duration seconds	4
byte count	196
table id	0
packet count	0
cookie	900071..
match	src ip, dst ip,...
...	...

Source: by author (2015).

Note that the ***match*** entry has the header information of packets belonging to this flow.

3.3.2 Statistics Manager

After receiving the last saved traffic snapshot produced by the *Network Driver*, the *Statistics Manager* extracts a primitive set of flow features that describe the profile of network flows, *i.e.*, all types of traffic features that can be obtained from the network without additional processing. It comprises a main internal procedures responsible for handling network information and for computing additional flow features. Note that the implementation of this component does not require modifications to the SDN controller. Further, this component can communicate with other SDN applications (*e.g.*, monitoring or management systems) in order to export its information.

The process behind its execution is as follows. The Network Driver sends to the controller a request for traffic information. Note that the time interval for this request can be configured on demand. For example, flows with short duration and frequent bursts require a smaller time interval between requests, while a longer time interval can be used for flows with longer duration and more constant behavior. Triggered by the traffic information request, the control plane then gathers the traffic information from the flow tables of each switch and replies to the Network Driver. The traffic snapshot information is then sent to the Statistics Manager to select only the native counters, *i.e.*, byte and packet counters. The Statistics Manager uses the native counters gathered from the switches to calculate new flow features (*e.g.*, packet length mean), extend the flow features, and store these in a data structure that we call *Flow Feature Set*.

The extended set of flow features is classified into three categories: (*i*) *statistical features* – mean, variance, first, and third quartiles; (*ii*) *scalar features* – maximum and minimum values, flow size, and flow duration; and (*iii*) *complex feature* – Discrete Fourier Transform (DFT) of packet inter-arrival-time. We advocate that these extended flow features can improve the performance of classification schemes in SDN. First, the statistical features are suitable to summarize the temporal behavior of traffic profiles. For example, the mean packet count represents the central value of a set of observations. Thus, they are better descriptors in comparison to SDN native packet counters. Second, scalar features are convenient to indicate the instantaneous profile of traffic flows, such as duration. As a result, they are important in the detection of short-lived communications or packet bursts, since statistical features alone are not sensitive to these traffic profiles. Finally, complex features are useful to anticipate the need for more sophisticated traffic information. Different from the previous categories, complex features can be used to compress a wide range of traffic observations. For example, Discrete Fourier Transform can be used to refine several measurements of packet's inter-arrival-time into a set of frequency components, which is a more distinguished representation. Some of these features were investigated by Auld *et al.* (AULD; MOORE; GULL, 2007) and Moore *et al.* (MOORE; ZUEV, 2005b).

The main reason for calculating these extended traffic flow features is that OpenFlow is currently the most important protocol for SDN implementation (FEAMSTER; REXFORD; ZEGURA, 2013b), however it offers a limited number of flow features to describe traffic behavior. Although traffic classification can be performed using the native traffic counters provided by OpenFlow (*e.g.*, packet counts), this approach presents several limitations, since these counters do not allow detecting atypical traffic behavior, such as packet bursts. Because of the wide range of possible traffic profiles, more descriptive traffic discriminators are necessary. For example, we introduce a third type of counter, named *samples counter*, which indicates the number of times each flow appears during feature polling. This counter is maintained by the Statistics Manager and is essential to compute some of the more advanced flow features, as discussed below.

Consequently, OpenFlow's native counters were used as the basis for deriving two new flow features, namely packet length P_l and packet inter-arrival-time[1] P_{it}. These are estimations of

[1]We use packet length and packet inter-arrival-time statistics because individual values do not usually give substantial information about the traffic profile.

the real packet length and real inter-arrival-time, since calculating their precise values would require deep packet inspection and the monitoring of every packet in the network, which are expensive tasks. Consequently, we apply sampling strategies to estimate these features and use Equation 3.4 and Equation 3.5 to calculate packet length and inter-arrival, respectively. B_c represents byte count, P_c represents packet count between two requests, and T is the time-interval between two requests.

$$P_l = \frac{B_c}{P_c} \tag{3.4}$$

$$P_{it} = \frac{T}{P_c} \tag{3.5}$$

To calculate mean and variance, the Statistics Manager updates the mean μ and the variance σ^2 for the flow features every time a new information request is processed. Mean is calculated using Equation 3.6, where μ_n is the updated mean, μ_o is the old mean, θ is the new sample, and n is the samples counter.

$$\mu_n = \frac{n * \mu_o + \theta}{n+1} \tag{3.6}$$

The updated variance σ_n^2 is calculated using Equation 3.7, where σ_o^2 is the old variance.

$$\sigma_n^2 = \frac{n}{n+1} * (\sigma_o^2 + \frac{(\theta - \mu_n) * (\theta - \mu_o)}{n}) \tag{3.7}$$

We calculate DFT of packet inter-arrival-time samples and use the top ten components as flow features according to Auld *et al.* (AULD; MOORE; GULL, 2007) work. DFT represents the samples of a variable in the frequency domain. It is defined by Equation 3.8, where N is the number of samples, x_n is a sample, and X_k is the resulting component.

$$X_k = \sum_{n=0}^{N-1} x_n . e^{-2\pi ikn/N}, \; k \, \epsilon \, \mathbb{Z} \tag{3.8}$$

The final result is an an extended set of flow features as depicted in Table 3.2

3.3.3 Features Selector

The Features Selector summarizes all data collected by the *Network Driver* and by the *Statistics Manager* in order to determine the optimal set of flow features to use in traffic classification. In this component the process of analyzing the full set of features and selecting the optimal features subset is started. First, the Features Selector receives the flow feature set sent by the Statistics Manager, and performs two operations: (*i*) it creates a training and a test set that will be used by the Flow Classifier; and (*ii*) it organizes the full flow feature set in the specific format employed by the Features Selector. The formatted data is used as input to the feature selection

Table 3.2: Extended set comprising 33 new flow features.

Statistical Features	**Scalar Features**	**Complex Features**
Bytes per second mean	Bytes per second maximum value	Packet inter-arrival-time Fourier Transform 1^{st} Component
Bytes per second variance	Bytes per second minimum value	Packet inter-arrival-time Fourier Transform 2^{nd} Component
Packets per second mean	Packets per second maximum value	Packet inter-arrival-time Fourier Transform 3^{rd} Component
Packets per second variance	Packets per second minimum value	Packet inter-arrival-time Fourier Transform 4^{th} Component
Packets length mean	Packets length maximum value	Packet inter-arrival-time Fourier Transform 5^{th} Component
Packets length variance	Packets length minimum value	Packet inter-arrival-time Fourier Transform 6^{th} Component
Packets length 1^{st} quartiles	Packets inter-arrival-time maximum value	Packet inter-arrival-time Fourier Transform 7^{th} Component
Packets length 3^{rd} quartiles	Packets inter-arrival-time minimum value	Packet inter-arrival-time Fourier Transform 8^{th} Component
Packet inter-arrival-time mean	Flow duration	Packet inter-arrival-time Fourier Transform 9^{th} Component
Packet inter-arrival-time variance	Flow size in packets	Packet inter-arrival-time Fourier Transform 10^{th} Component
Packet inter-arrival-time 1^{st} quartiles	Flow size in bytes	
Packet inter-arrival-time 3^{rd} quartiles		

Source: by author (2015).

algorithms implemented in this module, which are able to identify the most meaningful features for flow classification. Finally, the Flow Classifier module classifies the flows using the feature subset yielded by the Feature Selector to evaluate the fitness of each subset for some traffic profile. Similarly to the Feature Selector module, the Flow Classifier also supports the implementation of a range of different algorithms.

In our proof-of-concept implementation, the Features Selector is instantiated with (*i*) *Principal Component Analysis* (PCA) and (*ii*) *Genetic Algorithm* (GA). The PCA algorithm (PEARSON., 1901) evaluates the relationships between a set of variables, determining the variability associated with each of them. PCA is used as a strategy to reduce the number of variables by removing correlated values. The result of PCA is a subset of the original variables, called principal component, which accounts for the most significant variance in the original dataset. The Genetic Algorithm mimics natural evolution and combines current best solutions for producing new solutions, which are then analyzed to assess their quality. Briefly, the algorithm creates an initial population representing a set of solutions. During each iteration, called generation, it selects the fittest solutions and creates new ones through a combination (crossover) or mutation (OH; LEE; MOON, 2004). A series of parameters must be determined a priori, such as population size and mutation probability. Although we used two well-known algorithms for realizing the Feature Selector module, we emphasize that this module can support a wide-range of other algorithms.

3.3.4 Traffic Monitor

This component is responsible for implementing the Lightweight Phase of the ATLANTIC framework. Consequently, the main objective of this component is to monitor traffic flows using their information in a lightweight way. Note that this component can use all the information collected by other components to decide network configurations. To exemplify how this component can behave, we use entropy analysis to detect changes in traffic features. In particular, this is calculated according to *Shannon's entropy* definition. Considering that a traffic snapshot is an alphabet, then the mean information $H(X)$ for some subset of features can be calculated using the available flows.Every time that a new entropy E is calculated for a given snapshot, it can be classified as anomalous in the following way: considering that M represents the mean entropy observed in the network and S the standard deviation associated, then E will be an anomalous entropy if it is not within the interval $[M - S, M + S]$.

For example, consider the following example of traffic snapshot:

flow A: { ..., match = ip_src_a, ip_dst_a, ... }
flow B: { ..., match = ip_src_b, ip_dst_b, ... }
flow C: { ..., match = ip_src_c, ip_dst_c, ... }
flow D: { ..., match = ip_src_d, ip_dst_d, ... }
...

The Traffic Monitor will collect only the information about the 5-tuple present in a match rule and will calculate the entropy associated with every field. If this traffic snapshot has the entropy associated with the *src_ip* field equals to k and this value is not within the standard deviation S, an anomaly exists in the network and is associated with *src_ip* field.

Every time a sudden change in the calculated entropy is identified, the Traffic Monitor reports this situation to the Flow Manager component. This action instructs the Flow Classifier to analyze/classify network flows in order to determine if the anomaly is malicious or not.

3.3.5 Flow Classifier

This component is responsible for implementing the Heavyweight Phase of the ATLANTIC framework. Different algorithms can be used for flow classification. When this component receives a set of feature statistics, a range of classification schemes (*i.e.*, machine learning algorithms) can run independently over the flow features. Note that this component is responsible for defining the class of each specific flow, by deciding which class is the most frequent when considering all algorithm outputs. Currently, we apply K-means (ERMAN; ARLITT; MA-

HANTI, 2006) for clustering and Support Vector Machine (SVM) (JAIN; DUBES, 1988) for classification. We consider these algorithms suitable to implement our classifier because one can complement the results offered by the other, and the union of their outputs can be easily performed. Still, the *Flow Classifier* is customizable and can be extended with additional algorithms.

Several techniques can be used to combine traffic classifiers (XU; KRZYZAK; SUEN, 1992; KITTLER et al., 1998a). For example, meta-learning means learning about learning. In practice, meta-learning takes as input results produced by learning methods and generalizes a concept over them. Meta-learning research covers the following questions: (i) how to select the most appropriate learner in a static way, (ii) how to perform this selection dynamically, and (iii) how to combine the predictions of base-level classifiers. One way to combining classifiers is using Naïve Bayes. Another way is called *stacking*, which is the process of using a classifier to learn an output class by using the classes that are the output of other classifiers. The combination of classifiers should not rely necessarily on different classification strategies. An alternative is the use of a single classification strategy, such as SVM, and to use different parameter values in order to generate alternate classifiers. A typical solution to decide about several classifiers outputs (and at the same time intuitive) is to use voting (KITTLER et al., 1998b). In future work, we intend to study how these techniques can impact on our framework.

3.3.6 Flow Manager

Events from the *Traffic Monitor* and *Flow Classifier* are sent to the *Flow Manager* to indicate if an anomaly has been identified for a specific *flow-id*. Thus, the *Flow Manager* is responsible for deciding the mitigation actions to be taken when a malicious flow is identified. We consider that a *'Malicious flow identifier'* message is sent to the *Network Driver*, indicating that the flow identified as malicious should be blocked. The *Network Driver* component further uses the *'Block action'* message to install firewall rules in the data plane and then modify how this plane handles the malicious flow. An example of action that could be taken by the *Network Driver*, as an alternative to dropping packets associated with malicious flows, is to forward such packets to another component (*e.g.*, a deep-packet inspector).

4 FRAMEWORK IMPLEMENTATION

In this chapter, we discuss the implementation choices for the ATLANTIC framework. In particular, we explain how the main architectural components and their interactions are implemented, including a discussion on traffic monitoring and traffic snapshot.

4.1 Testbed Implementation

Next we show some details about the main tools used in the implementation and execution of the ATLANTIC framework.

- **Network controller:** As network controller, we chose Floodlight[1]. Floodlight provides a modified and extended REST API that exports new types of network information, such as routing rules expiration time. Additionally, the Floodlight controller has an active community of developers that have extended the controller with communication protocols, such as new versions of OpenFlow. For these reasons, we consider Floodlight a suitable alternative for obtaining traffic snapshots in the ATLANTIC framework. Still, our traffic snapshot is exported to ATLANTIC via REST in the JSON format. This choice was driven by several reasons. First, network controllers such as Floodlight and Beacon use this format to export information. Second, REST is the most used communication interface between the application plane and the control plane (SEZER et al., 2013).
- **Network Emulator:** The Mininet[2] emulator was developed in order to offer to researchers a realistic and reliable simulation environment for implementing ideas in SDN. Widely accepted and recognized, the Mininet emulator offered the ideal simulation environment for routing solution with multiple paths.
- **OpenFlow protocol 1.0:** Although other versions of the OpenFlow protocol has been recently proposed, such as the OpenFlow 1.5, the majority of network controllers available does not provide support for these new versions. Additionally, these new versions have not proposed extensions to the basic flow counters introduced by the first version of the protocol. For this reason, we chose version 1.0 for the base implementation of the ATLANTIC framework, although other versions of the protocol might be supported with small customizations of the Network Driver component.
- **R tool:** In order to obtain a reliable implementation of machine learning algorithms we chose the R tool, a popular and broad tool for mathematical scientific analysis. The implementation of ATLANTIC relies on the following libraries of the R tool to provide data classification:
    - Support Vector Machine (SVM): *library e1071*;

[1]http://www.projectfloodlight.org/floodlight/
[2]http://mininet.org/

- Random Forests: library *randomForest, miscTools*;
- Naive Bayes: library *e1071*;
- K-means: library *cluster*;
- Expectation Maximization: library *mclust*;
- Neural Networks: library *nnet*;
- Principal Component Analysis (PCA): library *psych*;
- Genetic Algorithm: library *genalg*;

4.2 Implementation Choices

In this section, some high-level implementation choices related to the development of ATLANTIC are discussed. In particular we clarify aspects related to the network request polling time, the definition of traffic snapshots, and the use of data logs to keep track of suspicious flows.

- **Polling time:** The most critical operational aspect related to the implementation of the Network Driver component is the polling time used to obtain information from the network controller. According to Giotis *et al.* (GIOTIS et al., 2014a) an adequate time to obtain network information in an accurate and precise design is 30 seconds. However, this is an estimative reported by the authors and depend on several environmental aspects, such as communication delay, network topology, and traffic. For this reason, ATLANTIC provides the customization of this aspect.
- **Traffic snapshot:** The traffic snapshot represents all the information obtained from the network control plane. ATLANTIC does not require the modification of the network infrastructure being monitored, *i.e.*, does not require the insertion of any additional device, protocol or system in the original network substrate. Due to the popularity and intense research using the OpenFlow protocol, we chose this technology to obtain information from the network without imposing any changes on this. The OpenFlow switch enables the access of the following information about the data plane: (i) network topology provided with identifiers for hosts, switches and communication links; (ii) flows that are active and inactive on the network including packet header information; (iii) flow statistics such as transmitted packets, (iv) network device information such as, MAC addresses, IP addresses, transport layer information and others.
- **Data logs**: Even using known techniques for anomaly detection and traffic classification, we anticipate that in some cases ATLANTIC can face an unknown type of network anomaly that goes unnoticed by the models used. In this case, ATLANTIC stores log files where information of suspicious traffic, but not malicious, are reported to the network administrator. Later on, he or she can decide how to treat these flows, for example, updating the classification model with new classification information. The specific format of data

log is the same that identify a flow:

$$unknown\_flow : (srcip, dstip, srcport, dstport, protocol) :$$
$$(pkt\_count, byte\_count, duration, ...)$$

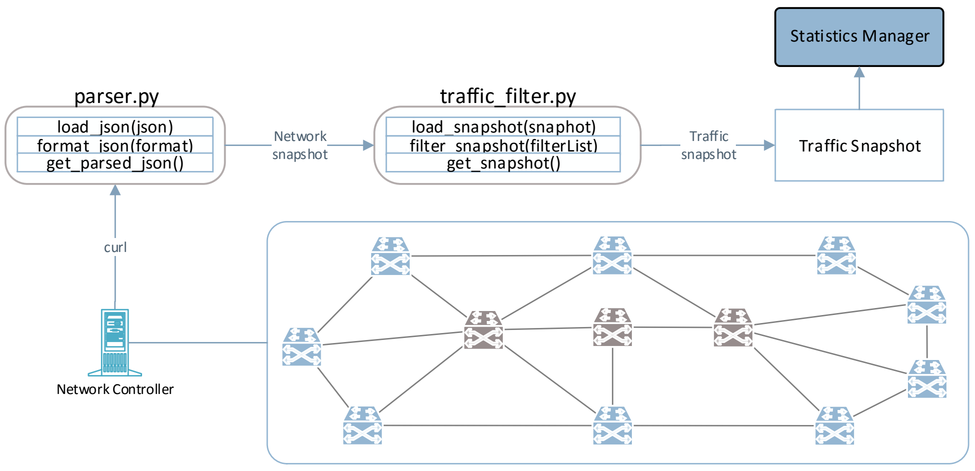
4.3 Component Interactions

In this section we describe each ATLANTIC component, discussing implementation details and the main interactions between each component. In the diagrams below, blue boxes represent entry/exit points for each component being discussed.

4.3.1 Network Driver Interactions

The Network Driver comprises three internal components responsible for collecting network information, as depicted in Figure 4.1:

Figure 4.1: Network Driver internal procedures



Source: by author (2015).

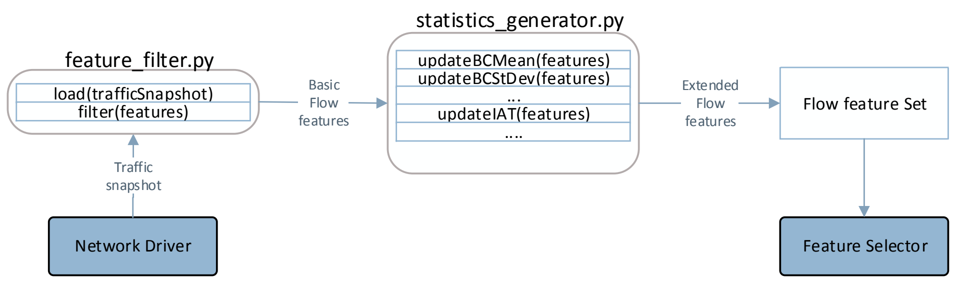
- *parser.py:* it is a parsing procedure able to understand and organize the network information exported by the Floodlight controller that uses the JSON format. Note that this parser can be extended to understand and organize other formats;
- *traffic_filter.py:* it is a filtering procedure able to remove irrelevant information exported within the network information parsed. For example, in the current implementation of ATLANTIC, the VLAN id information is not necessary for classification purposes, thus this information can be ignored;
- *Traffic Snapshot:* it is the output of this component with information about traffic flows in

a compact way regarding low memory usage. In our implementation, we use a hash-table data structure indexed by the flow id;

4.3.2 Statistics Manager and Feature Selector Interactions

The Statistics Manager operates jointly with the Feature Selector. Firstly, the statistics manager main procedures will be explained as depicted in Figure 4.2:

Figure 4.2: Statistics manager internal procedures



Source: by author (2015).

- *feature_filter.py:* this procedure is responsible for extracting only traffic counters from a traffic snapshot. Note that the OpenFlow 1.0 protocol only provides three traffic counters (packet count, byte count and duration), thus this information is collected from every flow generating a data structure with the following shape:

$$\langle (srcip, dstip, srcport, dstport, protocol) : (pkt\_count, byte\_count, duration) \rangle$$

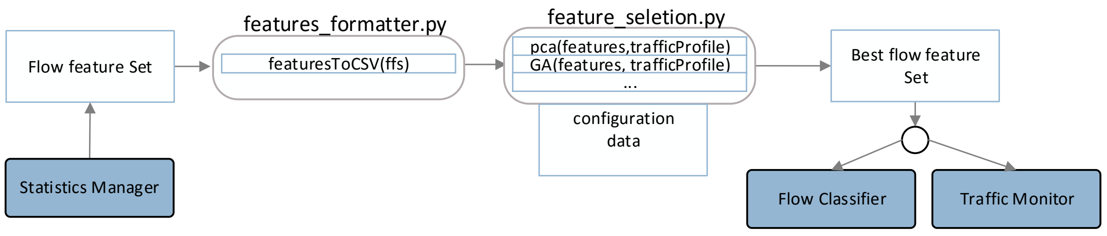
- *statistics_generator.py* this procedures extends the basic flow counter generating the extended flow features proposed in Chapter 3. The computation of these procedures applies the equations presented in Chapter 3, thus generating as final result an extended set of flow features with 33 entries.

$$\langle (srcip, dstip, srcport, dstport, protocol) : \\ (pkt\_count, byte\_count, duration, inter-arrival-time, packet\_length...) \rangle$$

After the Statistics Manager creates the extended set of flow features, the feature selector component performs the selection of the best traffic features that should be used to classify flows. Note that this component access the *configuration data* defined by the network administrator in order to determine (i) the set of traffic profiles that already have been analized by this component and (ii) the traffic profile that the network administrator is interested in analyzing. Anytime that a traffic with interest arrives in the network, the feature selection should perform the following procedures:

- *features_formatter.py:* a procedure able to convert the Flow Feature Set data structure into other formats suitable to be used as input to different types of machine learning algorithms. For example, the SVM algorithm implemented in R should receive as input a .csv file including the flow identifier and its features;
- *feature_selection.py:* this procedure performs the feature selection process as explained in Chapter 3. The algorithms PCA and Genetic are used jointly to determine the best subset of features to classify some kind of flow;
- *Best flow features Set:* it is an output that saves in the configuration data file which features are better to classify each type of flow;

Figure 4.3: Feature Selection internal procedures



Source: by author (2015).

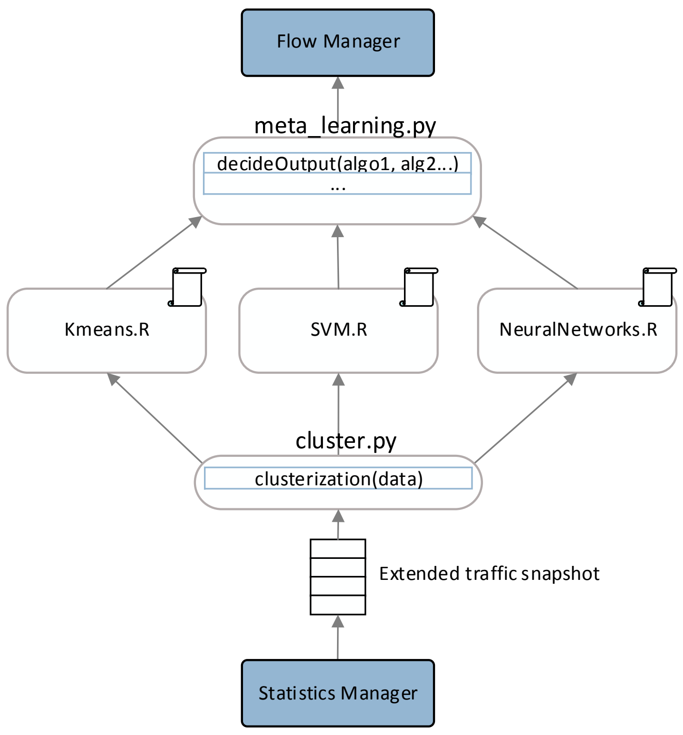
4.3.3 Flow Classifier Interactions

The flow classifier internal components include the combination of several types of supervised and unsupervised machine learning algorithms. We use the R tool to provide a reliable implementation of machine learning algorithms.

The main procedures involving this component are:

- *cluster.py:* Uses algorithms such as k-means to optimize the classification process grouping traffic samples for classification. Note that this procedure can be activated or not according to the network administrator desire and uses three unsupervised machine learning algorithms (K-means, Expectation Maximization and Neural Networks);
- *Classifiers:* three implementations of supervised machine learning algorithms (SVM, Naïve Bayes and Random Forest) are used in this procedure;
- *meta_learning.py:* a procedure that chooses the output that has the highest accuracy for traffic classification. It is configurable and ATLANTIC uses as default the output of SVM algorithm due to its high accuracy as reported by related work and our experiments. Note that any sort of meta-learning technique can be implemented;

Figure 4.4: Flow classifier internal procedures



Source: by author (2015).

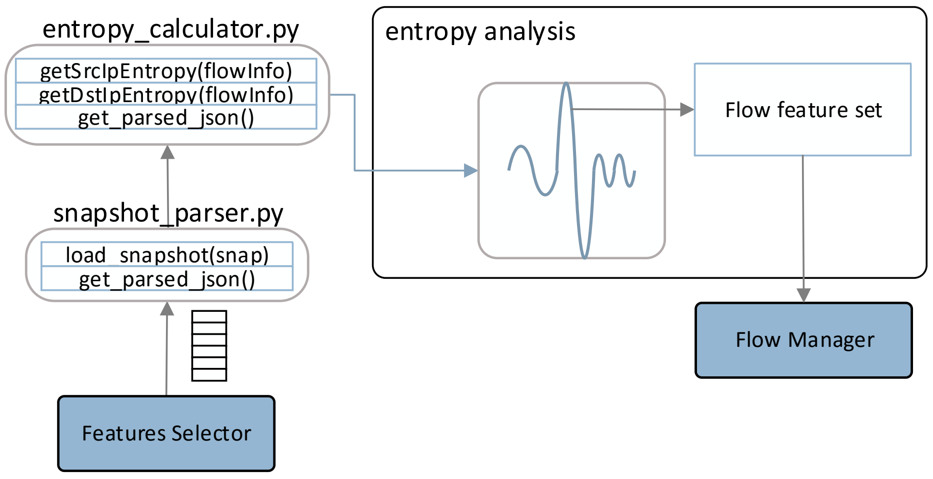
4.3.4 Traffic Monitor Interactions

The traffic monitor component uses three main procedures to implement a lightweigth traffic monitoring scheme:

- *snapshot_parser.py:* it is a procedure able to collect only the header fields present in a traffic snapshot;
- *entropy_calculator:* it is a procedure responsible for calculating the entropy associated with a traffic snapshot. The entropy calculation follows what is described in Chapter 3;
- *entropy analysis:* it is a procedure responsible for detecting deviations on the mean entropy value collected in the network;

It is an initial step to collect traffic information and determine what is the normal entropy associated with the network. This step was made using simulations that do not have an attack. In this way, the normal network entropy can be obtained. In real environments where the absense of malicious acitvites is not guaranteed in the time of entropy training, the historical mean entropy can be used. In this case, some network monitoring period should be used only to collect entropy information and eliminate the deviations on the mean value observed in the network.

Figure 4.5: Traffic monitor internal procedures



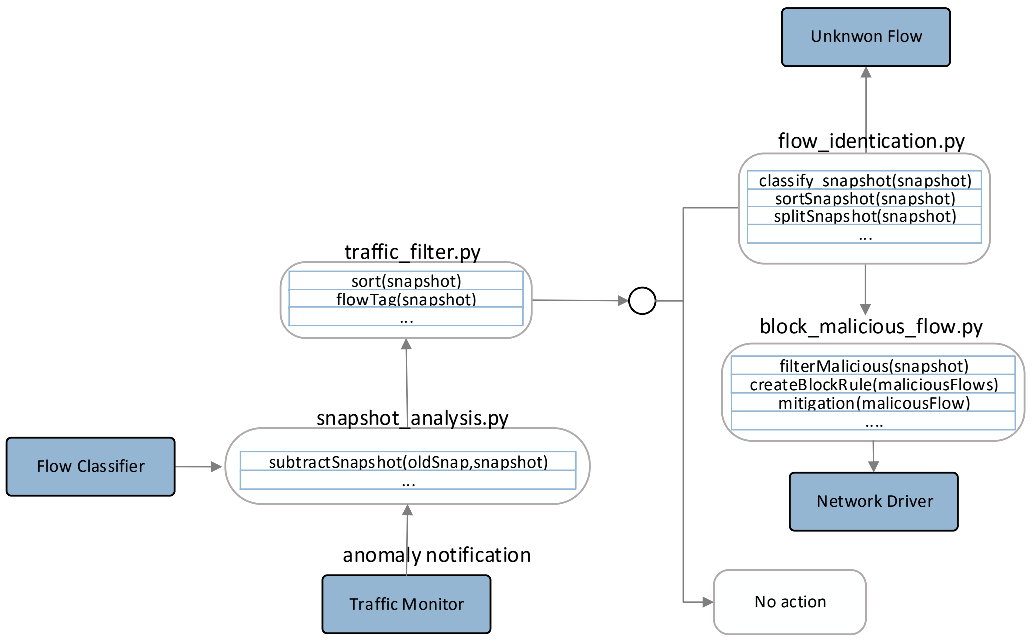
Source: by author (2015).

4.3.5 Flow Manager Interactions

The flow manager component handles network anomalies. It includes procedures to treat normal flows, procedures to treat anomalies and procedures to treat unkown flows. In the following, a brief description is provided:

- *snapshot_analysis.py:* it is the main procedure of the Flow Manager. Note that all traffic snapshots are delivery to the Flow Manager, however only when an anomaly notification arrives, this procedure analyzes the traffic snapshot. This analysis begins with the subtraction between the current traffic snapshot and the last one. The choice of using a hash-table indexed by the flow 5-tuple facilitates this subtraction because the hash keys in the intersection of two snapshots can be easily deleted. This subtraction is a necessary optimization step because the previous traffic snapshot is considered free from anomalies, so the anomaly should be a new flow in the current traffic snapshot. After this subtraction, the Flow Classifier component performs the traffic classification and delivers to this procedure a classified traffic snapshot.
- *traffic_filter.py:* this procedure receives a classified traffic snapshot and sorts its data structure in order to determine which flow needs mitigation actions. Flows that are considered benign do not require any action. However malicious or unknown flows need some treatment.
- *flow_identification.py:* this procedure is responsible for notifying the network administrator about unknown types of flows and for applying advanced techniques regarding flow mitigation. Currently, this procedure implements a firewall application able to identify a

Figure 4.6: Flow manager internal procedures



Source: by author (2015).

specific traffic flow and instruct the network to block malicious flows.

- *block_malicious_flows.py*: it is a procedure able to communicate with the Network Driver in order to configure block rules.

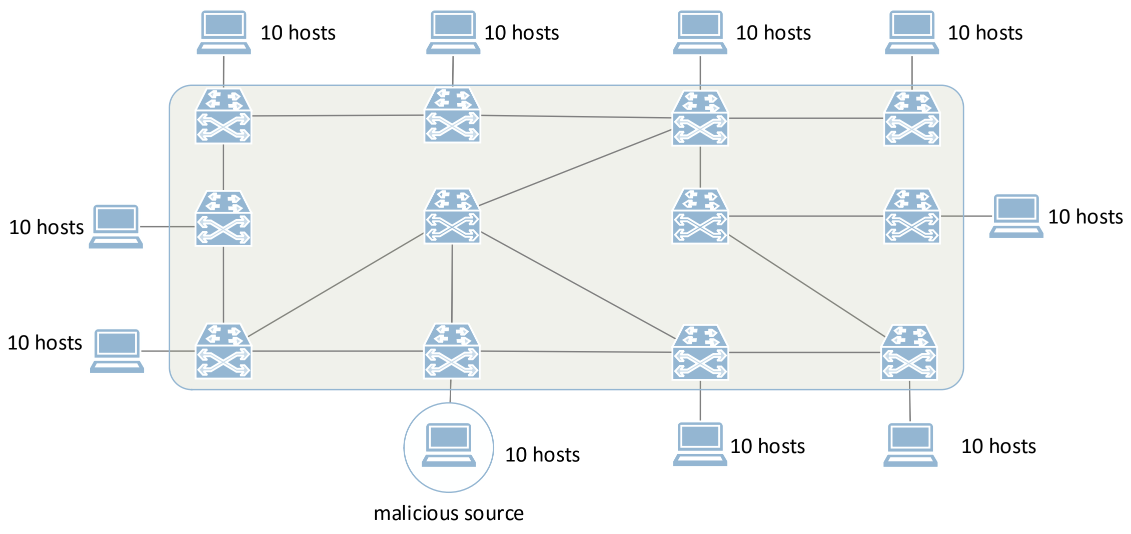
5 EXPERIMENTAL EVALUATION

In this chapter, we present an experimental evaluation of ATLANTIC. The evaluation process was divided into three steps. Firstly, we evaluate the performance of the lightweight phase (including the evaluation of aspects related to traffic snapshot request time/memory usage) and the performance of the heavyweight phase (including the evaluation of the classification accuracy of machine learning algorithms). Secondly, we show the experimental evaluation of the feature selection process, which is a significant step in the ATLANTIC framework. Finally, we present an overall framework evaluation, including a discussion on aspects related to mitigation strategies.

5.1 Simulation Profile

We ran the experiments using the Mininet emulator. We used a topology of a campus network. It consists of a partial mesh topology comprising 100 hosts and 11 switches, illustrated in Figure 5.1. We chose this scenario because of recent malicious attempts to attack similar environments (SANTANNA et al., 2015).

Figure 5.1: Network topology used in our experiments.



Source: by author (2015).

Table 5.1 describes the background traffic used in our experiments. When the simulation is set up, two services are configured: a video streaming server (whose service requests follow a lognormal distribution) and a Web server (whose service requests follow a exponential distribution) (ISOLANI et al., 2015). These services enable hosts to receive streaming over HTTP or send requests for Web pages, thus generating traffic related to file transfer. These traffic profiles have been validated in (ISOLANI et al., 2015). We generate these traffic profiles using VLC[1] for video streaming and SimpleHttpServer[2] to emulate an HTTP server.

Table 5.1: Background traffic profile used in the experiments

Parameter	Value
Number of hosts	100
Number of switches	11
Number of servers	2 (HTTP and Streaming)
Number of attack flows	3500
Traffic profile	Video: 75 %, Web: 25 %
Host behavior Web Server	Exponential Distribution ($\lambda = 0.033$,mean = 30 s)
Host behavior Video Streaming	Lognormal Distribution ($\mu = 11.75$,mean = 324 KBytes, std dev = 762 KBytes)

Source: by author (2015).

In order to simulate the user behavior, we set up a scenario where users watch a video for a certain amount of time, pause it, and then access a few Web pages. For each group of 6 hosts requesting HTTP traffic, there is 1 host requesting video streaming traffic. Network anomalies related to malicious activities are generated with the *scapy* tool[3], which enables the generation of realistic malicious attacks, such as port scanning and DDoS. The importance of these attacks has increased due to recent uses of DDoS to compromise campus communications (SANTANNA et al., 2015). Next, we explain the behavior of our simulated attacks.

- **Port scanning -** A malicious host can use port scanning to discover a set of open ports in a remote host. Open ports can be used to exploit vulnerabilities in a target system or be used in worm propagations (SCHAEFFER-FILHO et al., 2013). We simulate a malicious user that chooses a random host to start its attack to a server in the network. The attack consists of sending several TCP connection packets to ports ranging from 0 to 65536. When an open port is found, a notification is generated and this port can be used for worm propagations. Typically, *port scanning* generates packets in the network with a fixed IP address, but with varying transport protocol port.
- **DDoS attack -** We also defined a *DDoS attack* (BRAGA; MOTA; PASSITO, 2010) scenario. A DDoS attack in SDN can be used to overflow with a large amount of spurious flows a specific switch's flow table or to overload the network controller by producing several `packet_in` messages. Frequently, these attacks result in a range of source IP addresses accessing a single target IP. In this case-study, we create a SYN flood attack, *i.e.*, a malicious machine chooses a server (HTTP or streaming) to send multiple TCP SYN packets to service ports offered by this machine (8080 for the VLC streaming and 8000 for the HTTP server). Given that there are services running in those ports, the server

[1]http://www.videolan.org/vlc/
[2]https://docs.python.org/2/library/simplehttpserver.html
[3]http://www.secdev.org/projects/scapy/

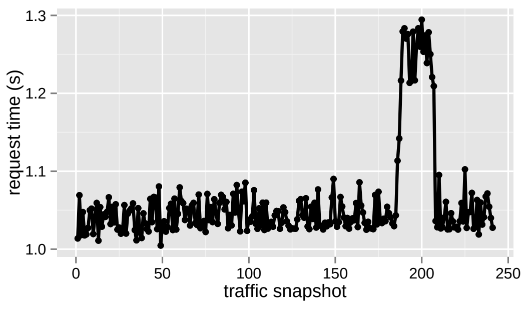
will process a request, allocate resources to handle it, and send an ACK to the requesting machine. Further, the attacker simulated in our experiments uses source IP spoofing, *i.e.*, he or she sends TCP SYN packets with the source IP address of another machine, thus causing the erroneous receipt of an unsolicited ACK and leaving the server with an open TCP connection.

5.2 Lightweight Anomaly Detection Evaluation

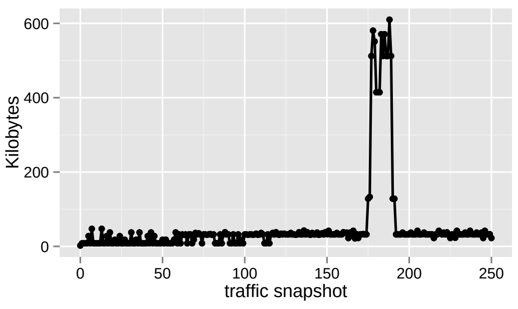
In this section we demonstrate the operation of the lightweight phase when it is instantiated with an entropy-based anomaly detection scheme to monitor the network in the presence of malicious anomalies. Additionally, we analyze the performance of this phase regarding memory usage and processing time.

Initially, the *Network Driver* communicates with the controller to request traffic information. There are two possible bottlenecks in this approach: the traffic snapshot transmission time and the amount of memory needed to store this information. To understand these issues, we monitored the number of flows generated in the network while users were accessing HTTP pages and video streaming during a few minutes. Next, we started a DDoS attack and monitored the amount of new traffic flows. The transmission time required to export this information to our framework and the amount of memory needed were observed. The polling interval was set to 5s in order to obtain a fine-grained view of the network traffic. According to Figure 5.3(a), the transmission time related to traffic snapshot when an attack is not happening remains under 1.07s. Around the 180^{th} snapshot, the DDoS attack starts and increases this transmission time to 1.28s in the worst case. Furthermore, according to Figure 5.3(b), the size of network snapshots increases from 32 to 600 kilobytes with approximately 4,400 flows, including malicious and benign.

Figure 5.2: Resources usage to request and store a traffic snapshot.



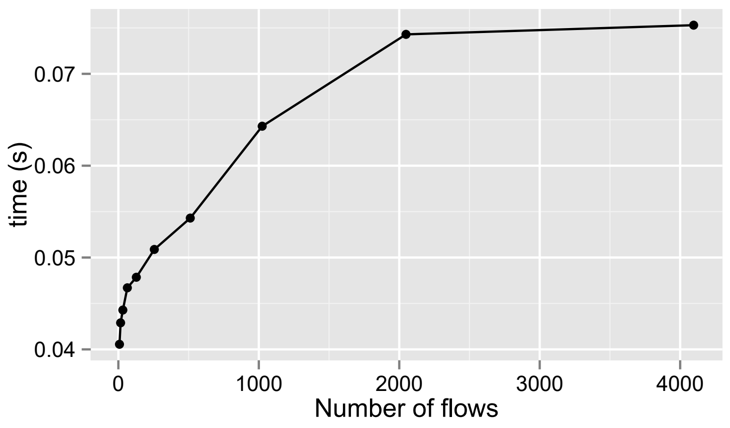
(a) Traffic snapshot requesting time.



(b) Traffic snapshot memory used.

Source: by author (2015).

Figure 5.3: Processing time of entropy calculation.



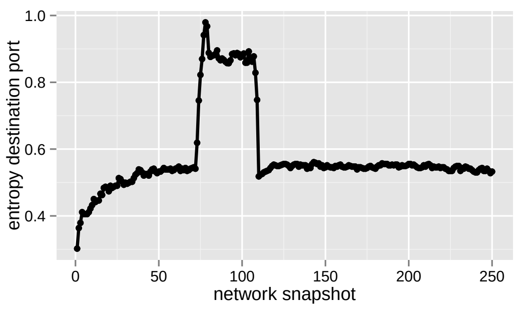
Source: by author (2015).

We argue that our simulations can realistically reflect the size of a large-scale campus scenario. In particular, it has been shown in (ISOLANI et al., 2015) a similar evaluation in campus scenarios comprising around 800 traffic flows on average. We also analyzed individual flow rules in ATLANTIC and verified that a single flow rule uses around 136 bytes and takes less than 0.8ms to be transmitted. Thus, in order to simulate the campus scenario presented in (ISOLANI et al., 2015), ATLANTIC would use 108,800 bytes (106.25 Kb) to store a traffic snapshot and 0.64s to transmit this information.

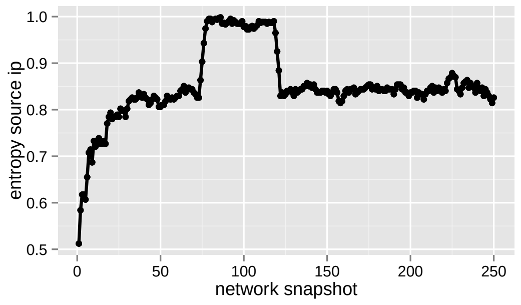
Entropy mean values associated with traffic features can be used to detect network anomalies. The performance of the *Flow Monitor* itself is related to entropy calculation time, which presents a logarithm fashion accordingly to the number of flows in a specific traffic snapshot. Figure 5.3 indicates that the entropy calculation time for 4,000 flows is 0.075s. We conclude that this type of traffic monitoring is suitable for ATLANTIC mainly because it is fast and accurate for detecting traffic deviations when we analyze, for example, the source IP entropy of a traffic snapshot. Figure 5.5(a) illustrates the variation in entropy of the destination port when a port scanning attack occurs. Around the 180^{th} snapshot, the average entropy that was around 0.55 rises to almost 1, indicating an anomaly. It is also possible to observe in Figure 5.5(b) the changes in the entropy of the destination IP address caused by the DDoS attack. Around the 280^{th} simulation snapshot, the DDoS attack stops, thus causing the entropy to revert back to normal (between 0.8 and 0.9).

When changes in the entropy of a specific feature are detected, the *Flow Classifier* component needs to determine the nature of existing flows in the network. The performance of this component is discussed in the next sub-section.

Figure 5.4: Entropy observed including benign and malicious flow.



(a) Destination port entropy in the port scanning scenario.



(b) Destination IP entropy in the DDoS attack scenario.

Source: by author (2015).

5.3 Heavyweight Anomaly Classification Evaluation

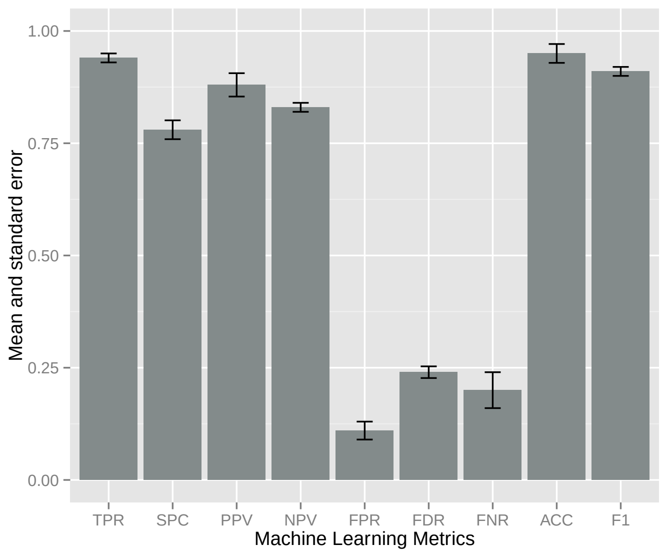
Each time a sudden change in the measured entropy of a specific feature is detected, a notification is generated and the ATLANTIC framework enters into its heavyweight phase. The first action taken by the *Flow Classifier* is the classification of all remaining flows after the subtraction of the current traffic snapshot from the last one.

To classify flows into separate classes representing DDoS and normal traffic, we instantiate the Flow Classifier with the following supervised machine learning algorithms: Support Vector Machines (SVM), Naïve Bayes and Neural Networks. When we analyze the classification accuracy as depicted in Figure 5.5, Figure 5.6, and Figure 5.7 we confirm that SVM is the best choice for traffic classification.

We highlight that the metrics used to evaluate the performance of the Flow Classifier component are the classical metrics used in machine learning (NGUYEN; ARMITAGE, 2008). We summarize them below:

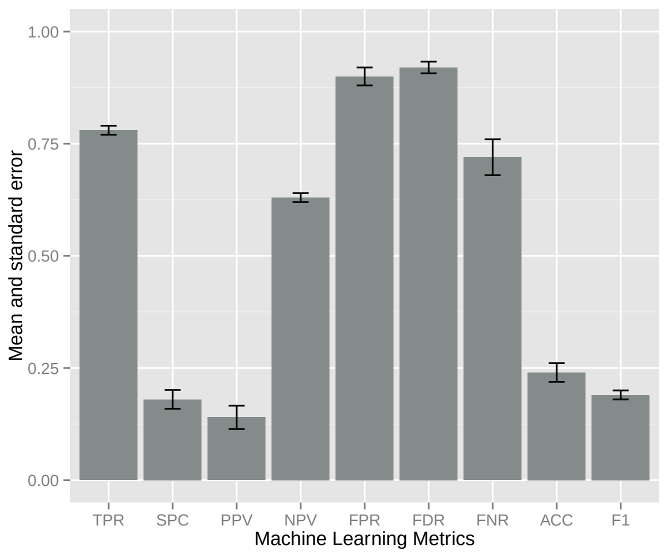
- TPR : sensitivity or true positive rate;
- SPC : specificity or true negative rate;
- PPV : precision;
- NPV: negative predictive value;
- FPR: false positive rate;
- FDR: false discovery rate;
- FNR: false negatives rate;
- ACC: accuracy;
- F1-score: harmonic mean of precision and specificity;

Figure 5.5: Machine Learning metrics for SVM.



Source: by author (2015).

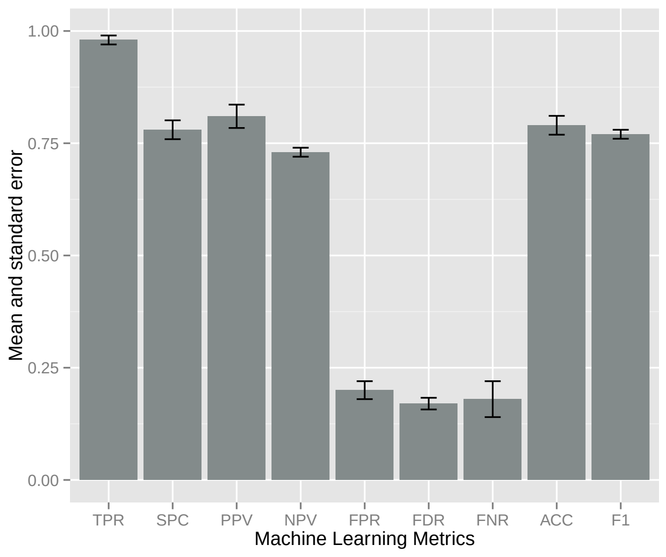
Figure 5.6: Machine Learning metrics for Naïve Bayes.



Source: by author (2015).

Additionally, ATLANTIC can use K-means, Random Forests and Expectation Maximization (JAIN; DUBES, 1988) as unsupervised machine learning algorithms. Our evaluation of unsupervised machine learning algorithms uses different metrics, in particular because these algorithms do not classify data. Instead, they are typically used to group data to optimize the use of supervised machine learning. For example, if 500 flow samples should be classified, then 500 executions of a supervised machine learning should be performed. However, initially using

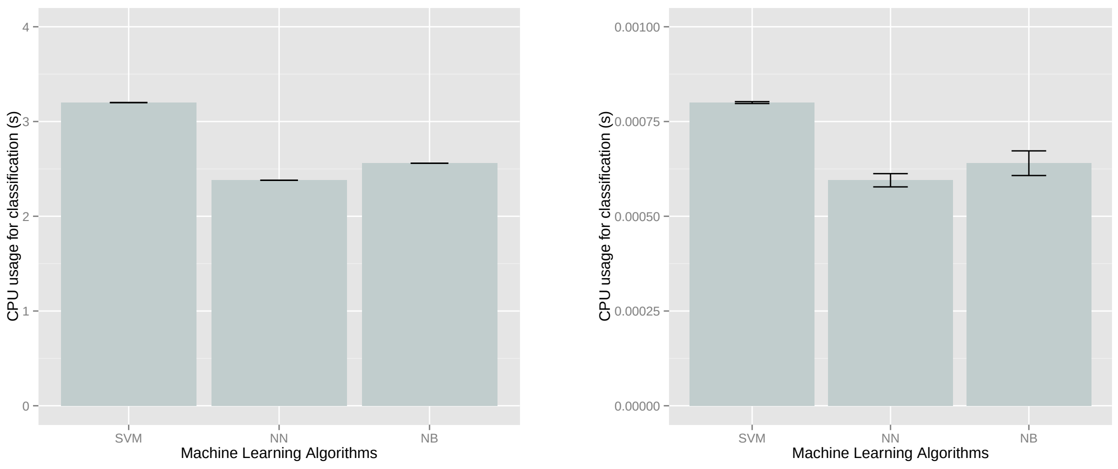
Figure 5.7: Machine Learning metrics for Neural Networks.



Source: by author (2015).

unsupervised machine learning to group these flows by similarity, resulting, for example, in 50 different groups can reduce the number of executions necessary for a supervised algorithm. The performance gain using the SVM algorithm in combination with a clusterization algorithm (K-means) can be viewed in Figure 5.9(a) and Figure 5.9(b).

Figure 5.8: Resources usage to request and store a traffic snapshot



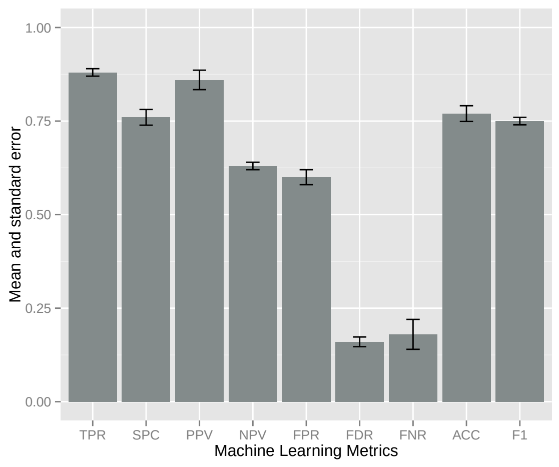
(a) Machine learning classification time without clusterization.

(b) Machine learning classification time with clusterization.

Source: by author (2015).

Based on the joint classification offered by K-means and SVM, we calculate the value of metrics such as *precision (PPV)* and *accuracy (ACC)*, which allow assessing the quality of the classification achieved by these algorithms. In particular, it is possible to apply K-means using different values of k in order to find the optimal configuration and, jointly with SVM, find which combination achieves better classification metrics. Figure 5.9 illustrates our results. It can be observed that SVM presents accuracy of 88.7% and precision of 82.3%. The simulations were executed 35 times until the error rate was less than 0.01. The results obtained are very similar to the values expected from traditional networks, as presented in (LI et al., 2011). Note that the classification accuracy of SVM algorithm is around 90.2% and SVM jointly with K-means is around 88.7%. This decrease of accuracy occurs because in some cases K-means groups outliers flows in a particular cluster, generating the misclassification. However, the overall values of machine learning metrics remain very similar with the SVM without K-means, thus encouraging the jointly use of these algorithms.

Figure 5.9: Machine Learning metrics for SVM and K-means.



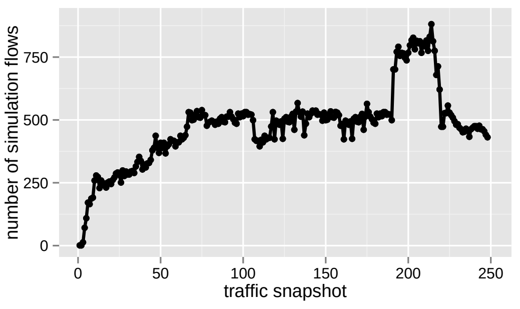
Source: by author (2015).

After obtaining the classification, the *Flow Manager* is notified and then it is able to block malicious flows and restore the entropy back to normal. To block these malicious occurrences in the network, the *Flow Manager* can instruct the *Network Driver* to use a firewall application or to use the OpenFlow drop action installed on the data plane.

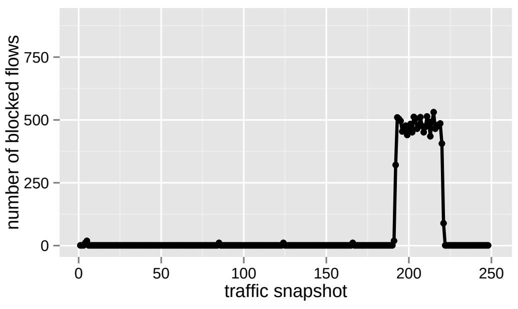
Note that each traffic snapshot may contain HTTP and streaming flows with different profiles, such as short-lived HTTP flows and long-term video streaming flows. Figure 5.11(a) and Figure 5.11(b) summarize the amount of active flows in the simulation, as well as the flows, signalized as malicious and subsequently blocked.

The heavyweight phase lasts around 3 seconds in our simulations (Figure 5.11). When we compare this with the processing time of the lightweight phase (which takes around 0.07 seconds), we can more clearly appreciate the benefits of using more frequently the lightweight phase instead of always using the heavyweight phase to classify every traffic snapshot.

Figure 5.10: Number of simulation flows



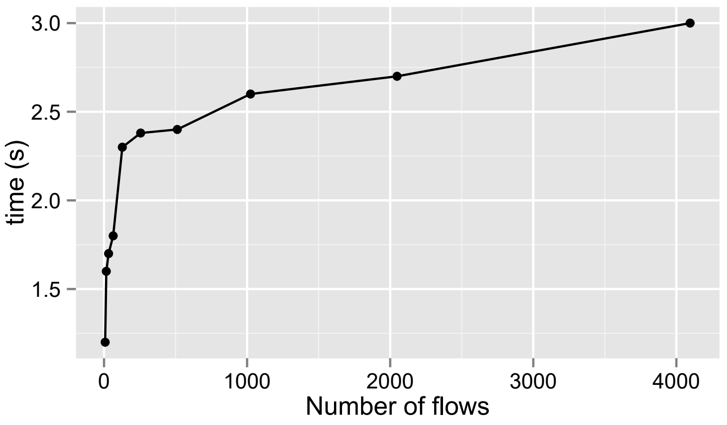
(a) Overall flows



(b) Blocked flows

Source: by author (2015).

Figure 5.11: Processing time of Heavyweight Phase.



Source: by author (2015).

5.4 Feature Selection Component Performance

In this section, we show the feature selection results in ATLANTIC. We evaluate this specific aspect because of its importance for accurate traffic classification and the lack of adequate research on this field in OpenFlow networks.

5.4.1 Experimental Results

Our main goal is to measure the accuracy of the resulting traffic classification using specific subsets of traffic features, discovered via the Genetic Algorithm (GA) and the Principal Component Analysis (PCA), as opposed to using the full-blown set of features. Note that accuracy is defined as the percentage of correctly classified instances among the total number of instances

(NGUYEN; MINH; YAMADA, 2013b). We defined four types of traffic profiles in our experiments: (*i*) DDoS attack, (*ii*) FTP traffic, (*iii*) video streaming using VLC Media Player, and (*iv*) background traffic generated using Scapy. We composed these types of traffic into three scenarios defined in Table 5.2. For each scenario we use the traffic classification accuracy as metric to define the optimal flow feature set. For comparison purposes, we use as benchmark the classification accuracy for each scenario obtained by using the complete set of flow features: 94.67% (scenario DDoS), 92% (scenario FTP), and 85.33% (scenario Video Streaming).

Table 5.2: Experimental scenarios.

Scenario	**Type of flow**
DDoS	DDoS attacks (60%) with Scapy flow (40%)
FTP	FTP traffic (35%) with Scapy flows (65%)
Video Streaming	Video streaming (50%) with Scapy flows (50%)

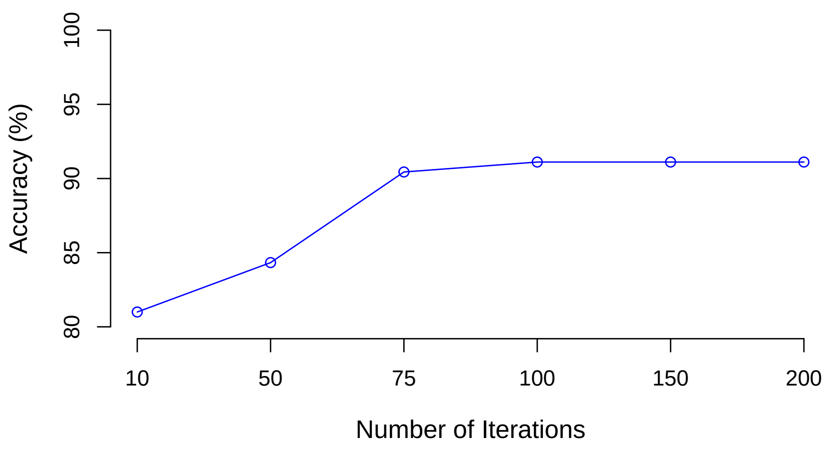
Source: by author (2015).

In order to produce more accurate results, the Genetic Algorithm (GA) must be initialized with a few parameters. We set the *population size* to 200 individuals randomly generated and the *crossover percentage* to 20%. We also must configure two other parameters: (*i*) *number of iterations* and (*ii*) *mutation probability*. In order to do so, we analyzed the classification accuracy for a number of iterations, namely 10, 50, 75, 100, 150, and 200, until the accuracy stabilized. Figure 5.12 presents the resulting accuracy obtained for a given number of iterations. As a result, we set the number of iterations to 100, since it is the smallest value with the highest accuracy.

As can be observed in Figure 5.13, the *mutation probability* did not have an impact in classification accuracy, and we use 0.01 as standard mutation probability. Because of the amount of time taken to execute this algorithm (nearly 88 minutes), we chose scenario Video Streaming to set these parameters, since it has the lowest classification accuracy when using all flow features. The classification accuracies obtained by using GA to find the optimal subset of flow features for each scenario were: 98% (scenario DDoS), 94.67% (scenario FTP), and 91.33% (scenario Video Streaming).

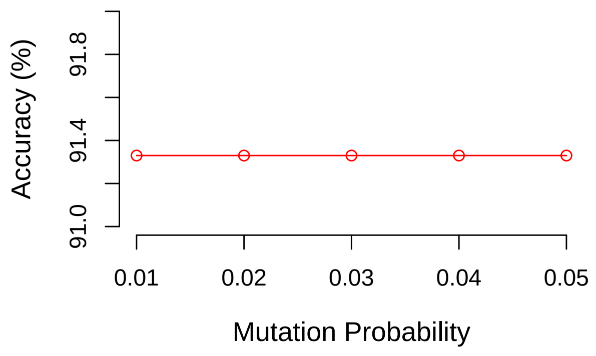
To find the optimal subset of flow features, we also applied PCA. PCA determines the most important features by creating one principal component to match each variable (feature), *i.e.*, in our experiments it creates 33 principal components. It works as follows: initially, only the components that jointly represent 90% of the features variability are chosen, resulting in 11 components; then, since a principal component is created as a weighed sum of the features, a subset is created for each component including the features with higher weight than a specific

Figure 5.12: Accuracy for each number of iterations.



Source: by author (2015).

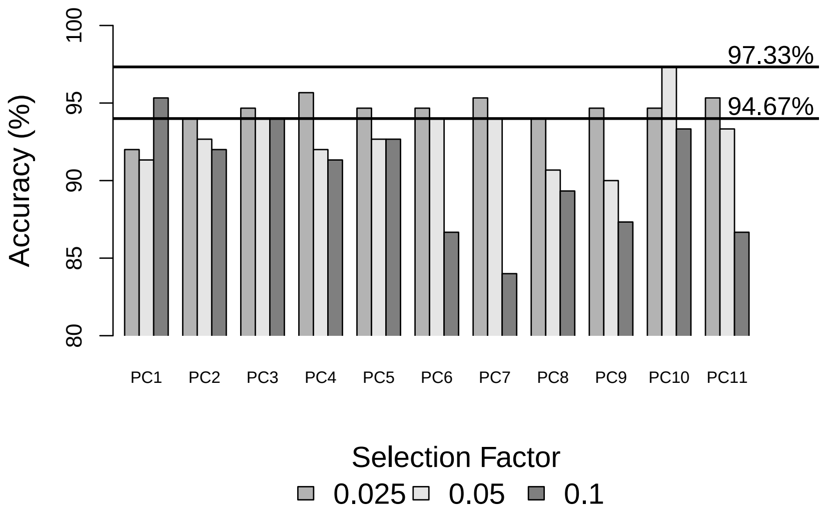
Figure 5.13: Accuracy for each mutation probability.



Source: by author (2015).

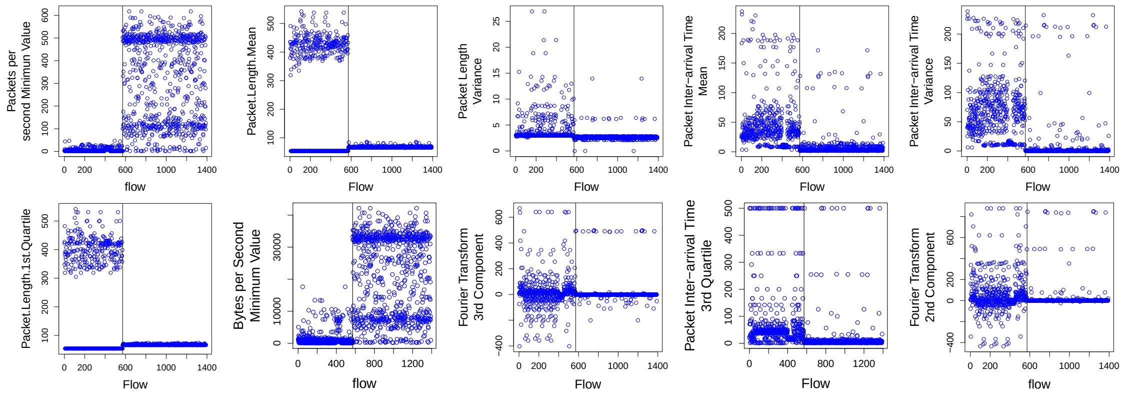
factor. In our experiments, we used 0.025, 0.05, and 0.1 as factors. As a result, we have three different subsets for each component. Figure 5.14 shows the classification accuracy for each subset in scenario DDoS. We can draw several conclusions from this: (*i*) not all subsets lead to a higher accuracy compared to the accuracy obtained by using the complete flow feature set (94.67%); (*ii*) factor 0.025 seems to lead to the best solutions in most cases, although the optimal solution is a subset of features selected using factor 0.05 in the 10^{th} principal component (97.33%); and (*iii*) factor 0.1 cannot be ruled out because it leads to the highest accuracy in the 1^{st} principal component, thus can also result in high accuracy in a different scenario. Based on these three observations, it is not possible to identify which factor gives the best subset for any

Figure 5.14: Accuracy of the 11 most meaningful principal components selected by PCA (for each factor) in scenario DDoS.



Source: by author (2015).

Figure 5.15: Ten most meaningful flow features according to PCA analysis in Scenario DDoS.



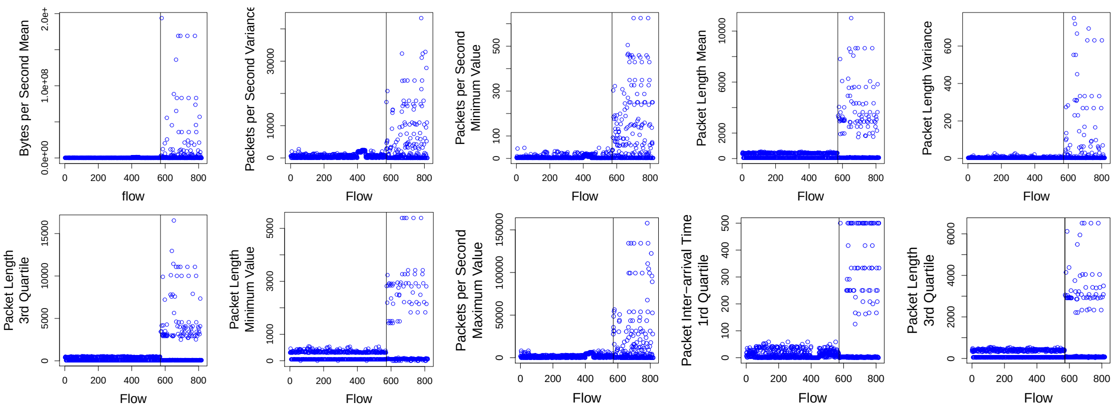
Source: by author (2015).

given scenario, consequently all three factor must be considered.

Because of space constraints, we illustrate only the 10 most meaningful features (in order of importance) selected through PCA in each scenario, depicted in Figure 5.15, Figure 5.16, and Figure 5.17. In each scatter plot, the vertical line separates the types of traffic being simulated: the left side represents the background traffic, and the right side represents DDoS traffic (in Figure 5.15), FTP flows (in Figure 5.16), and Video Streaming (in Figure 5.17). Classification accuracies for the optimal subset of features selected with PCA for each scenario were: 97.33% (scenario DDoS), 94% (scenario FTP), and 88.67% (scenario Video Streaming).

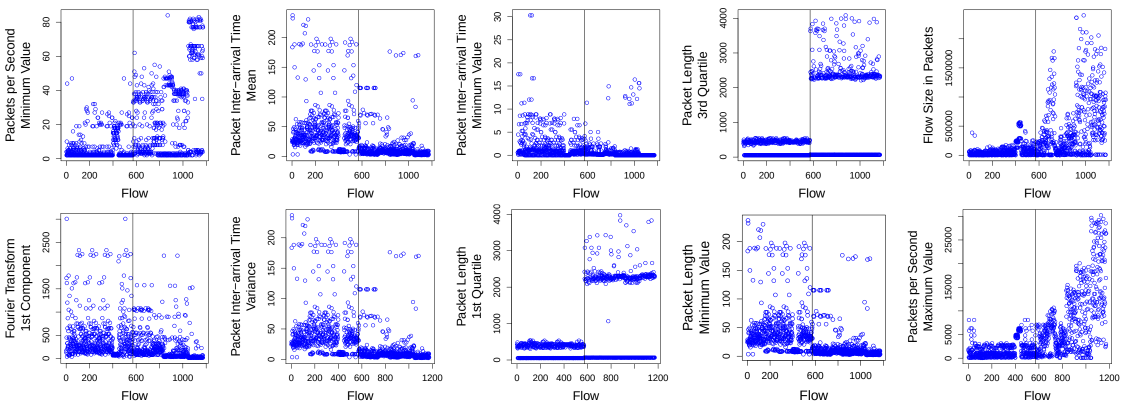
Figure 5.18 summarizes the classification accuracy in each scenario using all features, as

Figure 5.16: Ten most meaningful flow features according to PCA analysis in Scenario FTP.



Source: by author (2015).

Figure 5.17: Ten most meaningful flow features according to PCA analysis in Scenario Video Streaming.



Source: by author (2015).

well as using the optimal subsets selected with PCA and GA. In all scenarios, the accuracy was improved by using the optimal feature subsets generated by GA and PCA, when compared with the accuracy obtained by using the complete set of flow features. Table 5.3 details the feature subsets selected by PCA and GA in each scenario.

5.5 Discussion: Overall Framework Evaluation

We advocate that ATLANTIC meets the management requirements listed in Chapter 1. The lightweight step performs traffic monitoring using a global network view demonstrating that detailed traffic information can be easily achieved. This property contributes toward the comprehensive network view requirement needed for accurate anomaly detection.

Table 5.3: Optimal flow features subsets selected for each experiment.

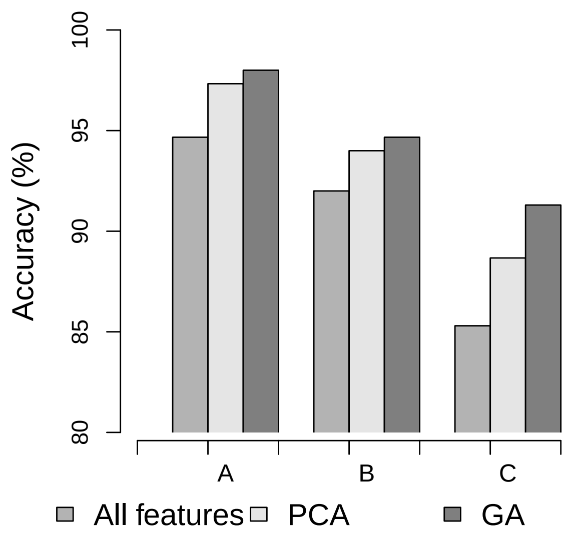
Scenario DDoS		Scenario FTP		Scenario Video Streaming	
PCA	**GA**	**PCA**	**GA**	**PCA**	**GA**
Bytes per second mean	Bytes per second mean	Bytes per second mean	Bytes per second variance	Bytes per second Minimum value	Bytes per second variance
Bytes per second Minimum value	Bytes per second variance	Bytes per second variance	Bytes per second Maximum value	Packets per second mean	Bytes per second Maximum value
Packets per second Minimum value	Bytes per second Maximum value	Bytes per second Maximum value	Minimum value of Bytes per second	Packets per second variance	Bytes per second Minimum value
Packets Length mean	Bytes per second Minimum value	Bytes per second Minimum value	Packets per second Mean	Packets per second Maximum value	Packets per second variance
Packets Length variance	Packets per second mean	Packets per second mean	Packets per second Variance	Packets per second Minimum value	Packets per second Minimum value
Packet Length Maximum value	Packets per second Maximum value	Packets per second variance	Packets per second Maximum value	Packet Length variance	Packet Length mean
Packet Length Minimum value	Packets per second Minimum value	Packet per second Maximum value	Packet per second Minimum value	Packet Length Minimum value	Packet Length variance
Packet Inter-arrival time mean	Packet Length mean	Packet per second Minimum value	Packet Length Mean	Packet Inter-arrival time mean	Packet Length Maximum value
Packet Inter-arrival time variance	Packet Length variance	Packet Length mean	Packet Length Maximum value	Packet Inter-arrival time variance	Packet Length Minimum value
Packet Inter-arrival time Maximum value	Packet Length Maximum value	Packet Length variance	Packet Length Minimum value	Packet Inter-arrival time Maximum value	Packet Inter-arrival time mean
Packet Inter-arrival time 1^{st} quartile	Packet Length Minimum value	Packet Length Maximum value	Packet inter-arrival time Mean	Packet Inter-arrival time Minimum value	Packet Inter-arrival time Maximum value
Packet Length 1^{st} quartile	Packet Inter-arrival time mean	Packet Length Minimum value	Packet inter-arrival time Variance	Flow Size in Packets	Flow Size in Bytes
Packet Length 3^{rd} quartile	Packet Inter-arrival time Minimum value	Packet Inter-arrival time variance	Flow Size in Bytes	Packet Inter-arrival time 1^{st} quartile	Packet Inter-arrival time 1^{st} quartile
Fourier Transform 2^{nd} Component	Flow Duration	Packet Inter-arrival time Maximum value	Flow Size in packets	Packet Inter-arrival time 3^{rd} quartile	Packet Inter-arrival time 3^{rd} quartile
Fourier Transform 3^{rd} Component	Flow Size in Bytes	Packet Inter-arrival time Minimum value	Packet inter-arrival time 1^{st} quartile	Packet Length 1^{st} quartile	Packet Length 3^{rd} quartile
Fourier Transform 6^{th} Component	Flow Size in Packets	Flow Size in Bytes	Packet Length 1^{st} quartile	Fourier Transform 1^{st} Component	Fourier Transform 1^{st} Component
	Packet Inter-arrival time 3^{rd} quartile	Packet Inter-arrival time 1^{st} quartile	Packet Length 3^{rd} quartile	Fourier Transform 2^{nd} Component	Fourier Transform 2^{nd} Component
	Packet Length 1^{st} quartile	Packet Inter-arrival time 3^{rd} quartile	Fourier Transform 2^{nd} component	Fourier Transform 4^{th} Component	Fourier Transform 3^{rd} Component
	Fourier Transform 1^{st} Component	Packet Length 1^{st} quartile	Fourier Transform 4^{th} component	Fourier Transform 5^{th} Component	Fourier Transform 4^{th} Component
	Fourier Transform 2^{nd} Component	Packet Length 3^{rd} quartile	Fourier Transform 5^{th} component	Fourier Transform 6^{th} Component	Fourier Transform 5^{th} Component
	Fourier Transform 3^{rd} Component	Fourier Transform 2^{nd} Component	Fourier Transform 7^{th} component	Fourier Transform 7^{th} Component	Fourier Transform 8^{th} Component
	Fourier Transform 4^{th} Component	Fourier Transform 3^{rd} Component	Fourier Transform 10^{th} Component	Fourier Transform 8^{th} Component	Fourier Transform 9^{th} Component
	Fourier Transform 5^{th} Component	Fourier Transform 4^{th} Component		Fourier Transform 9^{th} Component	Fourier Transform 10^{th} Component
	Fourier Transform 6^{th} Component	Fourier Transform 5^{th} Component		Fourier Transform 10^{th} Component	
	Fourier Transform 9^{th} Component	Fourier Transform 7^{th} Component			
		Fourier Transform 8^{th} Component			
		Fourier Transform 9^{th} Component			
		Fourier Transform 10^{th} Component			

Source: by author (2015).

The design of ATLANTIC allows the network administrator to easily monitor and modify the operation of the components involved in the framework heavyweight phase using its configuration files. This characteristic contributes to a more tailored anomaly classification, which can rely on human intervention when the automated components are not able to protect the network. Additionally, ATLANTIC can use the network controller to orchestrate all flows in the network, sampling or eventually blocking a particular flow whenever needed.

The ATLANTIC framework can be extended with more sophisticated strategies for anomaly mitigation. For example, we list some alternatives to the use of blocking strategies:

Figure 5.18: Traffic classification accuracy for each flow feature set in all three scenarios.



Source: by author (2015).

- **Rate Limit:** malicious flows related to DDoS attacks can be rate limited by the Flow Manager configuration rules using queue strategies to handle new *packet_in* in the network;
- **Generic Rules Filter:** When flow tables are populated with generic flow rules in such a way that several different types of flows can match the same forwarding behavior, a filtering strategy is necessary to isolate only the malicious flows. In this case, a list of *dst_ip* addresses can be used to insert specific flow rules to get flows directed to some critical server and, in this case, the malicious entry can be blocked;
- **Distributed ATLANTIC:** multiple instances of the Flow Manager can be responsible for handling network slices with different types of networks and controllers;

6 CONCLUDING REMARKS

Mechanisms related to anomaly detection are an important component in strategies used to ensure network resiliency against, for example, malicious attacks. Classification allows categorizing the network traffic in a number of *classes* enabling the detection of these activities. Further, as result of the classification, traffic belonging to a particular class can be treated differently (e.g., dropped or rate limited).

In this dissertation we presented ATLANTIC, a framework for anomaly traffic detection, classification and mitigation. Our framework comprises a lightweight phase responsible for monitoring traffic flows and a heavyweight phase responsible for anomaly classification and mitigation. As a result, traffic anomalies can be categorized and the information collected can be used to handle each traffic profile in a specific manner, such as blocking malicious flows. Next, we highlight some conclusions and key contributions about our research.

6.1 Summary of Contributions

The main contributions of our research are the following:

- A lightweight monitoring scheme that enables ATLANTIC to detect malicious activities without overloading the network, taking about 0.075s to collect and analyze traffic information consisting of 4400 flows in a topology with 100 switches;
- A heavyweight phase using machine learning algorithms, such as SVM, which took less than 3s to classify traffic flows, demonstrating that ATLANTIC performs well even in the presence of DDoS attacks;
- An architecture that enables ATLANTIC to execute the lightweight phase more frequently than the heavyweight phase, thus minimizing the overhead of the overall anomaly detection scheme;
- An illustration of how a sophisticated anomaly detection framework can be built over SDN;
- An architecture to collect, extend, and select a set of flow features for traffic classification in Software-Defined Networking (SDN) using a large set of traffic features indeed more meaningful than the native flow counters provided by OpenFlow;
- An evaluation demonstrating that for particular types of traffic, an optimal subset of flow features can be selected using both principal component analysis and genetic algorithm;

To the best of our knowledge, ATLANTIC is the only framework that does anomaly detection, classification and mitigation tasks jointly, and can assist the understanding of how a broad range of research can be used to classify network traffic in SDN. We use the best techniques to instantiate ATLANTIC and propose a combination not seen before: it is the first time that entropy analysis is used jointly with machine learning to detect malicious flows in SDN.

6.2 Final Remarks and Future Work

As part of our future work, we aim to evaluate the use of different algorithms for traffic classification and entropy analysis to enforce network protection. Also, we intend to investigate new mitigation strategies, such as the use of rate limiters, and new classification schemes, such as the combination of several network classifiers using meta-learning techniques (*e.g.,* stacking, and bayesian networks). Finally, we will investigate additional flow features, and insert routines to enforce QoS policies over network anomalies. The insertion of QoS policies will treat flows that are not malicious, but degrade the network normal operation.

REFERENCES

A. LANDA R., C. R. G. T. J.; GEORGE, P. Software-defined network support for transport resilience. In: IEEE NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS). **Proceedings...,** Krakow, 2014. p. 1 – 8.

AGARWAL, B.; MITTAL, N. Hybrid Approach for Detection of Anomaly Network Traffic using Data Mining Techniques. **Procedia Technology**, vol. 6, p. 996 – 1003, 2012. ISSN 2212-0173. 2nd International Conference on Communication, Computing and Security (ICCCS).

AMAZONAS, J. de A.; SANTOS-BOADA, G.; SOLÉ-PARETA, J. A critical review of OpenFlow/SDN-based networks. In: 16th INTERNATIONAL CONFERENCE ON TRANSPARENT OPTICAL NETWORKS (ICTON). **Proceedings...,**, Austria, 2014. p. 1–5.

AULD, T.; MOORE, A.; GULL, S. Bayesian Neural Networks for Internet Traffic Classification. **IEEE Transactions on Neural Networks**, vol. 18, no. 1, p. 223–239, Jan 2007. ISSN 1045-9227.

BERDE, P. et al. ONOS: Towards an Open, Distributed SDN OS. In: THIRD WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKING (HOTSDN '14), **Proceedings...,** New York, NY, USA: ACM, 2014, p. 1–6. ISBN 978-1-4503-2989-7.

BLUM, A. L.; LANGLEY, P. Selection of Relevant Features and Examples in Machine Learning. **Journal Artificial Intelligence - Special issue on relevance**, Elsevier Science Publishers Ltd., Essex, UK, vol. 97, no. 1-2, p. 245–271, dez. 1997. ISSN 0004-3702. Available from Internet: <http://dx.doi.org/10.1016/S0004-3702(97)00063-5>.

BRAGA, R.; MOTA, E.; PASSITO, A. lightweight DDoS flooding attack detection using NOX/OPENFLOW. In: IEEE 35TH CONFERENCE ON LOCAL COMPUTER NETWORKS (LCN), **Proceedings...,**. Washington, DC, USA, 2010. p. 408–415. ISBN 978-1-4244-8387-7.

CARAGUAY, V. et al. Evolution and Challenges of Software Defined Networking. In: IEEE SDN FOR FUTURE NETWORKS AND SERVICES (SDN4FNS), **Proceedings...,** 2013. p. 1–7.

CASADO, M. et al. Fabric: A Retrospective on Evolving SDN. In: FIRST WORKSHOP ON HOT TOPICS IN SOFTWARE DEFINED NETWORKS (HotSDN '12),**Proceedings...,**. New York, NY, USA: ACM, 2012, p. 85–90. ISBN 978-1-4503-1477-0.

CETINKAYA, E.; STERBENZ, J. A taxonomy of network challenges. In: 9TH INTERNATIONAL CONFERENCE ON THE DESIGN OF RELIABLE COMMUNICATION NETWORKS (DRCN), **Proceedings...,**, 2013. p. 322–330.

CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly Detection: A Survey. **ACM Comput. Surv.**, ACM, New York, NY, USA, vol. 41, no. 3, p. 15:1–15:58, jul 2009. ISSN 0360-0300.

CUI, H. et al. Design of intelligent capabilities in SDN. In: 4TH INTERNATIONAL CONFERENCE ON WIRELESS COMMUNICATIONS, VEHICULAR TECHNOLOGY, INFORMATION THEORY AND AEROSPACE ELECTRONIC SYSTEMS (VITAE), **Proceedings...,** 2014. p. 1–5.

CURTIS, A. R. et al. DevoFlow: Scaling Flow Management for High-performance Networks. In: ACM SIGCOMM 2011 CONFERENCE (SIGCOMM '11), **Proceedings...,** New York, NY, USA: ACM, 2011, p. 254–265. ISBN 978-1-4503-0797-0.

ERMAN, J.; ARLITT, M.; MAHANTI, A. Traffic Classification Using Clustering Algorithms. In: SIGCOMM WORKSHOP ON MINING NETWORK DATA (MINENET), **Proceedings...,** New York, NY, USA, 2006. p. 281–286. ISBN 1-59593-569-X.

FAHAD, A. et al. Toward an efficient and scalable feature selection approach for internet traffic classification. **Computer Networks**, vol. 57, no. 9, p. 2040 – 2057, 2013. ISSN 1389-1286.

FARHADI, H.; DU, P.; NAKAO, A. Enhancing OpenFlow actions to offload packet-in processing. In: ASIA-PACIFIC NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (APNOMS), **Proceedings...,** p. 1–6, Sept 2014.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN. **Queue - Large-Scale Implementations, Volume 11 Issue 12**, ACM, New York, NY, USA, vol. 11, no. 12, p. 20:20–20:40, December 2013. ISSN 1542-7730.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN. **Queue - Large-Scale Implementations**, ACM, New York, NY, USA, vol. 11, no. 12, p. 20:20–20:40, dez. 2013. ISSN 1542-7730. Available from Internet: <http://doi.acm.org/10.1145/2559899.2560327>.

GIOTIS, K. et al. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. **Computer Networks**, vol. 62, no. 0, p. 122 – 136, 2014. ISSN 1389-1286.

GIOTIS, K. et al. Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. **Computer Networks**, vol. 62, p. 122 – 136, 2014. ISSN 1389-1286.

GREENBERG, A. et al. The Cost of a Cloud: Research Problems in Data Center Networks. **SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, vol. 39, no. 1, p. 68–73, dez. 2008. ISSN 0146-4833.

HU, F.; HAO, Q.; BAO, K. A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation. In **IEEE Communications Surveys & Tutorials**. [S.l.: s.n.], 2014. vol. 16, p. 2181–2206. ISSN 1553-877X.

ISOLANI, P. H. et al. Interactive Monitoring, Visualization, and Configuration of OpenFlow-based SDN. In: 14TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM), **Proceedings...,** Ottawa, Canada, 2015. p. 207–215.

JAIN, A. K.; DUBES, R. C. **Algorithms for Clustering Data**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988. ISBN 0-13-022278-X.

JARRAYA, Y.; MADI, T.; DEBBABI, M. A Survey and a Layered Taxonomy of Software-Defined Networking. **IEEE Communications Surveys Tutorials**, vol. 16, no. 4, p. 1955–1980, Fourthquarter 2014. ISSN 1553-877X.

KIM, H. et al. Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices. In: ACM CONEXT CONFERENCE (CONEXT '08), **Proceedings...,**. New York, NY, USA: ACM, 2008, p. 11:1–11:12. ISBN 978-1-60558-210-8.

KITTLER, J. et al. On combining classifiers. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, vol. 20, no. 3, p. 226–239, Mar 1998. ISSN 0162-8828.

KITTLER, J. et al. On combining classifiers. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, vol. 20, no. 3, p. 226–239, mar. 1998. ISSN 0162-8828. Available from Internet: <http://dx.doi.org/10.1109/34.667881>.

KLOFT, M. et al. Automatic Feature Selection for Anomaly Detection. In: 1st ACM Workshop on Workshop on AISec,**Proceedings...,** New York, NY, USA: ACM, 2008. (AISec '08), p. 71–76. ISBN 978-1-60558-291-7.

LAKHINA, A.; CROVELLA, M.; DIOT, C. Mining Anomalies Using Traffic Feature Distributions. In **ACM SIGCOMM**. New York, NY, USA: [s.n.], 2005. p. 217–228. ISBN 1-59593-009-4.

LANZI, P. Fast feature selection with genetic algorithms: a filter approach. In: IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION, **Proceedings...,** 1997. p. 537–540.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network Innovation using OpenFlow: A Survey. **IEEE Communications Surveys Tutorials**, vol. 16, no. 1, p. 493–512, First 2014. ISSN 1553-877X.

LI, X. et al. An Internet Traffic Classification Method Based on Semi-Supervised Support Vector Machine. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC), **Proceedings...,** Kyoto, Japan, 2011. p. 1–5. ISSN 1550-3607.

MANTERE, M.; SAILIO, M.; NOPONEN, S. Feature Selection for Machine Learning Based Anomaly Detection in Industrial Control System Networks. In: IEEE INTERNATIONAL CONFERENCE ON GREEN COMPUTING AND COMMUNICATIONS (GREENCOM), **Proceedings...,** 2012. p. 771–774.

MCHALE, L. et al. Stochastic Pre-classification for SDN Data Plane Matching. In: IEEE 22ND INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS (ICNP), **Proceedings...,** 2014. p. 596–602.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Computer Communication Review**, ACM, New York, NY, USA, vol. 38, no. 2, p. 69–74, March 2008. ISSN 0146-4833.

MEHDI, S. A.; KHALID, J.; KHAYAM, S. A. Revisiting Traffic Anomaly Detection Using Software Defined Networking. In: 14TH INTERNATIONAL CONFERENCE ON RECENT ADVANCES IN INTRUSION DETECTION (RAID'11), **Proceedings...,** Berlin, Heidelberg: Springer-Verlag, 2011, p. 161–180. ISBN 978-3-642-23643-3.

MOORE, A. W.; ZUEV, D. Internet Traffic Classification Using Bayesian Analysis Techniques. In **ACM SIGMETRICS**. New York, NY, USA: [s.n.], 2005. p. 50–60. ISBN 1-59593-022-1.

MOORE, A. W.; ZUEV, D. Internet Traffic Classification Using Bayesian Analysis Techniques. **SIGMETRICS Perform. Eval. Rev.**, ACM, New York, NY, USA, vol. 33, no. 1, p. 50–60, jun. 2005. ISSN 0163-5999. Available from Internet: <http://doi.acm.org/10.1145/1071690.1064220>.

NAKAYAMA, H. et al. An implementation model and solutions for stepwise introduction of SDN. In: 16TH ASIA-PACIFIC NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (APNOMS), **Proceedings...,** p. 1–4, Sept 2014.

NGUYEN, K.; MINH, Q. T.; YAMADA, S. A Software-Defined Networking Approach for Disaster-Resilient WANs. In: 22ND INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS (ICCCN), **Proceedings...,**, 2013. p. 1–5.

NGUYEN, K.; MINH, Q. T.; YAMADA, S. A Software-Defined Networking Approach for Disaster-Resilient WANs. In: 22ND INTERNATIONAL CONFERENCE ON COMPUTER COMMUNICATIONS AND NETWORKS (ICCCN), **Proceedings...**, 2013. p. 1–5.

NGUYEN, T.; ARMITAGE, G. A Survey of Techniques for Internet Traffic Classification using Machine Learning. **IEEE Communications Surveys & Tutorials**, vol. 10, no. 4, p. 56–76, 2008.

NYCHIS, G. et al. An Empirical Evaluation of Entropy-based Traffic Anomaly Detection. In: 8TH ACM SIGCOMM CONFERENCE ON INTERNET MEASUREMENT (IMC), **Proceedings...,** New York, USA, 2008, p. 151–156. ISBN 978-1-60558-334-1.

OH, I.-S.; LEE, J.-S.; MOON, B.-R. Hybrid genetic algorithms for feature selection. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, vol. 26, no. 11, p. 1424–1437, Nov 2004. ISSN 0162-8828.

Open Networking Foundation. **OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01)**. 2009. Available at: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Accessed: August 2014.

PANTUZA, G. et al. Network management through graphs in Software Defined Networks. In: 10TH INTERNATIONAL CONFERENCE ON NETWORK AND SERVICE MANAGEMENT (CNSM),**Proceedings...,** 2014. p. 400–405.

PASCOAL, C. et al. Robust feature selection and robust pca for internet traffic anomaly detection. In **Proceedings of IEEE INFOCOM**. [S.l.: s.n.], 2012. p. 1755–1763.

PEARSON., K. LIII. On lines and planes of closest fit to systems of points in space. **Philosophical Magazine Series 6**, vol. 2, no. 11, p. 559–572, 1901. Available from Internet: <http://dx.doi.org/10.1080/14786440109462720>.

SANTANNA, J. J. et al. Booters - An Analysis of DDoS-as-a-Service Attacks. In: 14TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM), **Proceedings...,** Ottawa, Canada, 2015. p. 243 – 251.

SCHAEFFER-FILHO, A. et al. PReSET: A Toolset for the Evaluation of Network Resilience Strategies. In: 13TH IFIP/IEEE INTERNATIONAL SYMPOSIUM ON INTEGRATED NETWORK MANAGEMENT (IM), **Proceedings...,** Ghent, Belgium, 2013. p. 202–209.

SCHAEFFER-FILHO, A. et al. Network resilience with reusable management patterns. **IEEE Communications Magazine**, vol. 52, no. 7, p. 105–115, July 2014. ISSN 0163-6804.

SCHIFF, L.; BOROKHOVICH, M.; SCHMID, S. Reclaiming the Brain: Useful OpenFlow Functions in the Data Plane. In: 13TH ACM WORKSHOP ON HOT TOPICS IN NETWORKS (HotNets-XIII), **Proceedings...,** New York, NY, USA: ACM, 2014, p. 7:1–7:7. ISBN 978-1-4503-3256-9.

SEZER, S. et al. Are we ready for sdn? implementation challenges for software-defined networks. **Communications Magazine, IEEE**, vol. 51, no. 7, p. 36–43, July 2013. ISSN 0163-6804.

SHIN, S. et al. FRESCO: Modular Composable Security Services for Software-Defined Networks. In **Network and Distributed System Security Symposium (NDSS)**. [S.l.: s.n.], 2013. p. 1–16.

STERBENZ, J. P. G. et al. Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines. **Computer Networks.**, Elsevier North-Holland, Inc., New York, NY, USA, vol. 54, no. 8, p. 1245–1265, jun. 2010. ISSN 1389-1286.

WANG, W.; GOMBAULT, S. Efficient Detection of DDoS attacks with Important Attributes. In: 3RD INTERNATIONAL CONFERENCE ON RISKS AND SECURITY OF INTERNET AND SYSTEMS (CRISIS), **Proceedings...,** 2008. p. 61–67.

WICKBOLDT, J. et al. Software-defined networking: management requirements and challenges. **IEEE Communications Magazine**, vol. 53, no. 1, p. 278–285, January 2015. ISSN 0163-6804.

WICKBOLDT, J. A. et al. Software-Defined Networking: Management Requirements and Challenges. **IEEE Communications Magazine**, vol. 53, no. 1, p. 278–285, Jan 2015.

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.**, New York, NY, USA, vol. 36, no. 5, p. 5–16, out. 2006. ISSN 0146-4833.

XU, L.; KRZYZAK, A.; SUEN, C. Methods of combining multiple classifiers and their applications to handwriting recognition. **Systems, Man and Cybernetics, IEEE Transactions on**, vol. 22, no. 3, p. 418–435, May 1992. ISSN 0018-9472.

YU, Y. et al. An Adaptive Approach to Network Resilience: Evolving Challenge Detection and Mitigation. In: 8TH INTERNATIONAL WORKSHOP ON THE DESIGN OF RELIABLE COMMUNICATION NETWORKS (DRCN), **Proceedings...,** 2011. p. 172–179.

YUAN, R. et al. An SVM-based Machine Learning Method for Accurate Internet Traffic Classification. **Information Systems Frontiers**, Kluwer Academic Publishers, Hingham, MA, USA, vol. 12, no. 2, p. 149–156, apr 2010. ISSN 1387-3326.

YUAN, R. et al. An SVM-based Machine Learning Method for Accurate Internet Traffic Classification. **Information Systems Frontiers**, Kluwer Academic Publishers, Hingham, MA, USA, vol. 12, no. 2, p. 149–156, abr. 2010. ISSN 1387-3326.

ZHANG, Y. An Adaptive Flow Counting Method for Anomaly Detection in SDN. In: 9TH ACM CONFERENCE ON EMERGING NETWORKING EXPERIMENTS AND

TECHNOLOGIES (CONEXT ’13), **Proceedings...,** New York, USA: ACM, 2013, p. 25–30. ISBN 978-1-4503-2101-3.

AppendixA PUBLISHED PAPER – ISCC 2015

Resilience is the ability of the network to maintain acceptable levels of operation in face of anomalies, such as malicious attacks, operational overload or misconfigurations. Techniques for anomaly traffic classification are often used for characterising suspicious network traffic, thus supporting anomaly detection schemes in network resilience strategies. In this paper, we extend the PReSET toolset to allow the investigation, comparison and analysis of algorithms for anomaly traffic classification based on machine learning. PReSET was designed to allow the simulation-based evaluation of resilience strategies, thus enabling the comparison of optimal configurations and policies for combating different types of attacks (e.g., DDoS attacks, worm propagations) and other anomalies. In such resilience strategies, policies written in the Ponder2 language can be used to activate/reconfigure traffic classification modules and other mechanisms (e.g., traffic shaping), depending on monitored results in the simulation environment. Our results show that PReSET can be a valuable tool for network operators to evaluate anomaly traffic classification techniques in terms of standard performance metrics.

- **Title –**
  Tool Support for the Evaluation of Anomaly Traffic Classification for Network Resilience
- **Conference –**
  The Twentieth IEEE Symposium on Computers and Communications (ISCC 2015)
- **Type –**
  Main track (full-paper)
- **Qualis –**
  A2
- **Date –**
  Jul 06-09, 2015
- **Held at –**
  Larnaca, Cyprus

Tool Support for the Evaluation of Anomaly Traffic Classification for Network Resilience

Anderson Santos da Silva*, Juliano Araujo Wickboldt*, Alberto Schaeffer-Filho*,
Angelos K. Marnerides†, Andreas Mauthe†
*Institute of Informatics, Federal University of Rio Grande do Sul, Brazil
Email: {assilva, jwickboldt, alberto}@inf.ufrgs.br
†School of Computing and Communications, Lancaster University, United Kingdom
Email: {a.marnerides2, a.mauthe}@lancaster.ac.uk

Abstract—**Resilience is the ability of the network to maintain acceptable levels of operation in face of anomalies, such as malicious attacks, operational overload or misconfigurations. Techniques for anomaly traffic classification are often used for characterising suspicious network traffic, thus supporting anomaly detection schemes in network resilience strategies. In this paper, we extend the PReSET toolset to allow the investigation, comparison and analysis of algorithms for anomaly traffic classification based on machine learning. PReSET was designed to allow the simulation-based evaluation of resilience strategies, thus enabling the comparison of optimal configurations and policies for combating different types of attacks (e.g., DDoS attacks, worm propagations) and other anomalies. In such resilience strategies, policies written in the Ponder2 language can be used to activate/reconfigure traffic classification modules and other mechanisms (e.g., traffic shaping), depending on monitored results in the simulation environment. Our results show that PReSET can be a valuable tool for network operators to evaluate anomaly traffic classification techniques in terms of standard performance metrics.**

I. Introduction

Computer and communication networks are becoming increasingly important in supporting business, leisure and everyday activities in general. Network resources, such as bandwidth, need to be carefully dimensioned with respect to the demands of applications and characteristics of network traffic. Moreover, due to the possibility of cyber-attacks and security threats, there is a growing need for resilience to become a key property in computer networks. Resilience is the ability of the network to maintain acceptable levels of operation, in the face of anomalies such as malicious attacks, operational overload, configuration problems or equipment failures [1].

Resilience strategies can be defined in terms of the configuration of mechanisms for detection and remediation. On the one hand, *detection mechanisms* such as link monitors, anomaly detection systems and traffic classifiers allow the identification and characterisation of network conditions. On the other hand, *remediation mechanisms* such as traffic limiters are used in the subsequent mitigation of undesirable characteristics in the network. Resilience management requires the configuration of these mechanisms to be dynamically refined when new information about the network becomes available in response to, for example, high resource utilisation, performance degradation or application specific alarms.

In particular, traffic classification corresponds to a set of techniques and algorithms that aim to categorise network traffic. These techniques can be broken down into several domains, including Internet application protocol classification (i.e., classifying transport flows according to their corresponding application layer protocol), packet classification (i.e., categorising packets into transport flows), and traffic classification for anomaly detection (i.e., separating malicious and non-malicious flows). According to the result of the classification, traffic belonging to a given class can be treated differently. Due to the variety of applications, protocols and traffic profiles involved, an approach that can adapt and learn from past experiences is desirable. With this in mind, *machine learning* techniques show a promising trend in this field [2].

As part of an integrated framework [3] for network resilience, PReSET [4] was designed in order to allow the *off-line* evaluation of resilience strategies, through a simulation environment. It allows network operators to analyse and identify optimal configurations to combat different types of attacks and other anomalies. PReSET (***P****olicy-driven* ***Re****silience* ***S****trategy* ***E****valuation* ***T****oolset*) comprises a series of network components implementing resilience functions and services, integrated into a policy-based management (PBM) framework. PBM [5] can be used to control the operation of these mechanisms, and to specify how they should be reconfigured dynamically as information about the state of the network is obtained.

In this paper, we extend PReSET to allow the evaluation of anomaly traffic classification techniques based on machine learning. Our primary contribution is to offer to network operators and administrators a toolset for the simulation and analysis of a variety of anomaly classification algorithms, thus allowing the easy identification of the best configuration parameters and network policies, when different types of attacks and anomalies are simulated. Furthermore, policies written in the Ponder2 [6] language can then be used to enable/disable classification modules, or replace the algorithm being used, depending on the quality of the monitored results in PReSET. As a result, network operators can be more confident when deploying the actual mechanisms and configuration policies in the physical network. We focus on two algorithms that have been broadly used for the classification of network traffic [2]: K-means and Naïve Bayes, which are used to categorise malicious and benign network traffic behaviour.

This paper is organised as follows: Section II presents an overview of PReSET. Section III describes the general problem of network traffic classification, as well as K-means and Naïve

Bayes. Section IV describes our implementation and evaluation results. Section V outlines some related work, while Section VI concludes the paper.

II. PReSET: A Network Resilience Simulator

PReSET [4] has been developed to allow the simulation of policy-based resilience strategies. It is based on the integration of the OMNeT++ network simulator [7] and the Ponder2 policy framework [6]. PReSET supports the simulation of network attacks and anomalies, and the evaluation of the corresponding resilience strategies. Ponder2 policies can be used to orchestrate the behaviour of resilience mechanisms, e.g., components for intrusion detection and rate limiting, which are implemented as OMNeT++ modules. The coupling of these two technologies enables optimal resilience strategies, configuration parameters and policies to be identified, which can then be easily ported to physical devices.

Events generated by resilience mechanisms running within OMNeT++ are sent to the policy framework using a socket connection. These events indicate conditions observed, such as the detection of an attack. An event can trigger one or more *event-condition-action* (ECA) policies, and the actions specified by a policy will determine which resilience mechanism, running in the simulation, should be reconfigured and how. For this, OMNeT++ modules are instrumented with an XML-RPC server, which registers and exports a management interface for each resilience mechanism available. Ponder2 can then use these interfaces to invoke management actions to adapt the operation of simulated components, for example, to adjust the parameters of a traffic classification algorithm. The measurement capabilities of the simulator allow the prompt evaluation of management actions, for example, in terms of key performance indicators. A number of mechanisms have been implemented as OMNeT++ modules, including: *LinkMonitor*, *FlowExporter*, *RateLimiter* and *EntropyDetection* [8]. PReSET is described in details in [4].

III. Network Traffic Classification

Traffic classification techniques are capable of identifying patterns in the sampled network traffic. Their purpose ranges from the identification of malicious traffic up to the categorisation of Internet traffic for QoS support. However, with the increasing sophistication of applications, protocols and traffic profiles, strategies based on port numbers do not offer reliable classification. Furthermore, strategies based on *payload* inspection can be very accurate but at a high processing cost [9]. An alternative is the use of *machine learning*. Traffic classification based on machine learning can be used as part of anomaly detection schemes, in order to separate flows into classes. Although we only show in this paper binary anomaly classification – malicious and non-malicious classes – the same principles can be used to separate network traffic into different classes of anomalies.

In the following we describe the two algorithms used in this study: K-means and Naïve Bayes. We chose these due to their simplicity and broad use in several investigations [2]. However, PReSET is extensible and other algorithms can be used instead, as discussed in Section IV.

A. K-means

K-means is a clustering algorithm that considers that data can be described by n general features. The grouping of these features into a tuple of size n represents a point in an n-dimensional space. A *centroid* is determined from the average of the m closest points. Each centroid is represented as c_i for $i = 0, ..., k$, where k is the maximum number of centroids that can be created during the clustering process. Each sample p has its centroid c_p defined by the following equation:

$$c_p = min_{i=0}^{k} \sqrt{\sum_{j=0}^{n} (p_j - (c_i)_j)^2}$$

When a point is associated with the nearest centroid, the centroid coordinates will be updated, which in turn will trigger the recalculation of the closest centroid for all the existing samples. Due to its simplicity, K-means is frequently used for traffic classification, and typically few features are needed to describe a flow in K-means [2].

After all centroids have been calculated, it is still necessary to label the clusters, since K-means is only able to group similar data with respect to certain features. Although K-means can be used for Internet traffic classification in general (e.g., VoIP, P2P), we focus on separating benign from malicious traffic because of the impact it has in network resilience.

B. Naïve Bayes

Naïve Bayes uses the theory of *conditional probability* of a sample s belonging to class t. It is calculated as:

$$P(t|s) = \frac{P(s|t) * P(t)}{P(s)}$$

Where $P(t|s)$ is the conditional probability of class t given the occurrence of sample s, $P(s|t)$ is the conditional probability of a sample s given the occurrence of class t, and $P(t)$ and $P(s)$ are the probabilities of occurrence of class t and sample s, respectively. Assuming a set of classes and s as a data sample, a *bayesian* classifier estimates the *most likely class* C_{ML} considering the s_i constituent features of sample s, where $i = 0, ..., n$ in the following manner:

$$C_{ML} = argmax_{t \in T} \frac{P(s_1, s_2, s_3, ..., s_n | t) * P(t)}{P(s)}$$

Naïve Bayes uses the *naïve* hypothesis that the probability associated with each flow feature is independent from the others for a given class t. It is a supervised algorithm, which means it must be provided with a classification model called *training set*. The training set is a database where each entry has a sample value for a given flow feature and a class label. Our training set is based on the attack traces obtained from simulations in PReSET.

IV. Implementation and Experimental Results

K-means and Naïve Bayes have been implemented and integrated in PReSET. We analyse two main aspects regarding the use of these algorithms: *(a)* which flow features are necessary for the classification; *(b)* what are the performance metrics of these algorithms. We chose these algorithms to illustrate how accurate flow classification can be observed with the proposed classifier in the PReSET toolset, rather than showing an exhaustive list of machine learning algorithms.

A. Prototype Implementation

Figure 1 presents an overview of our `Classifier` implementation and its integration within PReSET. The `Classifier` module runs alongside other resilience components (e.g., `FlowExporter`) deployed in the simulated topology. It receives periodically (customised via a policy) flow records from the `FlowExporter` component. The `Classifier` pre-processes the received flows, by extracting the relevant features (feature selection is discussed in Section IV-B) and calculating their statistics. The `Classifier` is extensible and a range of anomaly classification algorithms can be used. Whenever a decision about a malicious flow is made, an event is published into Ponder2, and ECA policies will be used to decide how the network should be reconfigured (e.g., specify that packets belonging to the malicious flow must be dropped). To support the invocation of reconfiguration actions by policies, XMLRPC adaptors are used to interface with simulated components.

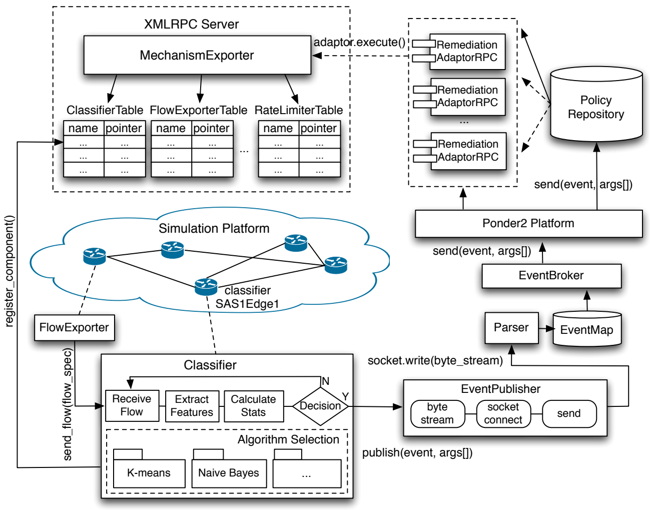


Fig. 1. Overview of the Classifier implementation and PReSET integration

The scenario simulated was a *Distributed Denial of Service* (DDoS) attack, which is based on the set of programs *Tribe Flood Network* [10]. The general characteristics of this simulation scenario are summarised in Table I. In this scenario, some parameters of the simulation were changed (such as start time of the attack), generating sub-scenarios of interest.

TABLE I. SIMULATION AND ATTACK TRAFFIC PARAMETERS

Parameter	Value
simulation total time	160 s
attack start time	40 s
malicious packet size	64 bytes
port to attack	80
probability of attack	0.1

To simulate large-scale IP networks and attacks, PReSET relies on ReaSE [3], which permits the creation of realistic topologies and the generation of background and attack traffic. The simulation in our experiments included 912 hosts, 82 web servers and 188 routers. The traffic profile used in our experiments is described in Table II.

TABLE II. TRAFFIC PROFILE PARAMETERS

Parameter	Value
type	Web Traffic
request length	200 bytes
requests per session	10
reply length	1000 bytes
time between requests	2 s
time to respond	0.5 s
time between sessions	3 s

In total, 843 flows were observed, of which 32 were malicious. Malicious flows come from malicious hosts launching the DDoS attack. It is noteworthy that each flow is considered unidirectional.

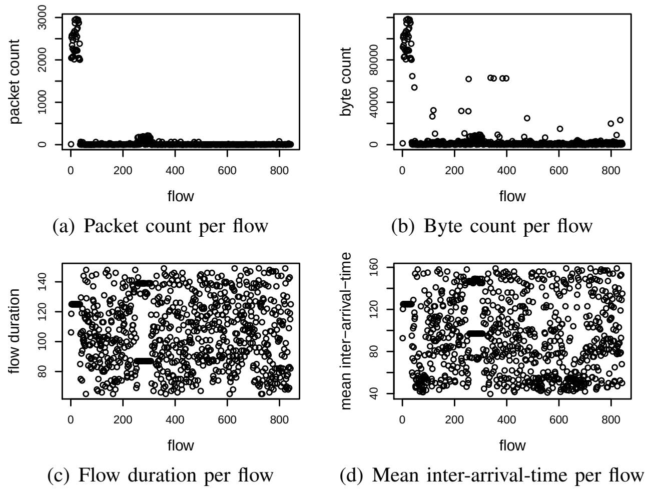


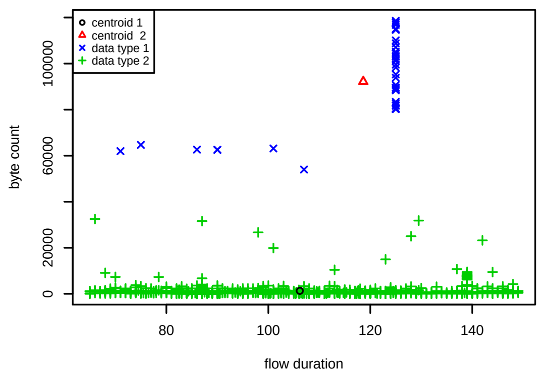
Fig. 2. All features distribution

B. Preliminary analysis and feature selection

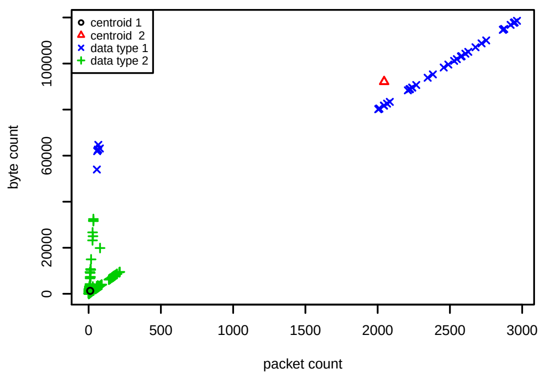
Each flow is represented by the 5-tuple ⟨*destination IP, source IP, destination port, source port, protocol*⟩. During the simulation, the following four traffic features were collected for each flow: *packet count*, *byte count*, *flow duration* and *mean packet inter-arrival-time*. Although *feature selection* is typically a challenging issue, it is still possible to find a set of relevant features by inspection. Based on the data collected from the simulator, a two-dimensional plot on each feature was produced, and the distribution of data was observed. These results can be seen in Figure 2, which shows all flows with their respective feature values. Observing the distributions, the *packet count* and *byte count* features seem to more clearly distinguish different flow samples. In order to validate the classification given by the learning algorithms in this paper, all the malicious flows were manually identified. This was possible due to the ease of customisation offered by PReSET. This manual classification could then be used to benchmark the performance of the algorithms analysed in the next sections.

C. K-means analysis

The K-means implementation for the `Classifier` was executed on all 843 flows involved in the PReSET simulation.

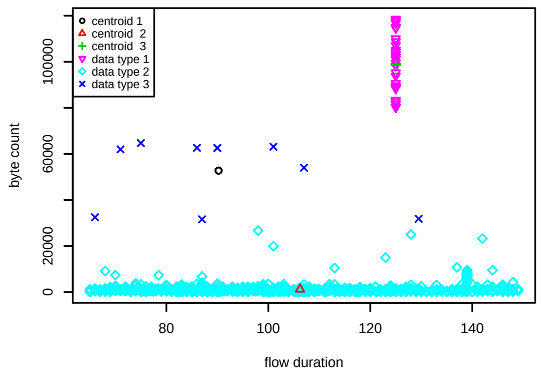


(a) flow duration vs. byte count

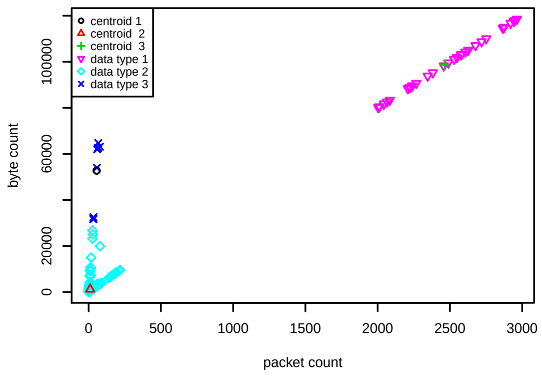


(b) packet count vs. byte count

Fig. 3. K-means classification k=2

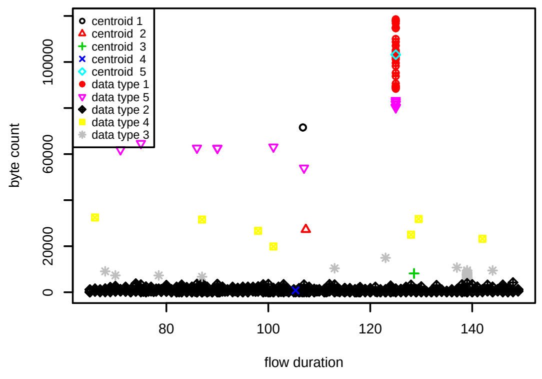


(a) flow duration vs. byte count

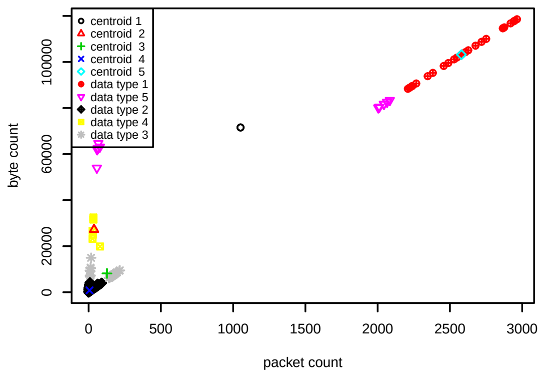


(b) packet count vs. byte count

Fig. 4. K-means classification k=3



(a) flow duration vs. byte count



(b) packet count vs. byte count

Fig. 5. K-means classification k=5

The primary goal of this experiment was to identify malicious flows, by separating them in a specific centroid. In a first analysis, we used K-means configured with two centroids ($k = 2$), representing malicious and benign traffic. The result of this clustering is illustrated in Figure 3. In total, all flow features defined in Section IV-B were used and all possible 2 by 2 combinations for plotting the data were considered. However, due to space limitations, Figure 3 presents only the cases in which a better separation of clusters was observed. In particular, we confirmed that *byte count* and *packet count* were good discriminators for the flow nature. By using manual flow labelling we benchmarked the results in Figure 3. We noted that K-means clustered the benign flows in centroid 1, and the malicious flows in centroid 2. However, it also wrongly assigned a few benign flows to centroid 2.

The use of K-means with a larger number of centroids enables an increase in the degree of clustering, thereby allowing more homogeneous centroids. For 3 centroids ($k = 3$), flows that were previously misclassified now belong to a specific centroid. The final clustering in this case is shown in Figure 4, demonstrating that the isolation of the centroids representing malicious flows is achieved (in centroid 3, data type 1). Since

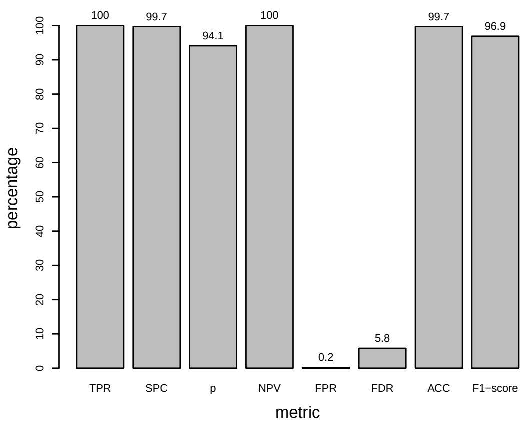


Fig. 6. Classification performance metrics for K-means with 3 centroids

K-means groups flows using feature information, the centroids generated should represent flows that share a similar profile.

For the execution of K-means with three centroids, Figure 6 summarises our results with respect to the standard *performance metrics* for classification algorithms [11]. The performance metrics commonly used are true positive rate (TPR), specificity (SPC), precision (p), negative predictive value (NPV), false positive rate (FPR), false discovery rate (FDR), accuracy (ACC) and F1-score. False positive flows are those that should have been marked as benign, but were marked as malicious. In contrast, false negatives are flows that should have been classified as malicious, but were marked as benign. According to Figure 6, 0.2% of the flows were misclassified (FDR) using the K-means implementation.

However, the use of K-means poses some difficulties. The first issue is to determine the optimal number of centroids. To illustrate this, we extended our analysis. Figure 5 illustrates the result when we run K-means with five centroids ($k = 5$). Notice that new traffic classes arise even within the centroid representing only the group of malicious flows, which is due to *overfitting* the model. Also, there is an issue in defining the optimal set of features to use, and in our case this decision was made by analysing the results of clustering through the manual inspection of the plots. Finally, K-means requires an oracle to provide the labelling and the nature of a centroid; this is because at the end of a run all centroids are determined, but the information they represent is not known.

D. Naïve Bayes analysis

Naïve Bayes has been implemented as a complementary approach to the classification provided by K-means, mainly to assist in the labelling of the centroids produced. A training set containing samples of malicious traffic manually classified and other general samples was used. This training set consisted of 562 flows, divided into malicious and non-malicious samples. During the test phase, all 843 flows were used.

The algorithm calculates the conditional probability of a given flow being malicious or not using the training set. Given a flow f with a feature value v, an interval threshold t (defined a priori) is used such that all entries that are within the interval $[v - t, v + t]$ are considered candidates for labelling f. These entries are separated in malicious and non-malicious and used to calculate the most likely class for f using the conditional probability for every feature. The results of this classification can be seen in Figure 7 where each flow has a likelihood of being malicious *and* non-malicious. Notice that every flow has two probabilities calculated: malicious and non-malicious probability[1]. The final step of the Naïve Bayes algorithm, after calculating the probabilities for each class, chooses the most probable class for a flow and outputs this class. When the training set is unable to provide sufficient information for the classification, a zero probability is returned and in case of tied probabilities, none of the classes is returned and the PReSET classifier module assumes that the flow is non-malicious.

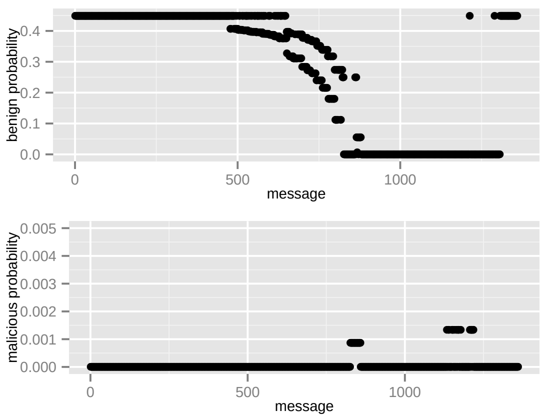


Fig. 7. Classification of Naïve Bayes for probability involved

Figure 8 summarises the standard classification performance metrics for the Naïve Bayes algorithm. According to Moore and Zuev [12], a precision of about 60% is expected for an implementation of Naïve Bayes without advanced techniques, such as *kernel estimation*. Therefore, the results obtained with this implementation are as expected.

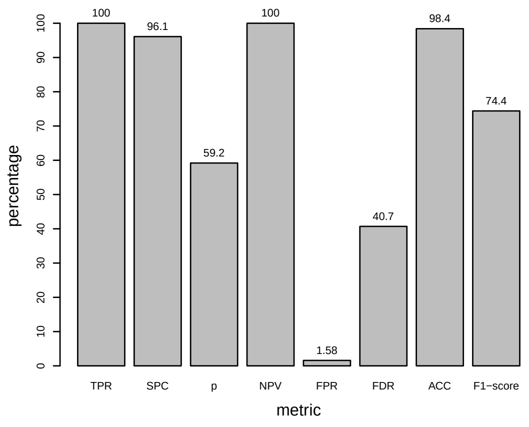


Fig. 8. Classification performance metrics for the Naive Bayes algorithm

E. Discussion

New flows could be captured by K-means and assigned to a centroid. In this case, it would be enough for the Naïve Bayes algorithm to classify at least one flow within the centroid

[1]Naïve Bayes can be used to classify flows. However, it also provides classification of individual messages. In this case, a flow can be considered malicious when a certain limit of messages classified as malicious is exceed.

to identify all other remaining flows in this cluster. Samples that could not be classified by Naïve Bayes could be clustered by K-means and thereby have their class inferred from other flows within the same cluster. Given the complementary characteristics of K-means and Naïve Bayes, their joint use seems to be a viable option. As part of our future work, we will investigate how PReSET can be used to combine different classification techniques. A key feature of PReSET is that it allows the use of policies written in Ponder2 to dynamically reconfigure resilience mechanisms in response to monitored events in the simulation. Future work will also investigate how the algorithms analysed in this paper can be reconfigured through policies [8] (e.g., dynamically altering the parameters of the algorithms or recorded traffic features, according to the performance of the classifier). Furthermore, more complete resilience strategies will be evaluated, by having policies to block or limit malicious traffic that is identified.

V. Related work

Gamer and Mayer [13] describe the use of OMNeT++ to evaluate the detection of large-scale network attacks, such as DDoS and *worm* propagations. In particular, OMNeT++ has been integrated with Distack [14], a framework for evaluating detection mechanisms. These mechanisms are based on a combination of shared libraries for basic functions, such as packet inspection, filtering, and sampling. PReSET itself uses some of the functionality provided by Distack. Further, Lam et al. [15] present techniques for the evaluation of resilience strategies against a variety of attacks also in OMNeT++. The authors demonstrate metrics to assign *scores*, which identify the parameters that influence the performance of resilience strategies. While these investigations have similarities with the theme of our paper (evaluation of resilience strategies in simulation environments), our work differs in the sense that it focuses specifically on the evaluation of traffic classification mechanisms. The contributions presented in this paper not only complement the work in [13], [14] and [15], but also extend PReSET with the ability to allow the development and evaluation of strategies for anomaly traffic classification.

VI. Conclusions

Traffic classification is an important component in strategies used to ensure network resilience against, for example, malicious attacks. Classification allows categorising the network traffic in a number of *classes*. Further, as result of the classification, traffic belonging to a particular class can be treated differently (e.g., dropped or rate limited). In this paper, two algorithms for network traffic classification based on *machine learning* were investigated, compared and implemented in PReSET.

Our main goal with this paper is to offer to network operators a toolset for the simulation and analysis of anomaly traffic classification algorithms, thus allowing the easy identification of the best algorithms, configuration parameters and network policies, when different types of attacks and anomalies are simulated. The toolset supports the analysis and comparison of classification techniques. By using characteristics observed in the simulation, for example, the required sampling rate, the computational cost, the classification accuracy and the percentage of false positives, PReSET permits the execution of reconfiguration mechanisms. As future work, we will evaluate policies written in Ponder2 to enable/reconfigure classification modules, depending on the quality of the results. We also plan to extend PReSET with alternative classification algorithms, including Support Vector Machine (SVM) and AutoClass [2]. This will permit a more systematic and comprehensive comparison of classification techniques.

References

[1] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2010.03.005

[2] T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008. [Online]. Available: http://dx.doi.org/10.1109/SURV.2008.080406

[3] A. Schaeffer-Filho, P. Smith, A. Mauthe, D. Hutchison, Y. Yu, and M. Fry, "A framework for the design and evaluation of network resilience management," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 401–408.

[4] A. Schaeffer-Filho, A. Mauthe, D. Hutchison, P. Smith, Y. Yu, and M. Fry, "PReSET: A toolset for the evaluation of network resilience strategies," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, 2013, pp. 202–209.

[5] W. Han and C. Lei, "A survey on policy languages in network and security management," *Computer Networks*, vol. 56, no. 1, pp. 477 – 489, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128611003562

[6] K. Twidle, E. Lupu, N. Dulay, and M. Sloman, "Ponder2 - a policy environment for autonomous pervasive systems," in *POLICY '08: IEEE Workshop on Policies for Distributed Systems and Networks*. Palisades, NY, USA: IEEE Computer Society, 2008, pp. 245–246.

[7] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *SIMUTools '08: Proceedings of the 1^{st} International Conference on Simulation Tools and Techniques*. Marseille, France: ICST, 2008, pp. 1–10.

[8] Y. Yu, M. Fry, A. Schaeffer-Filho, P. Smith, and D. Hutchison, "An adaptive approach to network resilience: Evolving challenge detection and mitigation," in *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*, 2011, pp. 172–179.

[9] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, ser. MineNet '06. New York, NY, USA: ACM, 2006, pp. 281–286. [Online]. Available: http://doi.acm.org/10.1145/1162678.1162679

[10] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms: Classification and state-of-the-art," *Comput. Netw.*, vol. 44, no. 5, pp. 643–666, Apr. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2003.10.003

[11] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[12] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1071690.1064220

[13] T. Gamer and C. P. Mayer, "Large-scale evaluation of distributed attack detection," in *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–8.

[14] T. Gamer, C. P. Mayer, and M. Zitterbart, "Distack – a framework for anomaly-based large-scale attack detection," in *SECURWARE '08: Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 34–40.

[15] D. Lam, E. Skiles, and P. Grisham, "Simulation tool for evaluation and design of resilience strategies," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, 2013, pp. 186–191.

AppendixB PUBLISHED PAPER – NCA 2015

Software-Defined Networking (SDN) aims to alleviate the limitations imposed by traditional IP networks by decoupling network tasks performed on each device in particular planes. This approach offers several benefits, such as standard communication protocols, centralized network functions, and specific network elements, for example, controller devices. Despite these benefits, there is still a lack of adequate support for performing tasks related to traffic classification, because (i) there are traffic profiles that are very similar, which makes their classification difficult (e.g., both HTTP and DNS flows are characterized by packet bursts); (ii) OpenFlow, the key SDN implementation today, only offers native flow features, such as packet and byte count, that do not describe intrinsic traffic profiles; and (iii) there is a lack of support to determine what is the optimal set of flow features to characterize different types of traffic profiles. In this paper, we introduce an architecture to collect, extend, and select flow features for traffic classification in OpenFlow-based networks. The main goal of our solution is to offer an extensive set of flow features that can be analyzed and refined and to be capable of finding the optimal subset of features to classify different types of traffic flows. The experimental evaluation of our proposal shows that some features emerge as meaningful, occupying the top positions for the classification of distinct flows in different experimental scenarios.

- **Title –**
  Identification and Selection of Flow Features for Accurate Traffic Classification in SDN
- **Conference –**
  The 14th IEEE International Symposium on Network Computing and Applications (IEEE NCA15)
- **Type –**
  Main track (full-paper)
- **Qualis –**
  B1
- **Date –**
  September 28-30, 2015
- **Held at –**
  Cambridge, MA USA
- **Digital Object Identifier (DOI) –**
  <10.1109/NCA.2015.12>

Identification and Selection of Flow Features for Accurate Traffic Classification in SDN

Anderson Santos da Silva, Cristian Cleder Machado,
Rodolfo Vebber Bisol, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho
Institute of Informatics – Federal University of Rio Grande do Sul – Porto Alegre, Brazil
Email: {assilva, ccmachado, rvbisol, granville, alberto}@inf.ufrgs.br

Abstract—**Software-Defined Networking (SDN) aims to alleviate the limitations imposed by traditional IP networks by decoupling network tasks performed on each device in particular planes. This approach offers several benefits, such as standard communication protocols, centralized network functions, and specific network elements, for example, controller devices. Despite these benefits, there is still a lack of adequate support for performing tasks related to traffic classification, because (*i*) there are traffic profiles that are very similar, which makes their classification difficult (*e.g.*, both HTTP and DNS flows are characterized by packet bursts); (*ii*) OpenFlow, the key SDN implementation today, only offers native flow features, such as packet and byte count, that do not describe intrinsic traffic profiles; and (*iii*) there is a lack of support to determine what is the optimal set of flow features to characterize different types of traffic profiles. In this paper, we introduce an architecture to collect, extend, and select flow features for traffic classification in OpenFlow-based networks. The main goal of our solution is to offer an extensive set of flow features that can be analyzed and refined and to be capable of finding the optimal subset of features to classify different types of traffic flows. The experimental evaluation of our proposal shows that some features emerge as meaningful, occupying the top positions for the classification of distinct flows in different experimental scenarios.**

Keywords*—*Flow Feature; Feature selection; Traffic Classification; Software-Defined Networking.

I. Introduction

Traffic classification assists network providers in guaranteeing quality of service (QoS), detecting malicious attacks, reallocating network resources, and performing traffic modeling [1]. In order to achieve more accurate traffic classification, it is important to retrieve precise information about individual traffic flows features, which include, for example, the average packet transmission time. Retrieving relevant network information in traditional IP networks presents several challenges. First, heterogeneous network devices, such as switches and routers, often expose flow information through proprietary management interfaces. Second, to tune how monitoring data is handled internally to devices, one needs to configure each device individually. Finally, because of the distributed control state in forwarding devices, the features of a flow can be changed along its transmission, *e.g.*, ending up with modifications on TCP/IP header fields or facing packet drops, thus causing unanticipated network behavior in subsequent hops.

Software-Defined Networking (SDN) offers a reformulation of the network control logic and alleviate the limitations imposed by traditional IP networks [2]. In SDN, the control plane is decoupled from the forwarding plane, *i.e.*, part of the control logic is moved from the forwarding devices to a logically centralized device often referred to as the network controller. In this approach, every control decision is taken by the network controller, while network devices become simple packet forwarders, programmable through a standardized protocol, such as OpenFlow [3]. However, despite the benefits brought by SDN, there is still an important lack of adequate support for performing tasks related to traffic classification in OpenFlow-based networks. Achieving high accuracy of traffic classification in SDN is difficult for several reasons: (*i*) there are traffic profiles that are very similar, which makes their classification difficult, *e.g.*, both HTTP and DNS flows are characterized by packet bursts; (*ii*) the native flow features available in OpenFlow, such as packet and byte counts, do not convey sufficient information to accurately distinguish between some types of flows; and (*iii*) there is a lack of support to determine what is the optimal set of flow features to characterize different types of traffic profiles.

Considering the aforementioned issues, we introduce in this paper an architecture to identify, extend, and select sets of flow features derived from native OpenFlow counters as well as from mathematical statistics, *e.g.*, mean, variance, maximum and minimum values, 1^{st} and 3^{rd} quartiles, and Fourier Transform. Our main objective is to offer an architecture capable of defining the optimal subset of flow features to identify different types of flows, thus helping to improve the accuracy of network tasks, such as anomaly detection and QoS enforcement. The contribution of this paper is threefold: (*i*) to devise a module capable of gathering and identifying network flow characteristics; (*ii*) to create a set of advanced flow features to characterize traffic profiles; and (*iii*) to investigate the use of two different feature selection techniques, the *Principal Component Analysis* (PCA) and the *Genetic Algorithm* (GA), to determine the subset of flow features that is more appropriate for classifying a particular traffic profile. Some of the traffic features proposed in this paper are indeed more meaningful than the native flow counters provided by OpenFlow, and allow a more accurate classification of different types of traffic. Further, our experimental results show that different subsets of flow features can yield a more accurate traffic classification depending on the traffic profile.

The remainder of this paper is organized as follows. In Section II, we discuss the related work. In Section III, we present our proposed solution for the identification and selection of traffic flow features in OpenFlow-based networks. In Section IV, we describe in details the experiments and initial results. Finally, in Section V, we present concluding remarks and outline the future work.

II. Background and Related Work

Traffic classification algorithms use as input the flows to be classified as well as flows' descriptive features. The accuracy of classification algorithms depends not only on the quality of collected features but also on the amount of features considered too. Too few features hinders more precise classifications; too many features hinders that as well, because of noise caused by an excessive number of variables (*i.e.*, features) being considered. As such, there is an optimal number of features that should be taken into account, and finding such an optimal number is a research challenge per se. To achieve that, feature selection techniques are necessary, and they have been investigated for many years already [4] [5]. However, network applications change continuously and new traffic profiles are hard to predict. As a result, traffic classification and feature selection become even more challenging.

Blum *et al.* [6] describe several solutions to deal with large amounts of irrelevant features. The authors discuss solutions for feature selection based on (*i*) heuristics methods, (*ii*) filtering of irrelevant data, and (*iii*) weight assignment to features. Fahad *et al.* [7] investigate the advantages of using techniques to select traffic features, such as information gain (IG), gain ratio (GR), principal component analysis (PCA), correlation based feature selection (CBFS), Chi-square, and consistency-based-search (CBS). The performance of each technique was evaluated using metrics such as stability (*i.e.*, selection of the same set of flow features despite producing variations in traffic classification) and similarity (*i.e.*, how similar is each subset of features generated by a technique). Finally, Lanzi *et al.* [8] present feature selection using heuristics, including a solution based on genetic algorithms.

Previous research on feature selection investigated the use of techniques and algorithms to assist traffic classification in traditional IP networks. These research efforts suffered from numerous limitations imposed by traditional IP networks, such as (*i*) distributed control state in forwarding devices, (*ii*) proprietary decision-making logic, and (*iii*) heterogeneous protocols and interfaces. These limitations have an impact on solutions that aim, for example, to obtain an overall view of the network traffic since it is difficult and expensive to collect decentralized and non-standard information. The emergence of the SDN paradigm brought new possibilities to assist traffic classification and feature selection. As an example, flow statistics from each switch can be gathered through standard interfaces and protocols, such as OpenFlow. However, specific solutions to feature selection specifically in SDN environments have not been investigated yet. Thus, our goal is to explore the native collection of flow statistics and extend it to obtain more appropriate features for traffic classification.

III. An Architecture for Flow Feature Identification and Selection in SDN

In this section, we propose an architecture to collect, extend, and select flow features for traffic classification in Software-Defined Networking (SDN). Its main goal is to offer an extensive set of flow features that can be analyzed and refined, and to be capable of finding the optimal subset of features to classify different types of traffic flows. We consider a flow as a 5-tuple consisting of source IP address, destination IP address, TCP/UDP source port, TCP/UDP destination port, and transport protocol identification. It is important to emphasize that this tuple can be adjusted according to the needs of each network infrastructure. Thus, some approaches can ignore information such as destination TCP/UDP port when the network infrastructure performs only one type of service, *e.g.*, HTTP.

A. Architecture Overview

Our proposed architecture is depicted in Figure 1. It comprises two main components: (*i*) *Flow Feature Manager*, which is responsible for handling network information and for computing additional flow features; and (*ii*) *Flow Feature Selector*, which is responsible for analyzing each flow feature exposed by the Flow Feature Manager, and for selecting the most meaningful ones to be applied in traffic classification. Note that the implementation of these components, which execute in the *Application Plane*, does not require modifications to the SDN controller implementation. Further, these components can communicate with other SDN applications (*e.g.*, monitoring or management systems) regardless of the SDN controller implementation.

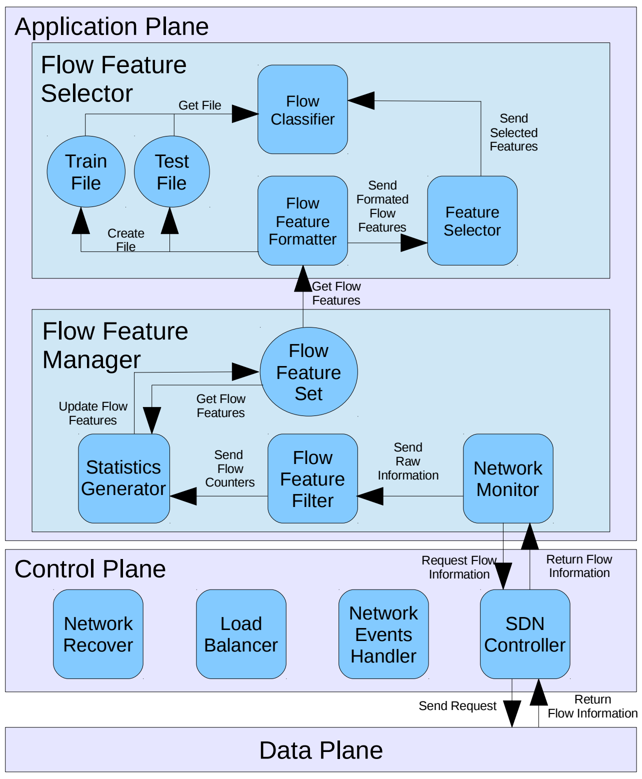


Fig. 1: Architecture for feature identification and selection.

In the following, we describe these components in details:

- **Flow Feature Manager** comprises three modules: (*i*) the Network Monitor, responsible for gathering network information from the network through requests sent to the controller; (*ii*) the Flow Feature

Filter, which filters the information received from the Network Monitor and sends it to the Statistics Generator; and (*iii*) the Statistics Generator, which extends the information gathered in a set of more complex flow features, as discussed later on in this section.

- **Flow Feature Selector** comprises three modules: (*i*) the Flow Feature Formatter, which organizes the data for the feature selection algorithms and automatically creates the training and testing sets for the traffic classification algorithms; (*ii*) the Feature Selector, which is responsible for creating the optimal subset of flow features using a specific technique, *e.g.*, Principal Component Analysis (PCA); and (*iii*) the Flow Classifier that evaluates the flow feature subsets created using traffic classification accuracy as metric.

These components operate as follows. Periodically, the Network Monitor sends to the controller a request for traffic information. The time interval for this request can be configured on demand. For example, flows with short duration and frequent bursts require a smaller time interval between requests, while a longer time interval can be used for flows with longer duration and more constant behavior. The controller then gathers the traffic information from the flow tables of each switch[1] and replies to the Network Monitor. This information is then sent to the Flow Feature Filter, selecting only the native counters, *i.e.*, byte and packet counters. The Statistics Generator uses the native counters gathered from the switches to calculate new flow features (*e.g.*, packet length mean), extend the flow features, and store these in a data structure that we call *Flow Feature Set*.

Subsequently, the process of analyzing the full set of features and selecting the optimal features subset is started. First, the Flow Feature Formatter receives the flow feature set sent by the Flow Feature Manager, and performs two operations: (*i*) it creates a training and a test set that will be used by the Flow Classifier; and (*ii*) it organizes the full flow feature set in the specific format employed by the Flow Feature Selector. The formatted data is used as input to the feature selection algorithms implemented in this module, which are able to identify the most meaningful features for flow classification. Finally, the Flow Classifier module classifies the flows using the feature subset yielded by the Feature Selector to evaluate the fitness of each subset for some traffic profile. Similarly to the Feature Selector module, the Flow Classifier also supports the implementation of a range of different algorithms.

In our proof-of-concept implementation, the Flow Feature Selector is instantiated with (*i*) *Principal Component Analysis* (PCA) and (*ii*) *Genetic Algorithm* (GA). The PCA algorithm [10] evaluates the relationships between a set of variables, determining the variability associated with each of them. PCA is used as a strategy to reduce the number of variables by removing correlated values. The result of PCA is a subset of the original variables, called principal component, which accounts for the most significant variance in the original dataset. The Genetic Algorithm mimics natural evolution and combines current best solutions for producing new solutions, which are then analyzed to assess their quality. Briefly, the algorithm creates an initial population representing a set of solutions. During each iteration, called generation, it selects the fittest solutions and creates new ones through a combination (crossover) or mutation [11]. A series of parameters must be determined a priori, such as population size and mutation probability. Although we used two well-known algorithms for realizing the Flow Feature Selector module, we emphasize that this module can support a wide-range of other algorithms.

B. Extended Flow Features

OpenFlow is currently the most important protocol for SDN implementation [12]. Although traffic classification can be performed using the native traffic counters provided by OpenFlow (*e.g.*, packet counts), this approach presents several limitations, since these counters do not allow detecting atypical traffic behavior, such as packet bursts. Because of the wide range of possible traffic profiles, more descriptive traffic discriminators are necessary. In this paper, we propose an extension of the native OpenFlow counters through statistics analysis, with the goal of producing more descriptive information about the traffic behavior, such as bytes per second mean and packets per second variance. More specifically, we rely on the standard byte and packet counts retrieved from the network controller, but create an enhanced flow feature set. We introduce a third type of counter, named *samples counter*, which indicates the number of times each flow appears during feature polling. This counter is maintained by the Statistics Generator and is essential to compute some of the more advanced flow features, as discussed below.

The extended set of flow features introduced in this paper is classified into three categories: (*i*) *statistical features* – mean, variance, first, and third quartiles; (*ii*) *scalar features* – maximum and minimum values, flow size, and flow duration; and (*iii*) *complex feature* – Discrete Fourier Transform (DFT) of packet inter-arrival-time. Table I presents the complete set comprising 33 new flow features. We advocate that these extended flow features can improve the performance of classification schemes in SDN. First, the statistical features are suitable to summarize the temporal behavior of traffic profiles. For example, the mean packet count represents the central value of a set of observations. Thus, they are better descriptors in comparison to SDN native packet counters. Second, scalar features are convenient to indicate the instantaneous profile of traffic flows, such as duration. As a result, they are important in the detection of short-lived communications or packet bursts, since statistical features alone are not sensitive to these traffic profiles. Finally, complex features are useful to anticipate the need for more sophisticated traffic information. Different from the previous categories, complex features can be used to compress a wide range of traffic observations. For example, Discrete Fourier Transform can be used to refine several measurements of packet's inter-arrival-time into a set of frequency components, which is a more distinguished representation. Some of these features were investigated by Auld *et al.*[13] and Moore *et al.* [14].

OpenFlow's native counters were used as the basis for deriving two new flow features, namely packet length P_l and

[1]In the OpenFlow switch, the flow table stores actions for every incoming packet [9]. The flow table is organized in a structure containing a subset of packets headers associated with each flow information.

TABLE I: Extended set comprising 33 new flow features.

Statistical Features	**Scalar Features**	**Complex Features**
Bytes per second mean	Bytes per second maximum value	Packet inter-arrival-time Fourier Transform 1^{st} Component
Bytes per second variance	Bytes per second minimum value	Packet inter-arrival-time Fourier Transform 2^{nd} Component
Packets per second mean	Packets per second maximum value	Packet inter-arrival-time Fourier Transform 3^{rd} Component
Packets per second variance	Packets per second minimum value	Packet inter-arrival-time Fourier Transform 4^{th} Component
Packets length mean	Packets length maximum value	Packet inter-arrival-time Fourier Transform 5^{th} Component
Packets length variance	Packets length minimum value	Packet inter-arrival-time Fourier Transform 6^{th} Component
Packets length 1^{st} quartiles	Packets inter-arrival-time maximum value	Packet inter-arrival-time Fourier Transform 7^{th} Component
Packets length 3^{rd} quartiles	Packets inter-arrival-time minimum value	Packet inter-arrival-time Fourier Transform 8^{th} Component
Packet inter-arrival-time mean	Flow duration	Packet inter-arrival-time Fourier Transform 9^{th} Component
Packet inter-arrival-time variance	Flow size in packets	Packet inter-arrival-time Fourier Transform 10^{th} Component
Packet inter-arrival-time 1^{st} quartiles	Flow size in bytes	
Packet inter-arrival-time 3^{rd} quartiles		

packet inter-arrival-time[2] P_{it}. These are estimations of the real packet length and real inter-arrival-time, since calculating their precise values would require deep packet inspection and the monitoring of every packet in the network, which are expensive tasks. Consequently, we apply sampling strategies to estimate these features and use Equation 1 and Equation 2 to calculate packet length and inter-arrival, respectively. B_c represents byte count, P_c represents packet count between two requests, and T is the time-interval between two requests.

$$P_l = \frac{B_c}{P_c} \tag{1}$$

$$P_{it} = \frac{T}{P_c} \tag{2}$$

To calculate mean and variance, the Statistics Generator updates the mean μ and the variance σ^2 for the flow features every time a new information request is processed. Mean is calculated using Equation 3, where μ_n is the updated mean, μ_o is the old mean, θ is the new sample, and n is the samples counter.

$$\mu_n = \frac{n * \mu_o + \theta}{n+1} \tag{3}$$

The updated variance σ_n^2 is calculated using Equation 4, where σ_o^2 is the old variance.

$$\sigma_n^2 = \frac{n}{n+1} * (\sigma_o^2 + \frac{(\theta - \mu_n) * (\theta - \mu_o)}{n}) \tag{4}$$

In this work, we calculate DFT of packet inter-arrival-time samples and use the top ten components as flow features according to Auld *et al.* [13] work. DFT represents the samples of a variable in the frequency domain. It is defined by Equation 5, where N is the number of samples, x_n is a sample, and X_k is the resulting component.

$$X_k = \sum_{n=0}^{N-1} x_n . e^{-2\pi ikn/N},\ k \,\epsilon\, \mathbb{Z} \tag{5}$$

IV. Prototype and Experimental Evaluation

In this section, we describe a proof-of-concept implementation, comprising the actual algorithms used to instantiate the modules described in the previous section. We also present our experimental evaluation and discuss initial results.

A. Prototype Implementation

We developed a prototype for the Flow Feature Manager and the Flow Feature Selector components in Python. The sets of flow features are stored in a *dictionary* data structure. To select the optimal flow feature subset, the Feature Selector module implements two distinct algorithms: Principal Component Analysis (PCA) [10] and Genetic Algorithm (GA) [11]. In addition, to analyze the classification accuracy of each subset of flow features, the Flow Classifier module implements the well-known Support Vector Machine (SVM) algorithm

[2]We use packet length and packet inter-arrival-time statistics because individual values do not usually give substantial information about the traffic profile.

for traffic classification [15]. The PCA, GA, and SVM were implemented using R[3] libraries (psych, genalg, and e1071, respectively). Still, each module can be customized with other classification algorithms and feature selection techniques as needed. Finally, for managing and monitoring the network infrastructure, we chose the Floodlight Controller[4] version 1.0, as it offers a suitable communication interface between our application and the controller through a REST API.

B. *Experiments and Initial Results*

Our goal is to measure the accuracy of the resulting traffic classification using specific subsets of traffic features, discovered via the Genetic Algorithm (GA) and the Principal Component Analysis (PCA), as opposed to using the full-blown set of features. Accuracy is defined as the percentage of correctly classified instances among the total number of instances [1]. To evaluate the proposed architecture, we used a standard tree topology with three levels comprising seventeen switches, and sixty-four hosts. We defined four types of traffic profiles in our experiments: (*i*) DDoS attack, (*ii*) FTP traffic, (*iii*) video streaming using VLC Media Player[5], and (*iv*) background traffic generated using Scapy[6]. We composed these types of traffic into three scenarios defined in Table II. For each scenario we use the traffic classification accuracy as metric to define the optimal flow feature set. For comparison purposes, we use as benchmark the classification accuracy for each scenario obtained by using the complete set of flow features: 94.67% (scenario A), 92% (scenario B), and 85.33% (scenario C).

TABLE II: Experimental scenarios.

Scenario	**Type of flow**
A	DDoS attacks (60%) with Scapy flow (40%)
B	FTP traffic (35%) with Scapy flows (65%)
C	Video streaming (50%) with Scapy flows (50%)

In order to produce more accurate results, the Genetic Algorithm (GA) must be initialized with a few parameters. We set the *population size* to 200 individuals randomly generated and the *crossover percentage* to 20%. We also must configure two other parameters: (*i*) *number of iterations* and (*ii*) *mutation probability*. In order to do so, we analyzed the classification accuracy for a number of iterations, namely 10, 50, 75, 100, 150, and 200, until the accuracy stabilized. Figure 2 presents the resulting accuracy obtained for a given number of iterations. As a result, we set the number of iterations to 100, since it is the smallest value with the highest accuracy.

As can be observed in Figure 3, the *mutation probability* does not have an impact in classification accuracy, and we use 0.01 as standard mutation probability. Because of the amount of time taken to execute this algorithm (nearly 88 minutes), we chose scenario C to set these parameters, since it has the lowest classification accuracy when using all flow features. The classification accuracies obtained by using GA to find the

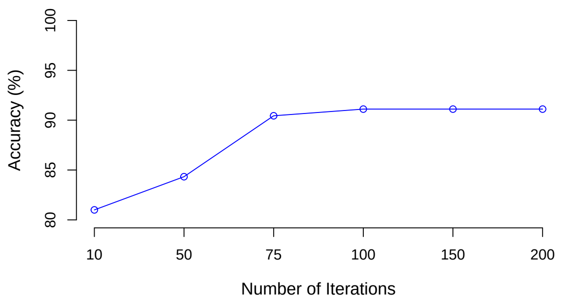


Fig. 2: Accuracy for each number of iterations.

optimal subset of flow features for each scenario were: 98% (scenario A), 94.67% (scenario B), and 91.33% (scenario C).

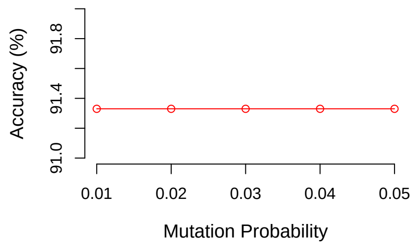


Fig. 3: Accuracy for each mutation probability.

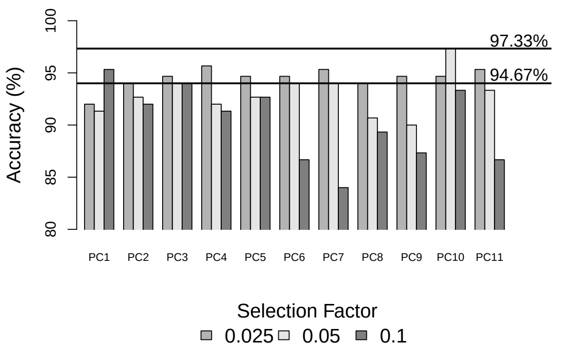


Fig. 4: Accuracy of the 11 most meaningful principal components selected by PCA (for each factor) in scenario A.

To find the optimal subset of flow features, we also applied PCA. PCA determines the most important features by creating one principal component to match each variable (feature), *i.e.*, in our experiments it creates 33 principal components. It works as follows: initially, only the components that jointly represent 90% of the features variability are chosen, resulting in 11 components; then, since a principal component is created as a weighed sum of the features, a subset is created for each component including the features with higher weight than a specific *factor*. In our experiments, we used 0.025, 0.05, and 0.1 as factors. As a result, we have three different subsets for each component. Figure 4 shows the classification accuracy for

[3]www.r-project.com
[4]http://www.projectfloodlight.org/
[5]http://www.videolan.org/vlc/index.html
[6]http://www.secdev.org/projects/scapy/

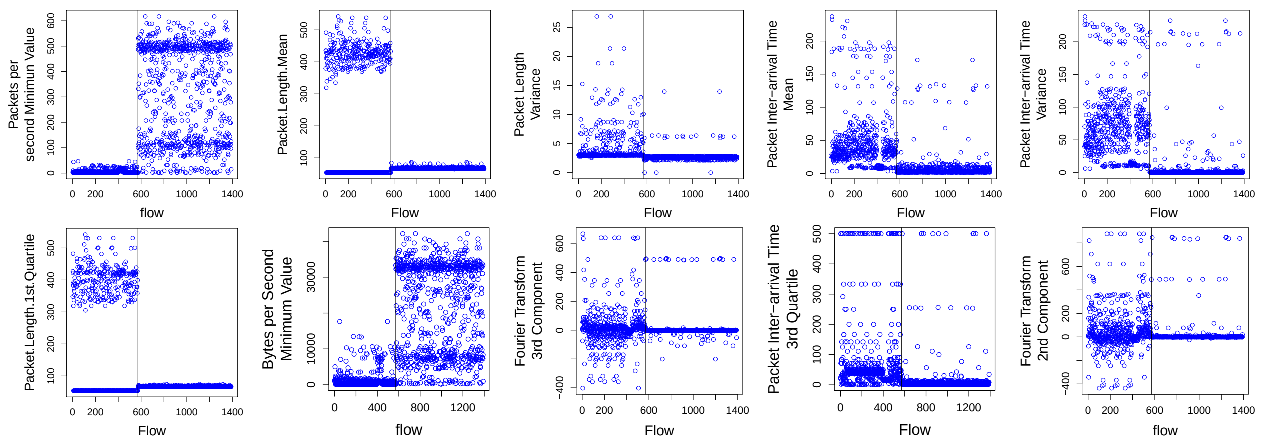


Fig. 5: Ten most meaningful flow features according to PCA analysis in Scenario A (DDoS).

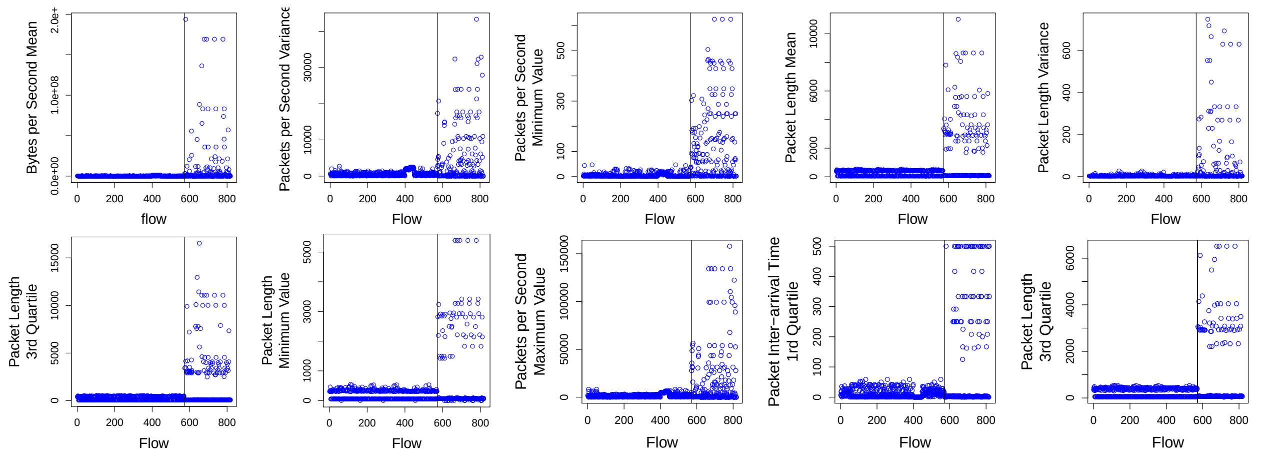


Fig. 6: Ten most meaningful flow features according to PCA analysis in Scenario B (FTP).

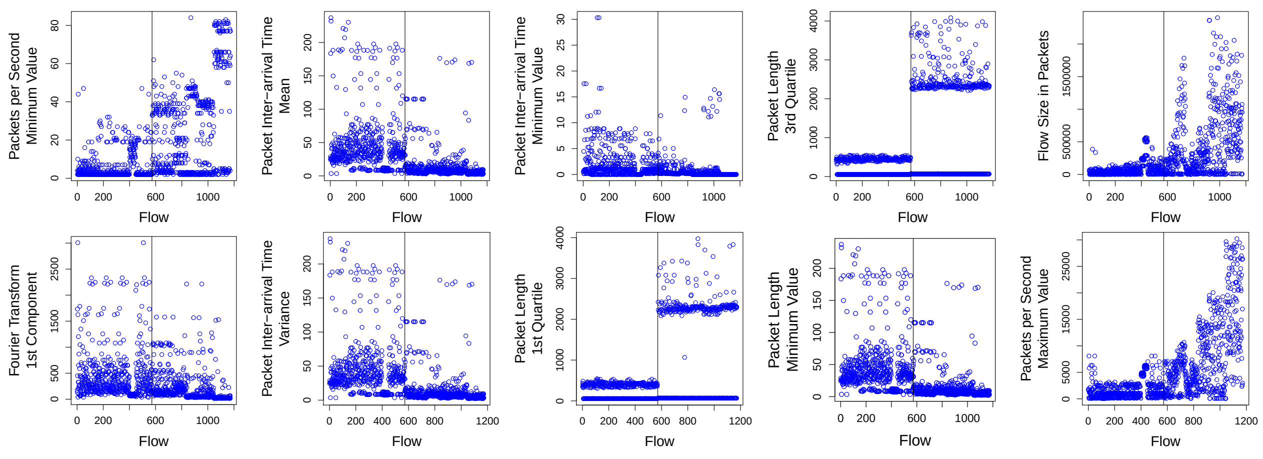


Fig. 7: Ten most meaningful flow features according to PCA analysis in Scenario C (Video Streaming).

each subset in scenario A. We can draw several conclusions from this: (*i*) not all subsets lead to a higher accuracy compared to the accuracy obtained by using the complete flow feature set (94.67%); (*ii*) factor 0.025 seems to lead to the best solutions in most cases, although the optimal solution is a subset of features selected using factor 0.05 in the 10^{th} principal component (97.33%); and (*iii*) factor 0.1 cannot be ruled out because it leads to the highest accuracy in the 1^{st} principal component,

TABLE III: Optimal flow features subsets selected for each experiment.

Scenario A		**Scenario B**		**Scenario C**	
PCA	**GA**	**PCA**	**GA**	**PCA**	**GA**
Bytes per second mean	Bytes per second mean	Bytes per second mean	Bytes per second variance	Bytes per second Minimum value	Bytes per second variance
Bytes per second Minimum value	Bytes per second variance	Bytes per second variance	Bytes per second Maximum value	Packets per second mean	Bytes per second Maximum value
Packets per second Minimum value	Bytes per second Maximum value	Bytes per second Maximum value	Minimum value of Bytes per second	Packets per second variance	Bytes per second Minimum value
Packets Length mean	Bytes per second Minimum value	Bytes per second Minimum value	Packets per second Mean	Packets per second Maximum value	Packets per second variance
Packets Length variance	Packets per second mean	Packets per second mean	Packets per second Variance	Packets per second Minimum value	Packets per second Minimum value
Packet Length Maximum value	Packets per second Maximum value	Packets per second variance	Packets per second Maximum value	Packet Length variance	Packet Length mean
Packet Length Minimum value	Packets per second Minimum value	Packet per second Maximum value	Packet per second Minimum value	Packet Length Minimum value	Packet Length variance
Packet Inter-arrival time mean	Packet Length mean	Packet per second Minimum value	Packet Length Mean	Packet Inter-arrival time mean	Packet Length Maximum value
Packet Inter-arrival time variance	Packet Length variance	Packet Length mean	Packet Length Maximum value	Packet Inter-arrival time variance	Packet Length Minimum value
Packet Inter-arrival time Maximum value	Packet Length Maximum value	Packet Length variance	Packet Length Minimum value	Packet Inter-arrival time Maximum value	Packet Inter-arrival time mean
Packet Inter-arrival time 1^{st} quartile	Packet Length Minimum value	Packet Length Maximum value	Packet inter-arrival time Mean	Packet Inter-arrival time Minimum value	Packet Inter-arrival time Maximum value
Packet Length 1^{st} quartile	Packet Inter-arrival time mean	Packet Length Minimum value	Packet inter-arrival time Variance	Flow Size in Packets	Flow Size in Bytes
Packet Length 3^{rd} quartile	Packet Inter-arrival time Minimum value	Packet Inter-arrival time variance	Flow Size in Bytes	Packet Inter-arrival time 1^{st} quartile	Packet Inter-arrival time 1^{st} quartile
Fourier Transform 2^{nd} Component	Flow Duration	Packet Inter-arrival time Maximum value	Flow Size in packets	Packet Inter-arrival time 3^{rd} quartile	Packet Inter-arrival time 3^{rd} quartile
Fourier Transform 3^{rd} Component	Flow Size in Bytes	Packet Inter-arrival time Minimum value	Packet inter-arrival time 1^{st} quartile	Packet Length 1^{st} quartile	Packet Length 3^{rd} quartile
Fourier Transform 6^{th} Component	Flow Size in Packets	Flow Size in Bytes	Packet Length 1^{st} quartile	Fourier Transform 1^{st} Component	Fourier Transform 1^{st} Component
	Packet Inter-arrival time 3^{rd} quartile	Packet Inter-arrival time 1^{st} quartile	Packet Length 3^{rd} quartile	Fourier Transform 2^{nd} Component	Fourier Transform 2^{nd} Component
	Packet Length 1^{st} quartile	Packet Inter-arrival time 3^{rd} quartile	Fourier Transform 2^{nd} component	Fourier Transform 4^{th} Component	Fourier Transform 3^{rd} Component
	Fourier Transform 1^{st} Component	Packet Length 1^{st} quartile	Fourier Transform 4^{th} component	Fourier Transform 5^{th} Component	Fourier Transform 4^{th} Component
	Fourier Transform 2^{nd} Component	Packet Length 3^{rd} quartile	Fourier Transform 5^{th} component	Fourier Transform 6^{th} Component	Fourier Transform 5^{th} Component
	Fourier Transform 3^{rd} Component	Fourier Transform 2^{nd} Component	Fourier Transform 7^{th} component	Fourier Transform 7^{th} Component	Fourier Transform 8^{th} Component
	Fourier Transform 4^{th} Component	Fourier Transform 3^{rd} Component	Fourier Transform 10^{th} Component	Fourier Transform 8^{th} Component	Fourier Transform 9^{th} Component
	Fourier Transform 5^{th} Component	Fourier Transform 4^{th} Component		Fourier Transform 9^{th} Component	Fourier Transform 10^{th} Component
	Fourier Transform 6^{th} Component	Fourier Transform 5^{th} Component		Fourier Transform 10^{th} Component	
	Fourier Transform 9^{th} Component	Fourier Transform 7^{th} Component			
		Fourier Transform 8^{th} Component			
		Fourier Transform 9^{th} Component			
		Fourier Transform 10^{th} Component			

thus can also result in high accuracy in a different scenario. Based on these three observations, it is not possible to identify which factor gives the best subset for any given scenario, consequently all three factor must be considered.

Because of space constraints, we illustrate only the 10 most meaningful features (in order of importance) selected through PCA in each scenario, depicted in Figure 5, Figure 6, and Figure 7. In each scatter plot, the vertical line separates the types of traffic being simulated: the left side represents the background traffic, and the right side represents DDoS traffic (in Figure 5), FTP flows (in Figure 6), and Video Streaming (in Figure 7). Classification accuracies for the optimal subset of features selected with PCA for each scenario were: 97.33% (scenario A), 94% (scenario B), and 88.67% (scenario C).

Figure 8 summarizes the classification accuracy in each scenario using all features, as well as using the optimal subsets selected with PCA and GA. In all scenarios, the accuracy was improved by using the optimal feature subsets generated by GA and PCA, when compared with the accuracy obtained by using the complete set of flow features. Table III details the feature subsets selected by PCA and GA in each scenario.

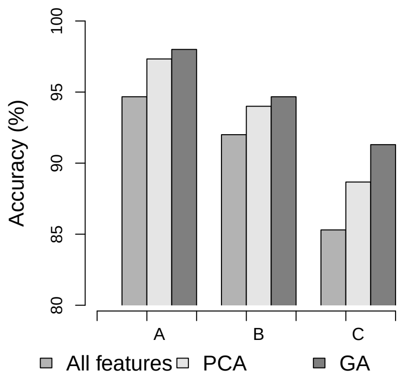


Fig. 8: Traffic classification accuracy for each flow feature set in all three scenarios.

V. Concluding Remarks

In this paper, we introduced an architecture to collect, extend, and select a set of flow features for traffic classification in Software-Defined Networking (SDN). We also demonstrated that for particular types of traffic, an optimal subset of flow features can be selected using both principal component analysis and genetic algorithm.

Our experimental results show that some of the traffic features proposed in this paper are indeed more meaningful than the native flow counters provided by OpenFlow, and allow a more accurate classification of different types of traffic. Also, our results show that these features can be combined, according to each type of flow, and be used for indicating the profile of a particular flow. Furthermore, in all scenarios the subset of flow features selected by either GA or PCA allows a more accurate traffic classification when compared to classification accuracy obtained with the complete set of flow features.

As future work, we will investigate additional flow features, and analyze other types of flow services. Finally, we plan to explore and apply other algorithms and mechanisms for traffic classification and compare them with the results we have collected so far.

Acknowledgement

This work is partially supported by ProSeG - Information Security, Protection and Resilience in Smart Grids, a research project funded by MCTI/CNPq/CT-ENERG #33/2013.

References

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

[2] J. A. Wickboldt, W. P. de Jesus, P. H. Isolani, C. B. Both, J. Rochol, and L. Z. Granville, “Software-Defined Networking: Management Requirements and Challenges,” *IEEE Communications Magazine*, vol. 53, no. 1, pp. 278–285, Jan 2015.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[4] C. Pascoal, M. Rosario de Oliveira, R. Valadas, P. Filzmoser, P. Salvador, and A. Pacheco, “Robust feature selection and robust pca for internet traffic anomaly detection,” in *Proceedings of IEEE INFOCOM*, March 2012, pp. 1755–1763.

[5] M. Mantere, M. Sailio, and S. Noponen, “Feature Selection for Machine Learning Based Anomaly Detection in Industrial Control System Networks,” in *IEEE International Conference on Green Computing and Communications (GreenCom)*, Nov 2012, pp. 771–774.

[6] A. L. Blum and P. Langley, “Selection of Relevant Features and Examples in Machine Learning,” *Journal Artificial Intelligence - Special issue on relevance*, vol. 97, no. 1-2, pp. 245–271, Dec. 1997. [Online]. Available: http://dx.doi.org/10.1016/S0004-3702(97)00063-5

[7] A. Fahad, Z. Tari, I. Khalil, I. Habib, and H. Alnuweiri, “Toward an efficient and scalable feature selection approach for internet traffic classification,” *Computer Networks*, vol. 57, no. 9, pp. 2040 – 2057, 2013.

[8] P. Lanzi, “Fast feature selection with genetic algorithms: a filter approach,” in *IEEE International Conference on Evolutionary Computation*, Apr 1997, pp. 537–540.

[9] Open Networking Foundation, “OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01),” December 31 2009, available at: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Accessed: August 2014.

[10] K. Pearson., “LIII. On lines and planes of closest fit to systems of points in space,” *Philosophical Magazine Series 6*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: http://dx.doi.org/10.1080/14786440109462720

[11] I.-S. Oh, J.-S. Lee, and B.-R. Moon, “Hybrid genetic algorithms for feature selection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1424–1437, Nov 2004.

[12] N. Feamster, J. Rexford, and E. Zegura, “The Road to SDN,” *Queue - Large-Scale Implementations*, vol. 11, no. 12, pp. 20:20–20:40, Dec. 2013. [Online]. Available: http://doi.acm.org/10.1145/2559899.2560327

[13] T. Auld, A. Moore, and S. Gull, “Bayesian Neural Networks for Internet Traffic Classification,” *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 223–239, Jan 2007.

[14] A. W. Moore and D. Zuev, “Internet Traffic Classification Using Bayesian Analysis Techniques,” *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 50–60, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1071690.1064220

[15] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, “Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices,” in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT ’08. New York, NY, USA: ACM, 2008, pp. 11:1–11:12.

AppendixC PUBLISHED JOURNAL – COMPUTER NETWORKS

Software-Defined Networking (SDN) is an architecture for computer networking that provides a clear separation between network control functions and forwarding operations. The abstractions supported by this architecture are intended to simplify the implementation of several tasks that are critical to network operation, such as routing and network management. Computer networks have an increasingly important societal role, requiring them to be resilient to a range of challenges. Previously, research into network resilience has focused on the mitigation of several types of challenges, such as natural disasters and attacks. Capitalizing on its benefits, including increased programmability and a clearer separation of concerns, significant attention has recently focused on the development of resilience mechanisms that use software-defined networking approaches. In this article, we present a survey that provides a structured overview of the resilience support that currently exists in this important area. We categorize the most recent research on this topic with respect to a number of resilience disciplines. Additionally, we discuss the lessons learned from this investigation, highlight the main challenges faced by SDNs moving forward, and outline the research trends in terms of solutions to mitigate these challenges.

- **Title –**
  Resilience Support in Software-Defined Networking: A Survey
- **Journal –**
  Computer Networks
- **Qualis –**
  A1
- **URL –**
  <http://www.sciencedirect.com/science/article/pii/S1389128615003229>
- **Date –**
  October 20, 2015
- **Digital Object Identifier (DOI) –**
  <doi:10.1016/j.comnet.2015.09.012>

Resilience Support in Software-Defined Networking: A Survey

Anderson Santos da Silva[a,*], Paul Smith[b], Andreas Mauthe[c], Alberto Schaeffer-Filho[a]

[a]*Institute of Informatics, Federal University of Rio Grande do Sul, Brazil*
[b]*Safety and Security Department, AIT Austrian Institute of Technology, Austria*
[c]*School of Computing and Communications, Lancaster University, United Kingdom*

Abstract

Software-Defined Networking (SDN) is an architecture for computer networking that provides a clear separation between network control functions and forwarding operations. The abstractions supported by this architecture are intended to simplify the implementation of several tasks that are critical to network operation, such as routing and network management. Computer networks have an increasingly important societal role, requiring them to be resilient to a range of challenges. Previously, research into network resilience has focused on the mitigation of several types of challenges, such as natural disasters and attacks. Capitalizing on its benefits, including increased programmability and a clearer separation of concerns, significant attention has recently focused on the development of resilience mechanisms that use software-defined networking approaches. In this article, we present a survey that provides a structured overview of the resilience support that currently exists in this important area. We categorize the most recent research on this topic with respect to a number of resilience disciplines. Additionally, we discuss the lessons learned from this investigation, highlight the main challenges faced by SDNs moving forward, and outline the research trends in terms of solutions to mitigate these challenges.

Keywords: Software-defined networking; network resilience; OpenFlow; network challenges.

1. Introduction

Computer networks are important for businesses and to support the operation of societally critical infrastructures, such as future (smart) electrical grids and government services. The growth in number and variety of end-to-end services that networks must support has led to a great deal of heterogeneity in the way networks are implemented, resulting in (*i*) complex protocols to handle the communication between network devices [1], (*ii*) difficult deployment of network policies by network administrators [2] and (*iii*) limited routing scalability [3, 4, 5]. Additionally, challenges to normal network operation, such as malicious attacks and prohibitive communication delay, demonstrate that computer networks have long-standing resilience requirements [6].

Resilience is the ability of the network to maintain an acceptable level of service when confronted with operational challenges [7]. A challenge is an atypical event that hinders the expected normal network operation [6, 8]. In order to deal with a wide range of challenges, network resilience encompasses six major disciplines: security, survivability (including fault tolerance), performability, traffic tolerance, disruption tolerance and dependability [7]. When a network challenge arises, mitigation mechanisms should be activated, ideally without human intervention, to rapidly protect a network and the services it supports. However, the broad range of potential challenges that could befall a network requires sophisticated network (resilience) management systems that can detect and mitigate their effects [8]. Existing management systems have limitations, including a lack of flexibility with respect to challenge identification and mitigation, which has encouraged research that considers this problem in the context of new network architectures [9].

In both the research and industry communities, Software-Defined Networking (SDN) [10] has recently gained significant attention. The main characteristic of the SDN architecture is that it decouples the implementation of network control logic from forwarding oper-

*Corresponding author.
Email addresses: `assilva@inf.ufrgs.br` (Anderson Santos da Silva), `paul.smith@ait.ac.at` (Paul Smith), `a.mauthe@lancaster.ac.uk` (Andreas Mauthe), `alberto@inf.ufrgs.br` (Alberto Schaeffer-Filho)

ations, thus enabling more flexible network control and management. In this context, a centralized *control plane* determines how forwarding devices, such as switches, will behave by configuring them using standardized protocols, such as OpenFlow [11]. The SDN architecture and the OpenFlow protocol, as its canonical implementation, offer (*i*) a comprehensive view of the network that is centralized in the *control plane*, (*ii*) high-levels of programmability of network applications, and (*iii*) fine-grained flow monitoring. These properties can be used to support the implementation of resilience mechanisms and help to minimize the complexity of managing them for network operators. Despite these benefits, new resilience challenges can arise because of the use of SDN, e.g., with respect to the fault tolerance of the control plane; research into addressing these issues is currently a major concern.

This paper presents a survey on the support for network resilience in software-defined networking. Research into this topic has recently intensified, as illustrated in Figure 1, which summarizes the number of research papers addressing resilience aspects in SDN included in this survey, according to their year of publication. We organize the literature surveyed using the resilience taxonomy proposed by Sterbenz *et al.* [7], thus enabling a reliable categorization of the existing research efforts on SDN. Our survey discusses aspects such as existing solutions for resilience challenges, current open issues and research trends in this field. The aim of the survey is to present to the reader a comprehensive and structured view of network resilience in the SDN spectrum, and how resilience aspects are supported in these architectures.

We have observed that solutions related to fault management, infrastructure planning, routing and security applications, network measurement and anomaly detection are frequently used to address resilience challenges in the SDN context. However, we have identified several open issues in this research space, including the protection of the communication channel between network controller and forwarding devices; adequate support for sophisticated QoS solutions to enhance performability; and the need to detect novel malicious attacks targeting network devices.

The remainder of this paper is organized as follows. Section II presents necessary background material on SDN and network resilience. Section III discusses the proposed categorization of SDN efforts, with respect to different resilience disciplines. Section IV shows a summary of the main research topics studied, topics already solved and others under investigation. Finally, Section V presents the concluding remarks.

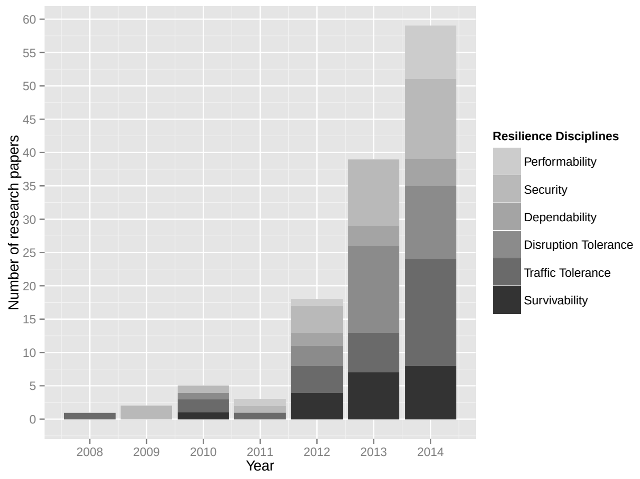


Figure 1: Number of research papers included in this survey, according to their year of publication

2. Background

This section discusses the basic concepts and terminology used in this work. In particular, Software-defined networking and network resilience are contextualized.

2.1. Software-Defined Networking

Software-Defined Networking (SDN) is an architecture for computer networks aimed at decoupling the network control functions (*control plane*) from the forwarding devices (*data plane*) [10]. The *control plane* is responsible for determining the network control logic, such as implementing routing protocols. The aim of the SDN architecture is to simplify the deployment of new control plane functions, such as routing strategies, when compared to traditional networks [12, 13], in which the control and data planes are more tightly coupled and typically operate in an entirely distributed fashion.

The SDN architecture defines three conceptual planes and communication interfaces as depicted in Figure 2:

- The *application plane* is responsible for executing applications that run over the network infrastructure. Generally, these applications perform modifications regarding network aspects, such as network policies and routing behavior, with some degree of human intervention [12]. Examples of network applications deployed in this plane are network visualization, path reservation and network provisioning;

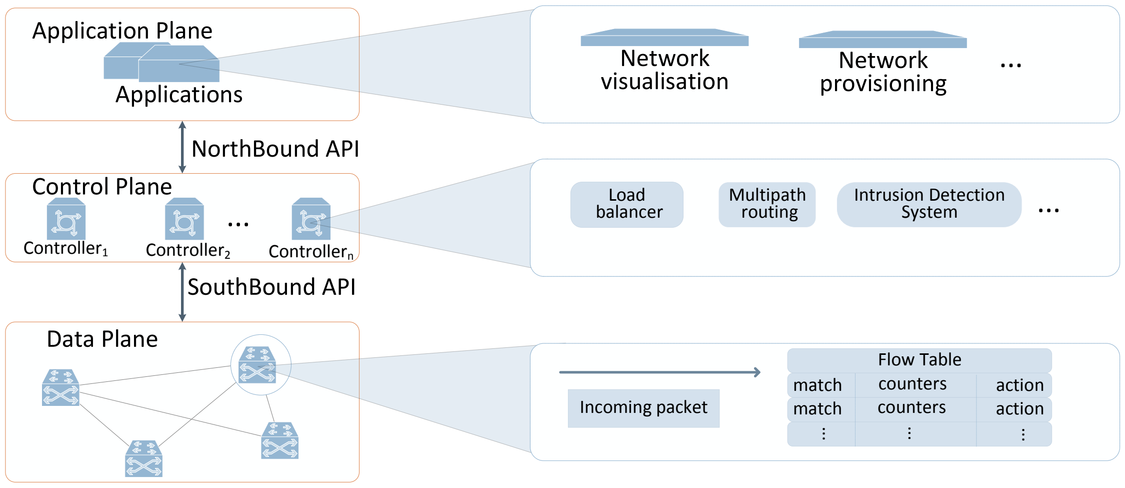


Figure 2: SDN architecture: conceptual planes and communication interfaces

- The *control plane* defines control logic, such as routing schemes. Additionally, the control plane can manage the information collected by switches at the *data plane*, such as flow statistics, to orchestrate the traffic behavior. This plane has a global network view, being able to offer mechanisms for fault diagnosis, make decisions over current traffic distributions and enforce QoS policies. Usually, the *control plane* is physically distributed into *controller* devices, but logically centralized [14];

- The *data plane* includes the devices that are responsible for forwarding data, which are generally referred to as *switches*. An OpenFlow switch offers the notion of programmable flow tables, *i.e.,* tables that define an action for each packet associated with a specified flow. A flow table can be dynamically configured by the *control plane*. When a new packet arrives in a given switch it can be (*i*) dropped; (*ii*) flooded through all output ports; (*iii*) sent to a specific output port; or (*iv*) sent to the network controller [15]. For every flow the switches involved in this communication store statistical information that can be accessed by the *control plane*.

Furthermore, the communication between the different planes occurs through the following interfaces:

- *Northbound API*: Implements the communication interface between the *control plane* and the *application plane*. This API enables the programmability of the network controller by exposing network data abstractions to the *application plane*. Currently, the most used protocol for this communication is REST (REpresentational State Transfer);

- *Southbound API*: Implements the communication interface between the *control plane* and the *data plane*. Through this interface it is possible for the *control plane* to configure switches with forwarding actions according to received notifications of incoming packets from the *data plane* [16]. This is typically standardized and implemented by the OpenFlow protocol [11, 17].

It can be seen that through these interfaces the SDN architecture introduces a great deal of flexibility in flow management, impacting directly in areas such as security, traffic management and performability [18, 19]. Also, SDN has the potential to reduce the cost of network deployment, because simplified data plane switches are relatively inexpensive components, when compared to more complex routers [20]. Furthermore, OpenFlow has proven to be ideal for the development of prototype network applications [21]; based on this success, research in this field has increased [22]. These characteristics generated enthusiasm in both industry and academia. Many surveys covering historical aspects, architecture and challenges related to SDN have been published and further discussions can be found in [23, 24, 25, 12].

2.2. *Network Resilience*

Computer networks support many societally critical functions such as business transactions, military operations and electricity supply. However, a wide range of challenges such as malicious attacks, operational overload and mis-configurations can occur, which could result in failures. Additionally, networked systems can include hardware and software faults that could similarly result in failures, if triggered. Approaches to network resilience aim to protect the network and overcome a degradation in the performance of services when confronted with challenges and faults.

2.2.1. General Principles

There are a number of different ways in which a network can fail to provide a desired level of service, such as a given end-to-end delay or level of availability. Failures can be caused by so-called *challenges*, such as a malicious attack or natural disaster, or a fault. For example, critical failures resulting from natural disasters like Hurricane Katrina [26], in 2005, destroyed much of the network infrastructure on the east coast of the United States. The damage was such that communication links were interrupted and the electrical distribution system was compromised. Furthermore, our increasing dependence on network infrastructures has attracted the attention of cyber criminals. These individuals aim to disrupt the operation of large corporations or even nations through cyber-attacks that are targeted at the communication infrastructure [27]. In this case, a failure is the result of deliberate malicious activities that can compromise a target network service.

The variety of ways that networks can be challenged creates the need for a wide range of resilience mechanisms, which should be deployed across systems, network layers and infrastructures, as necessary. Unfortunately, an ideal resilience system that is capable of protecting the network from any challenge in any environment is difficult to achieve and expensive. A trade-off between the complexity of the mechanisms and their cost exists, and this restriction defines what can be deployed in practice [28]. Consequently, prohibitive costs can make ideal resilience solutions, such as fully fault tolerant systems, infeasible [20].

In general, a number of stages can be implemented to ensure the resilience of networks [7, 29]. Initially a set of *defense* mechanisms should be deployed that address the known challenges a network may face; these might include firewalls or redundant network paths, for example. In some cases, these defense measures will fail, e.g., because new challenges emerge or they are insufficiently provisioned. Consequently, it is important to *detect* challenges and service degradation, and subsequently *diagnose* the root cause of a challenge. Using the outcomes from these stages, mechanisms to *remediate* the challenge can be used to adapt the system operation, in order to ensure continued or graceful degradation of service. For example, suspicious network traffic can be subjected to deep-packet inspection (a form of detection and diagnosis), while malicious traffic can be blocked (a remedial action). Finally, when a challenge has abated, the network should *recover* to normal operation by disengaging remediation mechanisms, for example.

2.2.2. Resilience Disciplines

In this section, we discuss the resilience disciplines that are related to networked systems. According to Sterbenz *et al.* [7], a number of existing disciplines address aspects of network resilience, which can be placed into two categories: *(i)* disciplines that provide mechanisms to address different classes of challenges and faults; and *(ii)* those specifying measurable properties that indicate the resilience of a network. We summarize these disciplines:

- *Survivability*: is a superset of *fault tolerance*, which addresses small numbers of random uncorrelated faults, by considering *numerous correlated failures* that could be caused by challenges such as malicious attacks and large-scale natural disasters. While *redundancy* is often considered sufficient for fault tolerance, ensuring the survivability of networks requires approaches to *diversity* to be implemented;

- *Traffic tolerance*: enables the network to tolerate unusual traffic load without interrupting its operation. It deals with legitimate traffic management, *e.g., flash crowds*, largely via traffic engineering mechanisms. However, DDoS attacks can behave similarly to legitimate traffic, thus creating the need to identify and treat this type of traffic in a specific manner;

- *Disruption tolerance*: enables the network to tolerate weak and episodic connectivity that is typical of mobile and wireless networks. Approaches to disruption tolerance can include error correction schemes, multi-path routing and in extreme cases (e.g., for mobile ad hoc networks) store-carry-forward schemes. Often in this context, there are *energy* trade-offs that need to be addressed;

- *Dependability*: quantifies the reliance that can be placed on the service delivered by a system. Consequently, this definition encompasses concepts related to *availability* – an indicator of whether a service will be present when requested – and *reliabil-*

Table 1: SDN research efforts on Fault Tolerance and Survivability

SDN Planes	Fault Tolerance	Survivability
Application plane	Reitblatt *et al.* [30] Heller *et al.* [32] Canini *et al.* [33] Scott *et al.* [34]	Chandrasekaran *et al.* [31]
Control plane	Jain *et al.* [35] Botelho *et al.* [37] Jian *et al.* [39] Fonseca *et al.* [41] Tootoonchina *et al.* [42] Zhang *et al.* [44]	Williams *et al.* [36] Liu *et al.* [38] Kempf *et al.* [40] Chandrasekaran *et al.* [31] Muller *et al.* [43]
Data plane	Jain *et al.* [35] Botelho *et al.* [37]	Liu *et al.* [38] Kempf *et al.* [40]

ity – a measure of continued operation for a specified period of time. Also, dependability relates to measures of *safety*, *integrity* and *maintainability*;

- *Security*: is a property and set of measures that relate to unauthorized access to a networked system, and includes notions of self-protection. It includes concepts such as *confidentiality*, *nonrepudiality* and *AAA* (*auditability*, *authorizability*, *authenticity*). Additionally, security properties intersect with the dependability concepts of *availability* and *integrity*, with subtly different semantics – for security, these properties typically relate to *information assets*, rather than services.
- *Performability*: metrics describe the network's ability to deliver the performance required by its users; these requirements are normally expressed in Quality of Service (QoS) agreements. Typical performability metrics include delay, jitter, throughput and goodput, for example.

3. State-of-art in SDN Resilience

This section discusses the main research efforts that have addressed resilience aspects in the context of SDN. The methodology employed to produce this survey is as follows. First, a total of 142 research papers and technical reports on SDN and resilience were gathered, based on criteria such as date and relevance. Second, the publications were categorized into the different resilience disciplines that are proposed by Sterbenz *et al.* [7] (see Sec. 2). Third, these works were arranged into one or more SDN planes (Figure 2), in order to provide a high-level view of the intended resilience support over the different planes. In the following, for each resilience discipline, we discuss the main challenges and the solutions used to protect the network in SDN environments. Despite our efforts to accurately categorize the publications within the several resilience disciplines, we acknowledge that in some cases the same publication could be classified differently according to the assessment of the reader.

3.1. Fault Tolerance and Survivability

A natural concern about the SDN architecture is the survivability of the network controller, forwarding devices in the data plane and applications that monitor and change the network operation. To achieve the protection of these components, survivability encompasses the treatment of *fault tolerance* in the face of uncorrelated failures that are caused by faults, and *multiple correlated failures* that are caused by natural disasters and attacks, for example. Table 1 presents the major research efforts addressing survivability in SDN. In the following we discuss these works in detail.

Fault tolerance: is the ability of a system to provide continued operation or degrade gracefully in the presence of faults. The classical approach to fault tolerance is to introduce redundancy into the system, *e.g.,* through the use of replicas to protect critical components in the network [45]. In the SDN context, redundancy can be used to (*i*) protect the network controller from service failures, (*ii*) to protect the forwarding devices and communication links from link disruption, and (*iii*) to protect network applications from misconfiguration [44]. Despite the benefits of fault tolerance, a fully fault tolerant system brings complex issues that are related to the management of replicated components and equipment costs. SDN can help to address these issues because its architecture can accommodate the control of complex network functions. For example, the *control plane* can be used to orchestrate the behavior of active and redundant devices, eliminating the need for new devices to perform such tasks [41].

Jain *et al.* [35] relate experience with the use of fault tolerant approaches to address network outages and failures with B_4, a private WAN that connects the Google's data center. They used OpenFlow to manage individual *switches* that implement several fault tolerance tasks, such as multipath routing regarding flow priority and dynamic reallocation of bandwidth when link failures occur. The redundancy is achieved using software replicas to protect individual and control processes in the *control plane*. These replicas are placed on different physical servers and, in case of network faults, a consensus algorithm can be used to elect the replica that will assume the demand of the network. All the configurations and communications used to protect the network are assisted by an SDN-based architecture.

Botelho *et al.* [37] treats the consistency between a network controller and their redundant backups, focusing on performance aspects. The main conclusion of the authors is that strict consistency and fault tolerant systems can operate with acceptable performance. However, issues related to latency were shown to limit the responsiveness of the system. Jian *et al.* [39] confirm this observation and show that the performance of the network controller can be critical to link failure detection, even in fault tolerant systems.

We understand that a large number of applications that are related to fault tolerance running in the *control plane* can reduce its capacity to process network traffic. A solution to tackle this limitation is moving the majority of these applications to the *application plane*, which is more scalable and suitable for running these tasks mainly because this plane is designed exclusively to support the execution of general SDN applications. Using the programmability offered by the SDN architecture, Reitblatt *et al.* [30] propose FatTire, a language for writing fault-tolerant systems. The idea behind this approach is to offer an abstraction for network paths, such that the forwarding behavior of network packets can be orchestrated though regular expressions that are converted into primitive switch rules. This is an *application plane* solution, and can be updated according to the need of new resilience requirements. Furthermore, Fonseca *et al.* [41] investigate the redundancy of network controllers. Related to this is the work by Tootoonchian *et al.* [42], which deals with the distribution of controller state over the network, in order to offer a logically centralized network controller.

A recent concern involves the programmability offered by SDN and the challenges related to software debugging. In our understanding these issues also bear some relationship to Fault Tolerance. Heller *et al.* [32] discuss the overall problem space regarding troubleshooting in SDN but the authors do not propose any system or framework. Scott *et al.* [34] also deal with this aspect and in addition propose a troubleshooting system (called STS), which aims to alleviate the time-consuming nature of debugging by eliminating events that are not causally related to the source of a failure. Further, Canini *et al.* [33] are also concerned with troubleshooting issues, and apply model-checking techniques to represent the state space of the network. This strategy is useful to detect design flaws, a frequent type of software bug.

Survivability: multiple, uncorrelated failures can be unpredictable and difficult to diagnose, and in this case redundancy may not be enough. A typical strategy to handle this kind of failures is *diversity*, *i.e.,* to use a set of distinct resilience schemes to determine and treat the source of failure. This increases the success of detection/mitigation schemes because a wide-range of network challenges can be addressed.

For example, *diversity* can be typically employed to withstand catastrophic faults, such as natural disasters. Muller *et al.* [43] highlight that even if the *control* and *data planes* are compromised, different placement strategies of the network controller, path diversity and distinct recovery mechanisms can be used to ensure that the network can still function. Note that similar techniques can be used not only to improve aspects related to Survivability but also to Disruption Tolerance.

Chandrasekaran *et al.* [31] discuss how to handle challenges at the *application* and *control planes*. Their focus is to treat Byzantine failures, fail-stop crashes and other uncorrelated failures. They propose two abstractions for improving the network controller availability and diagnose network application faults: a module used for fault isolation and another to deal with network transactions. The joint operation of these components enables the orchestration of high-level applications, such as fault alerts and the specification of policies to enforce actions when failures occur. The proposed framework still enables the distribution of network events to different SDN applications, in order to tolerate multiple, uncorrelated failures.

SDN is a suitable environment to investigate *diversity* of automatic recovery mechanisms. For example, mechanisms used to plan for failure can be placed in distinct network controllers [36], and *control plane* applications can be used to ensure path reservation in the *data plane* [38]. Related to these ideas, Kempf *et*

Table 2: SDN research efforts on Dependability

SDN Planes	**Reliability**	**Availability**
Application plane	N/A	N/A
Control plane	Veisllari *et al.* [46] Deguo *et al.* [48] Ros *et al.*[50] Santos *et al.* [52] Dixit *et al.* [53]	Heller *et al.* [47] Hock *et al.* [49] Beheshti *et al.* [51]
Data plane	N/A	N/A

al. [40] highlight that fault management in SDN cannot be left to be fully implemented in the *control plane*, and instead delegates these tasks to the OpenFlow switches. They advocate that some control functions, such as connectivity monitoring, can be placed in the *data plane*. Consequently, *diversity* can be attained if different resilience mechanisms are implemented in these switches.

3.2. Dependability

Even if a system fails, it can be considered *dependable* if failures occur with an expected probability. Thus, dependability quantifies the reliance that can be placed on the service delivered by a system. In this sense, dependability encompasses concepts involving *reliability* and *maintainability*. It is also sometimes related to *safety*, *availability* and *integrity*[1]. Table 2 categorizes the major research efforts related to dependability in SDN. These are discussed in the following. Note that the research papers discussed in this section are all related to the *control plane*.

Reliability: Deguo *et al.* [48] state that the *data plane* cannot detect failures in the *control plane* and the number of control messages lost when a failure occurs can compromise the forwarding behavior of the network. A solution broadly accepted to deal with this challenge is to delegate some network control logic to the forwarding devices, such as rule cloning and multipath support [54]. A single point of failure is another reason that contributes to the decentralization of the *control plane* [52, 53].

Ros *et al.* [50] investigate the controller placement problem with respect to network reliability. The authors propose a metric called *k-terminal-reliability*, which is the probability of having at least one operational path in the network. The authors can optimize the solution for reliability by formulating the controller placement problem as a graph optimization problem, and inserting the *k-terminal-reliability* as a restriction when searching for the optimal solution.

The reliability of the *data plane* can be related to switches and link state. Metrics such as communication delay, throughput and latency are frequently used as indicators of reliability, as these represent expected values that can or cannot be satisfied at any time.

Availability: this is the probability of a system to be in a correct state in a given instant. New metrics that are related to this issue are not currently the focus intense study, however the controller placement problem is a research question that is related to availability. This problem investigates (*i*) how many controllers are needed to control a given network and (*ii*) what is the best place to position the controller regarding metrics of availability [47]. The most commonly used metrics measure the average-case latency, worst-case latency and the maximization of number of nodes with latency bound. Hock *et al.* [49] propose more elaborate metrics, using latency during controller failures, load imbalance and inter-controller latency. Beheshti *et al.* [51] propose metrics such as the number of protected switches (switches that can use backup links for the control traffic) and the number of unprotected switches.

3.3. Security

Historically the deployment of complex security functions (*e.g.,* intrusion detection systems and firewalls) has required the installation of dedicated security appliances. For some organizations the costs and management issues related to these deployments can be prohibitive. Additionally, in traditional networks the lack of a centralized control of these security functions can further complicate their deployment [61]. In contrast, SDN enables the implementation of applications that have the ability to support similar security functions in a much more flexible manner, and it offers a suitable place for the implementation of more accurate, reliable

[1]Safety and maintainability are rather general properties and to the best of our knowledge, there are no research efforts in SDN solely concentrated on these fields. Integrity is related to security and will be discussed in the next section.

Table 3: SDN research efforts on Security

SDN Planes	Confidentiality	Availability	Integrity	Nonrepudiality
Application plane	N/A	Chen *et al.* [55] Wang *et al.* [56] Seeber *et al.* [57] Tasch *et al.* [58]	N/A	N/A
Control plane	Benton *et al.* [59] Schehlmann *et al.* [63] Kreutz *et al.* [64] Porras *et al.* [66] Anwer *et al.* [69] Fayazbakhsh *et al.* [71] Naous *et al.* [73] Silva *et al.* [75] Ballard *et al.* [77] Schlesinger *et al.* [79]	Li *et al.* [60] Zaalouk *et al.* [61] Schehlmann *et al.* [63] Mazieres *et al.* [67] Chen *et al.* [55] Wang *et al.* [56] Seeber *et al.* [57]	Zaalouk *et al.* [61] Schehlmann *et al.* [63] Ye *et al.* [65] Collings *et al.* [68] Hu *et al.* [70] Xing *et al.* [72] Kampanakis *et al.* [74] Jafarian *et al.* [76] Smeliansky *et al.* [78] Abaid *et al.* [80] Benton *et al.* [59] Shin *et al.* [81] Kumar *et al.* [82] Li *et al.* [83] Qazi *et al.* [84] Son *et al.* [85]	Nayak *et al.* [62]
Data plane	Kreutz *et al.* [64] Shin *et al.* [81] Naous *et al.* [73] Qazi *et al.* [84]	Chen *et al.* [55] Wang *et al.* [56] Seeber *et al.* [57]	Hu *et al.* [70] Xing *et al.* [72] Smeliansky *et al.* [78]	N/A

and efficient security solutions. Table 3 presents the major research efforts addressing security in SDN. In the following we discuss these in detail.

Availability: Schehlmann *et al.* [63] state that the availability of the network controller can compromise the correct operation of network functions. To address the availability aspects of the network controller with respect to security, Li *et al.* [60] propose a novel SDN architecture based on BFT (Byzantine Fault Tolerant) [67] mechanisms to withstand malicious attacks on the *control plane*. The authors state that a distributed control plane can assist in protecting the network mainly because a single point of attack is avoided. Also, the authors highlight that the additional protection strategies, such as the use of BFT, can ensure the correct operation of critical network functions (*e.g.,* flow tables updates).

With a distributed *control plane*, it is natural to consider the distributed placement of security applications. However, this can increase the communication delay in security traversal routing, *i.e.,* the traversal of a given flow through secure devices to enforce security inspection. Chen *et al.* [55] address the security traversal problem with shortest path solutions, including the ability to dynamically select the optimal security traversal path. Cloud environments, for example, can benefit from solutions of this type, since security issues related to communication is critical [57, 56].

It is possible that not only the network controller but also the network applications will be target of attacks. In line with this, the availability of security applications is addressed by Tasch *et al.* [58]. The authors state that even consolidated applications such as RESONANCE [62] can have security problems, such as identity spoofing and repudiation when TLS is not available to protect the control communication.

Integrity: several works suggest that firewalls are more concerned with (the integrity aspects of) security. Despite all functions that these systems can perform, their main goal is to maintain the integrity of the communication link and network devices, *i.e.,* protect the network from illegitimate attempts to gain access to its services [68]. Firewalls are an example of network application that can be fully assisted by SDN because: (*i*) the need of additional middleboxes to enforce policies in the network is reduced because this functionality can be placed in the SDN *control plane*; (*ii*) the *con-*

trol plane has a comprehensive view of the network; and (*iii*) the management of heterogeneous devices is abstracted by the *control plane*. Also, Qazi *et al.* [84] present a policy security monitoring layer for efficient middlebox-specific *traffic steering*. Another work that goes in this direction is proposed by Son *et al.* [85].

Resolution of firewall policy violations and conflicts are addressed by Hu *et al.* [70]. The authors propose the *FlowGuard* framework to monitor and check network flows in order to detect firewall policy changes when the network state is updated. However, they identified the following challenges with respect to the implementation of firewalls in SDN: (*i*) as the network state is dynamically changed, new configurations are frequent and simple packet-in monitoring will not be effective for detecting flow policy violations; (*ii*) the *set-field* actions of the OpenFlow protocol enable the dynamic change of packet headers, creating an opportunity for malicious users to attack the network; (*iii*) as OpenFlow enables the use of *wildcards* to match only partial header fields of packets in these security policies, the elimination of a flow policy can affect benign traffic; (*iv*) the *data plane* is unable to monitor flow status, depending heavily on the *control plane*, thus it is challenging to perform stateful packet inspection.

Intrusion Detection Systems, such as Snort[2], have been used to protect the integrity of the network in traditional environments. SnortFlow [72] is an extension to Snort with SDN capabilities that enables the detection of intrusions and malicious activities in cloud environments. Detection mechanisms are traditionally based on Machine Learning [86], Signatures [87] and Entropy [88]. Shin *et al.* [81] group several security needs and deliver a complete framework for security implementation, sharing and composition of detection modules and mitigation in a SDN. This effort enables the evaluation of optimal mechanisms to protect the network, being able to detect and manage malicious activities. Another example of IDS with these responsibilities is presented by Kumar *et al.* [82]. Solutions based on packet classification using SDN are presented by Smeliansky *et al.* [78] and the detection of malware is discussed in the work of Abaid *et al.* [80]. Further, the PROTOGENI framework is presented by Li *et al.* [83] and serves as a tool for creating and evaluating different types of network attacks.

However, other solutions to protect the network from intrusions and malicious attacks are possible. Kampanakis *et al.* [74] advocate the use of *moving target defense* (MTD) in order to protect network services when malicious traffic try to compromise their integrity. The authors conclude that SDN makes the implementation of MTD techniques more practical, customizable and easier to deploy. Jafarian *et al.* [76] present a study related to this problem and propose a technique named *random host mutation*, *i.e.,* the path between source and target is randomized to avoid the effects of malicious traffic.

Security threats can also compromise the integrity of configuration messages sent by the *control plane* to the *data plane* by modifying or introducing errors in their content [65]. A simple solution for this challenge is to use encryption protocols, such as TLS. Benton *et al.* [59] state that the biggest concern related to security in SDN is the protection of the communication between the *control plane* and the *data plane*. The authors point out that the TLS protocol in the OpenFlow specification may sometimes be used incorrectly, because in order to put TLS in practice, the network operator must achieve security certificates for each of the devices involved in the communication and manually configure each of them. In contrast, to use plain text communication without any encryption, the network operator only needs to configure the network controller address in the *data plane*. Thus, the difficulty in deploying TLS can discourage its use.

Confidentiality: related to the issues discussed above, another challenge pointed out in the work of Benton *et al.* [59], and uniquely related to SDN, is called the *listener mode*. This existing functionality in many *switches* allows the establishment of a data connection in a pre-configured port without authentication. Although it is used primarily for debugging reasons, this connection can be used to modify rules in switches and discover information about the network. If TLS is not used correctly, an attacker can intercept packets and perform network discovery based on communications observed between the *control plane* and the *data plane*. Additionally, a switch can change rules without notifying the *control plane*. Clearly, the issues related to TLS deployment represent a major challenge to security in the SDN context. A possible solution to such problems is the use of *middleboxes* to perform the communication between the *data plane* and the *control plane*. Sherwood *et al.* [89] propose a virtualization platform called FlowVisor that intermediates the communication between the network controller and the switches. Other examples of middleboxes used to enhance security are presented by Anwer *et al.* [69] and Fayazbakhsh *et al.* [71]. In particular, Kreutz *et al.* [64] present the vectors of the most common threats in SDN, such as DDoS attacks, and comment on the TLS chal-

[2]http://www.snort.org

lenges above.

Silva *et al.* [75] describe the use of an anti-eavesdropping technique based on multipath routing for SDN-based SCADA systems used in electrical smart grids. Centered on the confidentiality of the OpenFlow protocol, Kloti *et al.* [90] consider attacks that exploit flow aggregation to discover information about the network state and topology, which would not be visible otherwise. The authors use two modeling techniques, namely Microsoft's STRIDE and *attack trees*, to identify and explore threats to SDNs. Schlesinger *et al.* [79] analyze the problem of dividing the network in slices, thereby allowing traffic isolation, which can guarantee confidentiality in the communication.

Frequently, the mitigation mechanisms used should install firewall rules in the *data plane*. Porras *et al.* [66] propose a software extension to the NOX controller called FortNOX, which is intended to avoid conflicting rules to be installed in the data plane. FortNOX is also used as a security policy management tool. Also in the context of security policies, Naous *et al.* [73] propose the protocol *ident++* that allows the search for information or rules placed on hosts. Ballard *et al.* [77] propose the OPENSAFE and ALARMS languages to simplify the specification of security policies using the OpenFlow protocol. Jafarian *et al.* [76] present an elegant solution for the mitigation of malicious activities and protection of IP addresses against spoofing by enabling the network to randomly modify the IP addresses used.

Nonrepudiality and **AAA:** very few works have addressed exclusively these aspects. Some firewalls deal with these issues, but it is possible that SDN can ensure these properties using TLS in its communication. Further reading can be found in [91], which presents a survey discussing interesting aspects of security in SDN.

3.4. Performability

In recent years, the performability of the *control plane* has been a main concern. Despite the benefits of a centralized control logic in the SDN/OpenFlow architecture, Curtis *et al.* [54] point out that the current OpenFlow specification does not meet the demands of high performance networks. This is mainly due to the following reasons: (*i*) there is a high dependence on a central logic and on the global view of the network; (*ii*) it is possible that path latency can slow down the communication between the *control* and *data planes* during flow setup; and (*iii*) there is an excessive dependence on the *control plane*, demanding considerable resources to maintain this feature. To address these challenges, the authors present DevoFlow, a modification of OpenFlow that reduces the amount of communication between the *control* and *data planes*, thereby reducing its overhead. DevoFlow achieves this goal by handling flows in the *data plane*. Additionally, Veisllari *et al.* [46] investigate the performability of the *control plane* with respect to scalability. Scalability is related to performance when we consider metrics such as delay, latency and throughput. The authors conclude that the current Internet flow definitions have high requirements on the processing rate of the SDN controller.

Table 4: SDN research efforts on Performability

SDN Planes	**QoS**
Application plane	Wei *et al.* [93]
Control plane	Curtis *et al.* [54]
	Veisllari *et al.* [46]
	Egilmez *et al.* [94]
	Akella *et al.* [95]
	Huang *et al.* [96]
	Xiong *et al.* [97]
	Wang *et al.* [98]
	Machado *et al.* [99]
Data plane	Zhang *et al.* [92]
	Egilmez *et al.* [94]
	Wang *et al.* [98]
	Machado *et al.* [99]

One of these requirements is the consistent population of flow tables regarding performance in traffic management. Zhang *et al.* [92] address the problem of redundant rules that can appear after successive insertions of flow rules. The authors discuss a compression method based on the combination of similar entries using techniques such as Huffman coding. Internet applications over the network have performance requirements with respect to the communication with the *control plane*. Efforts focusing on the performability of the Northbound API are presented by Wei *et al.* [93], who propose the use of caches to speed up the service of this interface. Table 4 categorizes the major research efforts related to performability in SDN, and next we discuss these efforts with respect to QoS (Quality of Service).

QoS: Sonkoly *et al.* [100] state that SDN developed slowly with respect to QoS support and that the current result is "even worse" than expected. In part this occurs due to several limitations of the devices present in the *data plane*, which is the same reason that makes the implementation of QoS difficult in traditional networks. The current version of the OpenFlow protocol supports only simple mechanisms to address QoS queues. Some counters provided by the *data plane*, *e.g.,* the number of packets received or flow duration, can be used to provide QoS on existing packets in the network, but are limited and inflexible. Additionally, the manipulation of existing switch queues is difficult because there is a lack of advanced interfaces to access their information. One solution is to use protocols such as *OF-config*, which offers a management interface with NETCONF features.

Egilmez *et al.* [94] propose an architecture for QoS called OpenQoS, which is used for grouping data streams and multimedia flows in sets of traffic classes allowing differential treatment for each type of class. Note that QoS-based approaches are inherently dependent on the queue concept and prioritization, which can be assisted by the programmability of the *control plane*. Egilmez *et al.* [101] address QoS for video streaming and deal with routing issues, such as packet loss. The authors consider large networks controlled by clusters of network controllers, and all the controllers decide jointly the best policy to enforce QoS in the network.

Other examples of QoS support in SDN are:

- *Cloud:* Akella *et al.* [95] address the problem of guaranteeing QoS requirements for cloud users, such as low delay. The most used QoS solution for multimedia applications is to differentiate the way in which different types of packets are forwarded. Also, the authors study bandwidth allocation for QoS using queuing techniques. The performance metrics used are response time and number of hops. Another effort towards QoS in clouds is presented by Huang *et al.* [96], which provide a theoretical and experimental analysis for end-to-end QoS provisioning;

- *Data Stores:* Xiong *et al.* [97] advocate the use of SDN to manage the performance of distributed queries in data stores. The authors discuss how to control the priority of network traffic or make bandwidth reservations using queuing theory;

- *Servers:* Wang *et al.* [98] propose an autonomic QoS management mechanism for SDN by extending the OpenFlow and OF-Config protocols. The authors introduce a packet context-aware QoS model (PCaQoS) to provide self-configuration;

- *Policy Based Management:* Machado *et al.* [99] propose a policy refinement approach for QoS management. The authors propose a method capable of identifying QoS requirements and use PBM (Policy Based Management) mechanisms and natural language to translate these requirements into primitive switch configurations.

3.5. Traffic Tolerance

Traffic tolerance enables the network to withstand unusual traffic profiles without compromising its expected behavior [7]. This discipline deals with traffic related questions, such as malicious attacks and legitimate traffic management. In the following, the most important studies in this topic are discussed. Table 5 summarizes the work presented in this section.

3.5.1. Legitimate traffic

Although protocols such as *SCTCP* and *MP-TCP* delegate traffic resilience to the network core, it is a somewhat limited approach because global network protection is difficult to achieve due to the lack of flexibility of traditional architectures. An alternative is to transfer this responsibility to the edge of the network, *i.e.,* to improve routing algorithms in order to make traffic management more resilient. Despite this seems more appealing and flexible, even a simple change to routing paths can lead to inappropriate solutions with respect to network performance and communication delay. Traditional, non-SDN network architectures do not offer support for advanced routing solutions, however, within an SDN architecture, software abstractions can be created to improve legitimate traffic resilience and complex routing schemes can be easily deployed.

A broad discussion on routing in SDN can be seen in the work of Rothenberg *et al.* [102]. According to the authors, the OpenFlow protocol represents a real opportunity for the deployment and evaluation of routing strategies. Typically, these solutions are implemented in the *control plane*, *i.e.,* the SDN controller is extended to deal with problems such as link failure, communication delay and load distribution. Agarwal *et al.* [5] deal with traffic engineering issues, but in the context of delay and packet losses. The work supports adaptive routing based on performance metrics, such as delay. The authors rely on the global network view to create a graph that represents all links available. After this, a mathematical formulation of the routing problem considering these metrics is produced, and a Fully Polynomial Time Approximation Scheme (FPTAS) is used to find the best solution for the problem. One advantage of this work is that it does not require protocol changes and can be used in scenarios where SDN is not completely deployed. Akyildiz *et al.* [105] deal with similar traffic issues. In general, the programmability offered by SDN encourages the use of classical algorithms, such as graph-based algorithms, to solve routing problems.

INFLEX is a framework that extends the network controller to create multiple routing planes that can be

Table 5: SDN research efforts on Traffic Tolerance

SDN Planes	**Legitimate Traffic**	**DDoS Attacks**
Application plane	N/A	N/A
Control plane	Taveira *et al.* [9] Rothenberg *et al.* [102] Agarwal *et al.* [5] Akyildiz *et al.* [105] Taveira *et al.* [9] Ramos *et al.* [107] Benson *et al.* [109] Raza *et al.* [111] Venmani *et al.* [113] Qazi *et al.*[84] Shimim *et al.* [116] Silva *et al.* [118] Laga *et al.* [119] Xiaogang *et al.* [120] Rodrigues *et al.* [121] Bennesby *et al.* [122]	Braga *et al.* [86] Shin *et al.*[103] Michale *et al.* [104] Giotis *et al.*[106] Benton *et al.*[59] Alcorn *et al.* [108] Belyaev *et al.* [110] Wang *et al.* [112] Passito *et al.* [114] Li *et al.* [115] Shtern *et al.* [117]
Data plane	Taveira *et al.* [9] Ramos *et al.* [107] Bensom *et al.* [109] Venmani *et al.* [113] Sgambelluri *et al.* [124]	Braga *et al.* [86] Krishnan *et al.* [123] Michale *et al.* [104] Benton *et al.* [59]

switched when a challenge is observed, for example, a link failure [9]. The core of the architecture lies on the Differentiated Services (DS) field of the IP protocol to create the notion of a routing plane for every packet in the network. Every host sets the DS field properly to guarantee that its packets will follow the same route. Additionally, hosts can request a new plane when a fault is observed (*e.g.,* if no response is received after a few seconds). Other authors, *e.g.,* Ramos *et al.* [107], similarly exploit the programmability of SDN to support alternative communication paths.

Benson *et al.* [109] address new traffic engineering strategies in data center networks to efficiently accommodate various types of traffic. Although the research area of traffic engineering is not focused exclusively on resilience, some of its concepts can be used to support resilience objectives, *e.g.,* prioritization of traffic profiles. In order to handle conflicting constraints, such as conflicting QoS requirements, the MeasuRouting framework presented by Raza *et al.* [111] can enforce QoS using traffic monitors that guarantee the demand of the network traffic.

With respect to link congestion, Venmani *et al.* [113] use OpenFlow to provide improvements to flow routing in backbone networks. These improvements include the use of the network controller to generate high-level policies to notify link failures. In this case, the controller can run a routine to recover the network back to its normal operation (*e.g.,* calculation of new paths and link failure detection). Sgambelluri *et al.* [124] remove this responsibility from the network controller and pass it to the *data plane*. The idea is to install backup rules with low priority, thus in case an error occurs the *data plane* itself can change the path used for communication. In scenarios where the network controller is usually overloaded, such solutions serve as an efficient alternative to avoid communication latency. To enforce capabilities related to policy management in the performance context, Qazi *et al.*[84] propose the use of middleboxes orchestrated by SIMPLE, a policy enforcement layer in the *control plane*. Additionally, the specification of packet-forwarding policies can be assisted by high-level languages, such as those proposed by Voellmy *et al.* [125] and Foster *et al.* [126].

Given that one of the main benefits of SDN is to allow flexibility in routing, several solutions for routing challenges have been investigated. Shimim *et al.* [116] and Silva *et al.* [118] deal primarily with multicast routing. They suggest the use of applications on the *control plane* to orchestrate flow behavior in order to provide multicast routing. Video routing is addressed by Laga *et al.* [119] and issues related to MPLS are discussed in the work of Xiaogang *et al.* [120]. Rodrigues *et al.* [121] discuss the optimization of network utilization across layers using bandwidth virtualization. The authors show that inefficiencies exist in links of data center WANs, but SDN can assist in addressing these restrictions. Bennesby *et al.* [122] address the problem of inter-domain routing with SDN support. The authors present performance and scalability results to demonstrate that SDN can help to mitigate the limitations of rigid BGP deployments, such as the difficulty in supporting architectural innovation. Additionally, an overview of transport networks in the context of SDN is provided by Alvizu *et al.* [127].

3.5.2. DDoS Attacks

According to Mehdi *et al.* [87], the deployment of an anomaly detection system in the traditional network core is difficult mainly due to the low detection rate that these systems can provide with limited network information. In SDN, however, the *control plane* has a comprehensive view of the network, which facilitates the implementation of detection mechanisms. Mehdi *et al.* [87] presents an overview of attack detection possibilities using SDN. The authors discuss four well-known algorithms: TRW-CB, MaxEnt, RateLimit and NETAD. Their study suggests that SDN is a platform suitable for the mitigation of DDoS attacks, mainly because of the use of standard protocols, services and interfaces, thus facilitating the deployment of new solutions.

Several strategies can be used to detect and mitigate DDoS attacks. Belyaev *et al.* [110] state that existing solutions to mitigate DDoS attacks can be classified as *active* (*e.g.,* based on the use of machine learning for detection) or *survival* (*e.g.,* trying to tolerate a DDoS attack). The latter is concerned with solutions based on load balancing. As pure load balancing is not effective during a DDoS attack, an iterative splitting of traffic paths where the network is overloaded may be necessary. This can increase the chances of tolerating a DDoS attack. For example, Wang *et al.* [112] present a mechanism to protect the control path against DDoS attacks by scaling the control channel capacity. This allows the network to handle a large number of flows, and makes the *control plane* more resilient.

Braga *et al.* [86] investigate the mitigation of DDoS attacks using Self-Organizing-Maps (SOM), a machine learning algorithm already used in traditional networks but with limited effects due to the restrictions of that architecture. Frequently, well-known solutions for traditional networks are implemented in the SDN context, as in the work of Ramadas *et al.* [157]. Further, Shin *et al.* [103] propose the insertion of triggers to control and change flow dynamics in the *data plane*. This can be used to expose the malicious flows for the detection and mitigation of DDoS attacks. For example, Krishnan *et al.* [123] present several detection methods and discuss which methods can be implemented in the *data plane*. Michale *et al.* [104] propose packet classification based on techniques such as prefix match and flow caches, to avoid the repeated classification of flows. Alcorn *et al.* [108] present a framework to model and simulate DoS and DDoS attacks in SDN/OpenFlow networks.

More recently, the use of information theory for packet classification has been investigated by Giotis *et al.* [106], who use entropy analysis for monitoring deviations in network behavior. Additionally, man-in-the-middle attacks are discussed by Benton *et al.* [59], who indicate that the adoption of TLS as a secure communication channel can present vulnerabilities. Passito *et al.* [114] present a solution that allows SDN domains to cooperate in the mitigation of DDoS attacks. Li *et al.* [115] use traffic engineering techniques to reduce the impact of a DDoS attack, and Shtern *et al.* [117] address the mitigation of Low and Slow Distributed Denial of Service (LSDDoS), a variant of traditional DDoS that can compromise network applications by simulating their behavior.

3.6. Disruption Tolerance

The distributed nature of network devices contributes to the unpredictability of delay in their communication. In addition, natural disasters, *e.g.,* hurricanes and earthquakes, often compromise links, preventing communication in the affected region and causing communication delays in other parts. Power outages and intermittent connection can also leave part of the network without operation. Issues related to these challenges, summarized in Table 6, comprise the disruption tolerance discipline. These are discussed in the following.

Connectivity: a threat constantly faced by networks is the disruption of the connectivity between its components. Link disruptions due to natural disasters or human interaction require the rapid establishment of alternative routes to restore in part the services affected. These new routes can generate bottlenecks in network devices that face an excessive demand for data processing. This might result in the delivery of degraded services to network users. Menth *et al.* [158] deal with link disruption issues and rerouting of packets in traditional networks, and illustrate how critical these questions are to the resilience of networks in general.

Table 6: SDN research efforts on Disruption Tolerance

SDN Planes	Connectivity	Mobility	Delay	Energy
Application plane	N/A	Pupatwibul *et al.* [128] Namal *et al.* [129] Yuhong *et al.* [130] SchulzZander *et al.* [131]	N/A	N/A
Control plane	Yeganeh *et al.* [132] Heller *et al.* [47] Zhang *et al.* [92] Beheshti *et al.* [51] Hock *et al.* [49] Nguyen *et al.* [141] Stephens *et al.* [144] Borokhovich *et al.* [147]	Pupatwibul *et al.* [128] Sabbagh *et al.* [134] Namal *et al.* [129] Sahri *et al.* [137] Zhipu *et al.* [139] Guolin *et al.* [142] Jeon *et al.* [145] Yuhong *et al.* [130] Shengli *et al.* [148] Ding *et al.* [149] SchulzZander *et al.* [131] Lara *et al.* [150] Jagadeesan *et al.* [151] Sperotto *et al.* [152] Giust *et al.* [153] Kyoungjae *et al.* [154] Hyunsik *et al.* [155]	Yeganeh *et al.* [132] Heller *et al.* [47] Hock *et al.* [49] Phemius *et al.* [138] Cai *et al.* [140] Rotsos *et al.* [143] Mahmoodi *et al.* [146]	Heller *et al.* [133] Wang *et al.* [135] Pfeiffenberger *et al.* [136]
Data plane	N/A	Guolin *et al.* [142] Shengli *et al.* [148] Lara *et al.* [150] SeongMun *et al.* [156]	N/A	Heller *et al.* [133]

Despite the fact that SDN is appropriate for innovation of the network control tasks, the problems faced by traditional IP networks persist even when we consider the full potential of this new approach [132]. A centralized *control plane* with an overview of the network topology has instigated studies that focus on new solutions for legacy problems in distributed systems, such as link failure. However, as in SDN the responsibility for defining the communication paths between network components lies with the network controller, a new category of problems arises that are related to the availability of this component.

The controller is responsible for defining how packets will be forwarded at the *data plane*. Thus, the protection of the controller is the first critical point to address. Heller *et al.* [47] and Zhang *et al.* [92] deal with the *controller placement problem*. Their objective is to ensure there is connectivity between the controller and the switches. Beheshti *et al.* [51] also tackle this problem, but additionally deal with traffic control issues, such as link disruption and component failure. Hock *et al.* [49] present a framework with the same objectives, but also considering metrics such as latency, component failures and load balancing. Their major conclusion is the impossibility of finding an optimal solution to place the controller that satisfies various different criteria, *e.g.,* latency and backup communication. Note that this resilience discipline is strongly related to dependability.

Several studies focused on links and routing protection are available. Nguyen *et al.* [141] define algorithms for finding alternative paths if a link disruption occurs in the network; Stephens *et al.* [144] present an architecture called *Plinko*, which is provably resilient to t link failures when the size of flow tables in the *data plane* is sufficiently large to accommodate backup flow rules; Borokhovich *et al.* [147] use graph theory to model the network as a graph and run algorithms such as Depth-first search (DFS) and Breadth-first search (BFS) to analyze the routing problem and ensure connectivity when a failure occurs.

Delay: communication delays can occur due to the rupture of intermediate links and, to ensure connectivity, alternative communication paths can be used. Link delays resulting from congestion due to the intensive use of network resources may be difficult to conceal [159]. When the problem is less severe, routing algorithms may solve part of the problem, as shown by Heller *et al.* [47] and Hock *et al.* [49]. These efforts investigate the influence that the position of the controller has on the communication latency between controller and switch. Phemius *et al.* [138] point out that switch buffers in the *data plane* play an important role in the overall performance of the network.

Another possible solution is to exploit parallelism to avoid network delay, as proposed by Cai *et al.* [140]. Their system implements a middlebox that handles requests to the controller in parallel and uses alternative paths to communicate. The same principle can be used in the OFLOPS platform [143], which offers support for the rapid development and test of network components. Further, in order to reduce the impact of communication delay on the performance of the network, Mahmoodi *et al.* [146] propose a modular redesign of the intermediate links between the core and mobile networks to handle increasing traffic volumes.

Mobility: in the context of wireless networks, mobility is an important characteristic for the convenience of users. Resilience strategies may use the concept of mobility in the face of a challenge, *e.g.,* a device may need to temporarily migrate to another region until normal service is re-established. Challenges related to user mobility are summarized in the work of Pupatwibul *et al.* [128], where the authors recognize that OpenFlow is a suitable technology for dealing with mobility issues. Sabbagh *et al.* [134] propose a solution based on reprogramming OpenFlow switches to solve the problem of relocation rules in order to maintain communication when the user moves from one point to another in network.

Namal *et al.* [129] present an architecture called OpenFlow Host Identity Protocol (OFHIP), which provides a mechanism for switches to change their IP addresses due to malicious attacks. Further, Sahri *et al.* [137] study failures of network components moving in the communication path. Despite the fact that these solutions are not exclusively focused on mobility, they use related concepts to mitigate failures. A few works address mobility in different contexts. Guolin *et al.* [142] define an architecture for heterogeneous radio access networks and deal with aspects of mobility, *e.g.,* QoS. Other examples are the works of Jeon *et al.* [145], Yuhong *et al.* [130], Shengli *et al.* [148] and Ding *et al.* [149], which advocate that SDN can be used for mobility. SchulzZander *et al.* [131] discuss the feasibility of Wi-Fi deployments in the SDN context. Lara *et al.* [150] propose *MobileFirst*, a clean-slate monitoring architecture that addresses concepts such as communication delay using routing based on VLAN tags. Further information about wireless and SDN can be found in the work of Jagadeesan *et al.* [151] and SeongMun *et al.* [156].

SDN offers an opportunity to facilitate the treatment of challenges in mobile networks, such as mobility management. Traditional solutions present limitations related to, for example, routing and continuity of active sessions. The use of decentralized device anchors represents an alternative to address these limitations. Further, the use of DMM (Distributed Mobility Management) in SDN has been advocated by Sperotto *et al.* [152] and in IETF proposals [153, 154, 155]. SDN can assist in mitigating these issues as it provides a flexible architecture for the deployment of network protocols and applications.

Energy: computer networks often demand scalable and massive services, which can give rise to challenges related to energy. Current data centers consume a large amount of energy and according to [133], energy issues in data centers are an important research topic. Briefly, the authors use techniques to dynamically adapt the power consumption in a data center network. One trend observed with respect to energy in the SDN context is that few studies are focused purely on energy issues. Some references are related to dependability when equipment failure is related to lack of power. These works were discussed in previous sections, since they are covered by other resilience disciplines.

Another study addressing energy aspects is presented by Wang *et al.* [135], which optimizes energy consumption by using different routing algorithms. The knowledge about the amount of energy spent by each device in the network to enforce QoS can also be used for energy saving purposes. Also, Pfeiffenberger *et al.* [136] advocate that SDN can be used to improve the management of energy aspects in communication networks.

The most studied topics about energy include optimization of energy consumption in the network and network resilience when power outages occur. The consequence of challenges associated with energy issues is link disruption. Thus, although energy issues are relevant in other disciplines discussed in this survey, these aspects are mostly related to ensuring connectivity among devices and disruption tolerance [7].

4. SDN Resilience: Solutions and Challenges

The number of papers and research efforts that address different aspects of resilience in SDN is rapidly growing. This section discusses specific challenges and

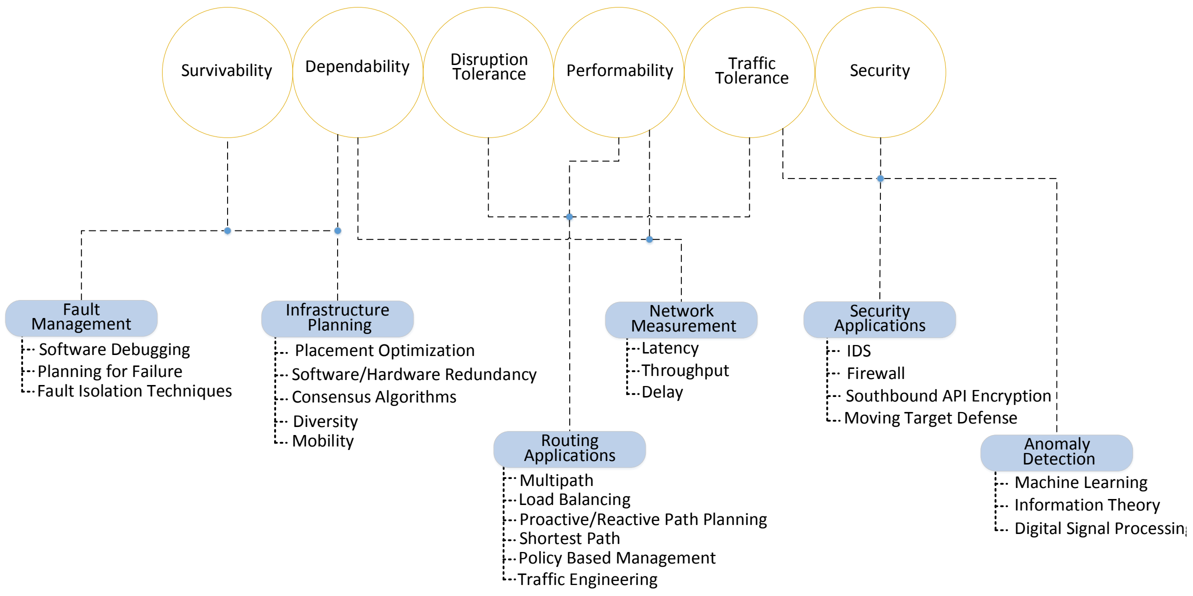


Figure 3: Summary of resilience disciplines, major challenges and areas of interest, and concrete techniques

some of the major issues current research addresses. This is done with a focus on the identified areas and resilience disciplines that were introduced earlier. Subsequently, we present a summary of open research issues and areas that need further work.

4.1. Summary of challenges and current solutions

Figure 3 summarizes the set of identified problems and challenges, according to the different areas and resilience disciplines. At the top of the diagram, the circles represent the six general resilience disciplines, and at the level below the major challenges and prominent research areas that are related to each resilience discipline are depicted. Under each of the challenges, examples of the various techniques, approaches, and concrete instantiations indicate the specific research focus related to the different challenges. They represent the issues most commonly addressed within the discussed literature.

The modular architecture of SDN enables more flexible ways to manage traffic flows [10]. Consequently, resilience solutions based on ***Routing Applications*** are employed in several disciplines (*e.g.,* they are being used in the context of Performability, Traffic Tolerance and Disruption Tolerance). For instance, backup paths and multipath routing can be used to protect the communication network from link disruption and energy outages [136]. Performability also relies on traffic engineering techniques and load balancing to enforce QoS requirements [96]. Further, Policy Based Management can be used to add flexibility to these solutions [99]. Such schemes are usually implemented through extensions of the *control plane* with applications that can monitor and manage traffic via the OpenFlow protocol. Software abstractions, such as topology graphs, can be used to simplify the management of traffic flows and routing [19], thus enabling the use of shortest paths and minimum spanning trees to find optimal solutions for traffic routing.

In the context of ***Infrastructure Planning*** the controller placement problem plays an important role. Since it shares similarity with the classic *facility location* optimization problem [160], several proposed solutions use a graph representations of the network topology to determine the optimal placement of network controllers [47, 49]. Also, solutions based on hardware and software redundancy have been widely investigated [35]. This is due to the fact that SDN offers a flexible architecture to manage redundant devices [30]. Further, schemes such as consensus algorithms to elect a new replica in case of a failure help to maintain consistency between components and their replicas [161]. In wireless networks, research indicates that SDN can assist with the implementation of solutions to guarantee connectivity between devices, *e.g.,* through software abstractions to change IP addresses of devices that migrate and re-route flows to guarantee communication [76].

In the investigated papers ***Fault Management*** often exploits the programmability features offered by the *control plane*, which enables the implementation

Table 7: Key findings observed about the current research on resilience in SDN

Discipline	Key aspects observed
Survivability	(*i*) there are cost issues that prevent the deployment of fully fault tolerant systems; (*ii*) management requirements of redundant devices can be high, for example, to maintain their consistency;
Dependability	(*i*) controller placement is the most studied problem in this area; (*ii*) maintainability is an uncovered field that can give rise to interesting research, for example, with the addition of a dedicated *management plane* to the SDN architecture;
Security	(*i*) several works focus on porting solutions used in traditional networks (*e.g.,* firewalls, IDSes) to SDN. Their main goal is to implement these techniques with more flexibility; (*ii*) mis-configurations and human-dependence in the use of TLS between the controller and switches can compromise the integrity and confidentiality of the communication;
Performability	(*i*) although QoS support in SDN is far from optimal, several solutions are more flexible than existing ones in traditional networks; (*ii*) the controller placement is an issue that can impact communication delay, latency and throughput of the network;
Traffic Tolerance	(*i*) this is the most developed resilience discipline because the SDN architecture has been traditionally concerned with the innovation of routing protocols; (*ii*) well-known solutions to mitigate DDoS attacks can be successfully implemented in SDN;
Disruption Tolerance	(*i*) again, controller placement is critical for protecting the network from disruption; (*ii*) solutions related to survivability are frequently used, such as path redundancy and link redundancy;

of network applications related to software debugging [33]. Also, there are sophisticated monitoring applications capable of collecting information about topology changes, device crashes and link disruptions. These applications might also perform fault isolation, thereby creating a reliable environment for fault detection and mitigation. The use of *group tables* [162] in an OpenFlow switch is an example of how SDN can simplify capacity planning, by enabling the definition of backup flow rules in the switch.

Resilience approaches that rely upon ***Network Measurement*** are the most effective when they focus on measuring latency, throughput and delay. These metrics can be used for assessing QoS and the degree of dependability that the network can offer.

The work on ***Security Applications*** can be divided into two broad sets: *(i)* security solutions built on top of SDN and *(ii)* security of the SDN architecture itself. Within the first group, there are several implementations of firewalls and IDSes that can perform their functions more flexibly and with lower management cost [70]. Within the second group, research is focused on mitigating intrusions in the *control plane*; vulnerabilities in the TLS protocol; and protecting the *control plane* against malicious attacks using network-wide policies and moving target defense [74].

Finally, there is a large amount of work on ***Anomaly Detection***, which has a strong relationship with the Security discipline, but also plays an important role in the Traffic Tolerance discipline. Anomaly detection schemes specific to SDN rely on a global network view to collect flow statistics and perform packet sampling. Most of these solutions rely on machine learning, information theory and digital signal processing techniques [163].

Ultimately, the resilience challenges observed can be divided into two classes. The first class refers to challenges related to the SDN architecture itself independently of any given implementation (*e.g.,* related to infrastructure planning and network measurement). For example, the controller placement is a theoretical problem that is relevant regardless of the controller implementation. The second class subsumes resilience challenges that depend on specific SDN implementations (*e.g.,* routing and security applications). In this case, the solutions in the literature are frequently based on the OpenFlow specification or highly dependent on the functionality provided by the controller implementation.

4.2. Open research questions and lessons learned

Several research questions related to resilience in SDN remain open. For example, high hardware costs related to fully fault tolerant systems can be partially mitigated in this scenario through the use of virtualized infrastructures. In this context, Network Functions Virtualization (NFV) [164] in the *application plane* can assist the development of new solutions in this area. There is also no real resilience metrics related to software implementations and their quality. Thus, Software Engineering practices could be a source for such new metrics to ensure a methodology for software implementations. Additionally, emerging types of traffic profiles suggest that applications related to traffic classification will be very useful for future SDN environments. This should be an active area of research for the next years, as well as the development of monitoring applications that rely on the global network view supported by the SDN architecture. Finally, the initial proposal of the OpenFlow protocol supports limited QoS capabilities, but the development of new protocols can create novel ways to tag flows, enabling sophisticated applications to enforce QoS in the network. Table 7 highlights these points and summarizes the main lessons learned from this study. Note that in addition to specific issues related to these individual disciplines, there is also the need to address resilience challenges that span across several disciplines. This might require the co-ordination of resilience mechanisms that operate at different layers and systems elements (*i.e.,* multi-level resilience), correlating data and also taking coordinated actions. Ultimately, an overall resilience architecture would then be able to discover network and systems anomalies more quickly, and enforce countermeasures at the most appropriate locations.

5. Concluding Remarks

Resilience in Software-Defined Networking (SDN) is the subject of intense investigation by the academic and industrial community in general. As SDN is a relatively new concept, a wide range of solutions to classical network problems have been revisited using this architecture, and many problems continue to be challenging. In this article, we have presented a comprehensive survey on the support for resilience in the SDN architecture, categorizing the existing research efforts on resilience across the different SDN conceptual planes. Furthermore, this survey has presented an overall view of resilience research, describing the trends and key aspects observed, as well as the evolution of this area since 2008, when the widely-adopted OpenFlow protocol was proposed.

The number of research projects that address resilience aspects in SDN has grown significantly. This can be observed by the number of papers included in our survey, and also by the number of research calls issued in this topic recently. The main result of our survey is a comprehensive view of the research space in SDN resilience demonstrating that *(i)* the *data plane* can be protected against link disruption, device failures and malicious attacks using applications placed in the *control* or *application planes*; *(ii)* the *control plane* has resilience requirements related to the consistency between several network controller instances, the security of these devices and general fault management over the entire network. There are several ways to decide where network controllers will be placed and this decision is critical for network operation. Additional controllers may be deployed according to security and survivability requirements; *(iii)* the *application plane* can accommodate several types of network applications, thus promoting research on more sophisticated resilience systems to protect the network against a wide range of challenges. High-level policy languages, such as Procera [125] and Pyretic [126], and troubleshooting systems can also be used to facilitate these tasks.

We emphasize that many of the resilience challenges are due to limitations in the implementation of the components used to realize the SDN paradigm. For example, *(i)* the OpenFlow protocol can be unsafe if TLS is not set up correctly; *(ii)* the Floodlight controller exposes almost all of its functionality through a REST API (possibly allowing illegitimate applications to gather network data); and *(iii)* the *listener mode* functionality (present in many OpenFlow switches) may allow the establishment of connections in a pre-configured port without authentication. Despite the efforts reported in this survey, there are still a number of open issues related to the resilience disciplines investigated, such as the co-ordination of different types of resilience schemes regarding performance and consistency. Consequently, research that takes a more systematic view of resilience systems is required (*e.g.,* considering resilience aspects across different system layers). This article assists in the identification of these aspects that demand further research.

References

[1] B. Fortz, M. Thorup, Optimizing OSPF/IS-IS Weights in a Changing World, IEEE Journal on Selected Areas in Communications 20 (4) (2006) 756–767. doi:10.1109/JSAC.2002.1003042.

[2] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks, IEEE Communications Surveys & Tutorials 16 (3) (2014) 1617–1634. doi:10.1109/SURV.2014.012214.00180.

[3] D. Klein, M. Jarschel, An OpenFlow Extension for the OMNeT++ INET Framework, in: International Conference on Simulation Tools and Techniques (ICST), Brussels, Belgium, 2013, pp. 322–329.

[4] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, J. van der Merwe, The Case for Separating Routing from Routers, in: Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture, FDNA '04, ACM, New York, NY, USA, 2004, pp. 5–12. doi:10.1145/1016707.1016709.

[5] S. Agarwal, M. Kodialam, T. Lakshman, Traffic Engineering in Software Defined Networks, in: IEEE Proceedings of the INFOCOM, Turin, 2013, pp. 2211–2219. doi:10.1109/INFCOM.2013.6567024.

[6] E. Cetinkaya, J. Sterbenz, A taxonomy of network challenges, in: 9th International Conference on the Design of Reliable Communication Networks (DRCN), 2013, 2013, pp. 322–330.

[7] J. P. G. Sterbenz, D. Hutchison, E. K. Cetinkaya, A. Jabbar, J. P. Rohrer, M. Scholler, P. Smith, Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines, Computer Networks. 54 (8) (2010) 1245–1265. doi:10.1016/j.comnet.2010.03.005.

[8] A. Schaeffer-Filho, P. Smith, A. Mauthe, D. Hutchison, Network resilience with reusable management patterns, IEEE Communications Magazine 52 (7) (2014) 105–115. doi:10.1109/MCOM.2014.6852091.

[9] C. R. G. Taveira João A., Landa R., P. George, Software-defined network support for transport resilience, in: IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1 – 8. doi:10.1109/NOMS.2014.6838243.

[10] N. Feamster, J. Rexford, E. Zegura, The Road to SDN, Queue - Large-Scale Implementations, Volume 11 Issue 12 11 (12) (2013) 20:20–20:40. doi:10.1145/2559899.2560327.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, SIGCOMM Computer Communication Review 38 (2) (2008) 69–74. doi:10.1145/1355734.1355746.

[12] Y. Jarraya, T. Madi, M. Debbabi, A Survey and a Layered Taxonomy of Software-Defined Networking, IEEE Communications Surveys Tutorials 16 (4) (2014) 1955–1980. doi:10.1109/COMST.2014.2320094.

[13] H. Cui, Y. Zhu, Y. Yao, L. Yufeng, Y. Liu, Design of intelligent capabilities in SDN, in: 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE), 2014, pp. 1–5. doi:10.1109/VITAE.2014.6934459.

[14] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, G. Parulkar, ONOS: Towards an Open, Distributed SDN OS, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 1–6. doi:10.1145/2620728.2620744.

[15] Open Networking Foundation, OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01), available at: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Accessed: August 2014 (December 31 2009).

[16] H. Farhadi, P. Du, A. Nakao, Enhancing OpenFlow actions to offload packet-in processing, Asia-Pacific Network Operations and Management Symposium (APNOMS) (2014) 1–6doi:10.1109/APNOMS.2014.6996603.

[17] J. de Almeida Amazonas, G. Santos-Boada, J. Solé-Pareta, A critical review of OpenFlow/SDN-based networks, in: 16th International Conference on Transparent Optical Networks (ICTON), 2014, pp. 1–5. doi:10.1109/ICTON.2014.6876509.

[18] H. Nakayama, T. Mori, S. Ueno, Y. Watanabe, T. Hayashi, An implementation model and solutions for stepwise introduction of SDN, 16th Asia-Pacific Network Operations and Management Symposium (APNOMS) (2014) 1–4doi:10.1109/APNOMS.2014.6996576.

[19] G. Pantuza, F. Sampaio, L. F. Vieira, D. Guedes, M. A. Vieira, Network management through graphs in Software Defined Networks, in: 10th International Conference on Network and Service Management (CNSM), 2014, 2014, pp. 400–405. doi:10.1109/CNSM.2014.7014202.

[20] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, The Cost of a Cloud: Research Problems in Data Center Networks, SIGCOMM Computer Communication Review 39 (1) (2008) 68–73. doi:10.1145/1496091.1496103.

[21] A. Lara, A. Kolasani, B. Ramamurthy, Network Innovation using OpenFlow: A Survey, IEEE Communications Surveys Tutorials 16 (1) (2014) 493–512. doi:10.1109/SURV.2013.081313.00105.

[22] L. Schiff, M. Borokhovich, S. Schmid, Reclaiming the Brain: Useful OpenFlow Functions in the Data Plane, in: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, HotNets-XIII, ACM, New York, NY, USA, 2014, pp. 7:1–7:7. doi:10.1145/2670518.2673874.

[23] M. Casado, T. Koponen, S. Shenker, A. Tootoonchian, Fabric: A Retrospective on Evolving SDN, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 85–90. doi:10.1145/2342441.2342459.

[24] V. Caraguay, A. Leonardo, L. I. Barona Lopez, L. J. Garcia Villalba, Evolution and Challenges of Software Defined Networking, in: IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1–7.

[25] F. Hu, Q. Hao, K. Bao, A Survey on Software Defined Networking (SDN) and OpenFlow: From Concept to Implementation, in: IEEE Communications Surveys & Tutorials, Vol. 16, 2014, pp. 2181–2206. doi:10.1109/COMST.2014.2326417.

[26] A. Kwasinski, W. Weaver, P. Chapman, P. Krein, Telecommunications Power Plant Damage Assessment Caused by Hurricane Katrina - Site Survey and Follow-Up Results, in: 28th Annual International Telecommunications Energy Conference (INTELEC '06), 2006, pp. 1–8. doi:10.1109/INTLEC.2006.251644.

[27] I. Onyeji, M. Bazilian, C. Bronk, Cyber Security and Critical Energy Infrastructure, The Electricity Journal 27 (2) (2014) 52 – 60.

[28] P. Cholda, A. Mykkeltveit, B. Helvik, O. Wittner, A. Jajszczyk, A survey of resilience differentiation frameworks in communication networks, IEEE Communications Surveys Tutorials 9 (4) (2007) 32–55. doi:10.1109/COMST.2007.4444749.

[29] P. Smith, D. Hutchison, J. Sterbenz, M. Schöller, A. Fessi, M. Karaliopoulos, C. Lac, B. Plattner, Network resilience: a systematic approach, IEEE Communications Magazine 49 (7) (2011) 88–97. doi:10.1109/MCOM.2011.5936160.

[30] M. Reitblatt, M. Canini, A. Guha, N. Foster, FatTire: Declarative Fault Tolerance for Software-defined Networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 109–114. doi:10.1145/2491185.2491187.

[31] B. Chandrasekaran, T. Benson, Tolerating SDN Application Failures with LegoSDN, in: Proceedings of the 13th ACM Workshop on Hot Topics in Networks, HotNets-XIII, ACM, New York, NY, USA, 2014, pp. 22:1–22:7. doi:10.1145/2670518.2673880.

[32] B. Heller, C. Scott, N. McKeown, S. Shenker, A. Wundsam, H. Zeng, S. Whitlock, V. Jeyakumar, N. Handigol, J. McCauley, K. Zarifis, P. Kazemian, Leveraging SDN Layering to Systematically Troubleshoot Networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 37–42. doi:10.1145/2491185.2491197.
URL http://doi.acm.org/10.1145/2491185.2491197

[33] M. Canini, D. Venzano, P. Perešíni, D. Kostić, J. Rexford, A NICE Way to Test Openflow Applications, in: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 10–10.

[34] C. Scott, A. Wundsam, B. Raghavan, A. Panda, A. Or, J. Lai,

E. Huang, Z. Liu, A. El-Hassany, S. Whitlock, H. Acharya, K. Zarifis, S. Shenker, Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences, in: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, ACM, New York, NY, USA, 2014, pp. 395–406. doi:10.1145/2619239.2626304.

[35] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, A. Vahdat, B4: Experience with a Globally-deployed Software Defined Wan, SIGCOMM Computer Communication Review. 43 (4) (2013) 3–14. doi:10.1145/2534169.2486019.

[36] D. Williams, H. Jamjoom, Cementing High Availability in Openflow with RuleBricks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 139–144. doi:10.1145/2491185.2491206.

[37] F. Botelho, F. Valente Ramos, D. Kreutz, A. Bessani, On the Feasibility of a Consistent and Fault-Tolerant Data Store for SDNs, in: Second European Workshop on Software Defined Networks (EWSDN), 2013, pp. 38–43. doi:10.1109/EWSDN.2013.13.

[38] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, D. Gelernter, Traffic Engineering with Forward Fault Correction, in: Proceedings of the 2014 ACM Conference on SIGCOMM, SIGCOMM '14, ACM, New York, NY, USA, 2014, pp. 527–538. doi:10.1145/2619239.2626314.

[39] J. Li, J. Hyun, J.-H. Yoo, S. Baik, J.-K. Hong, Scalable failover method for Data Center Networks using OpenFlow, in: IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–6. doi:10.1109/NOMS.2014.6838393.

[40] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, P. Skoldstrom, Scalable fault management for OpenFlow, in: IEEE International Conference on Communications (ICC), 2012, pp. 6606–6610. doi:10.1109/ICC.2012.6364688.

[41] P. Fonseca, R. Bennesby, E. Mota, A. Passito, A replication component for resilient OpenFlow-based networking, in: IEEE Network Operations and Management Symposium (NOMS), 2012, pp. 933–939. doi:10.1109/NOMS.2012.6212011.

[42] A. Tootoonchian, Y. Ganjali, HyperFlow: A Distributed Control Plane for OpenFlow, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, INM/WREN'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 3–3.

[43] L. Muller, R. Oliveira, M. Luizelli, L. Gaspary, M. Barcellos, Survivor: An enhanced controller placement strategy for improving SDN survivability, in: IEEE Global Communications Conference (GLOBECOM), 2014, pp. 1909–1915. doi:10.1109/GLOCOM.2014.7037087.

[44] Y. Zhang, N. Beheshti, R. Manghirmalani, NetRevert: Rollback Recovery in SDN, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 231–232. doi:10.1145/2620728.2620779.

[45] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable Secure Computing 1 (1) (2004) 11–33. doi:10.1109/TDSC.2004.2.

[46] R. Veisllari, N. Stol, S. Bjornstad, C. Raffaelli, Scalability analysis of SDN-controlled optical ring MAN with hybrid traffic, in: IEEE International Conference on Communications (ICC), 2014, pp. 3283–3288. doi:10.1109/ICC.2014.6883827.

[47] B. Heller, R. Sherwood, N. McKeown, The Controller Placement Problem, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 7–12. doi:10.1145/2342441.2342444.

[48] D. Li, L. Ruan, L. Xiao, M. Zhu, W. Duan, Y. Zhou, M. Chen, Y. Xia, M. Zhu, High availability for Non-stop network controller, in: IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014, pp. 1–5. doi:10.1109/WoWMoM.2014.6918986.

[49] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, Pareto-optimal resilient controller placement in SDN-based core networks, in: 25th International Teletraffic Congress (ITC), 2013, pp. 1–9.

[50] F. J. Ros, P. M. Ruiz, Five Nines of Southbound Reliability in Software-defined Networks, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 31–36. doi:10.1145/2620728.2620752.

[51] N. Beheshti, Y. Zhang, Fast failover for control traffic in Software-defined Networks, in: IEEE Global Communications Conference (GLOBECOM), 2012, pp. 2665–2670. doi:10.1109/GLOCOM.2012.6503519.

[52] M. Santos, B. Nunes, K. Obraczka, T. Turletti, B. de Oliveira, C. Margi, Decentralizing SDN's control plane, in: IEEE 39th Conference on Local Computer Networks (LCN), 2014, pp. 402–405. doi:10.1109/LCN.2014.6925802.

[53] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an Elastic Distributed SDN Controller, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 7–12. doi:10.1145/2491185.2491193.

[54] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: Scaling Flow Management for High-performance Networks, in: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM '11, ACM, New York, NY, USA, 2011, pp. 254–265.

[55] Y.-J. Chen, F.-Y. Lin, L.-C. Wang, B.-S. Lin, A dynamic security traversal mechanism for providing deterministic delay guarantee in SDN, in: IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014, pp. 1–6. doi:10.1109/WoWMoM.2014.6918983.

[56] X. Wang, Z. Liu, J. Li, B. Yang, Y. Qi, Tualatin: Towards network security service provision in cloud datacenters, in: 23rd International Conference on Computer Communication and Networks (ICCCN), 2014, pp. 1–8. doi:10.1109/ICCCN.2014.6911782.

[57] S. Seeber, G. D. Rodosek, Improving network security through SDN in cloud scenarios, in: 10th International Conference on Network and Service Management (CNSM), 2014, pp. 376–381. doi:10.1109/CNSM.2014.7014198.

[58] M. Tasch, R. Khondoker, R. Marx, K. Bayarou, Security Analysis of Security Applications for Software Defined Networks, in: Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference, AINTEC '14, ACM, New York, NY, USA, 2014, pp. 23:23–23:30. doi:10.1145/2684793.2684797.

[59] K. Benton, L. J. Camp, C. Small, OpenFlow Vulnerability Assessment, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 151–152. doi:10.1145/2491185.2491222.

[60] H. Li, P. Li, S. Guo, S. Yu, Byzantine-resilient secure software-defined networks with multiple controllers, in: IEEE International Conference on Communications (ICC), 2014, pp. 695–700. doi:10.1109/ICC.2014.6883400.

[61] A. Zaalouk, R. Khondoker, R. Marx, K. Bayarou, OrchSec: An orchestrator-based architecture for enhancing network-security

using Network Monitoring and SDN Control functions, in: IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–9. doi:10.1109/NOMS.2014.6838409.

[62] A. K. Nayak, A. Reimers, N. Feamster, R. Clark, Resonance: Dynamic Access Control for Enterprise Networks, in: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, WREN '09, ACM, New York, NY, USA, 2009, pp. 11–18. doi:10.1145/1592681.1592684.

[63] L. Schehlmann, S. Abt, H. Baier, Blessing or curse? Revisiting security aspects of Software-Defined Networking, in: 10th International Conference on Network and Service Management (CNSM), 2014, pp. 382–387. doi:10.1109/CNSM.2014.7014199.

[64] D. Kreutz, F. M. Ramos, P. Verissimo, Towards Secure and Dependable Software-defined Networks, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 55–60. doi:10.1145/2491185.2491199.

[65] Y. Wang, Y. Zhang, V. Singh, C. Lumezanu, G. Jiang, NetFuse: Short-circuiting traffic surges in the cloud, in: IEEE International Conference on Communications (ICC), 2013, pp. 3514–3518. doi:10.1109/ICC.2013.6655095.

[66] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, G. Gu, A Security Enforcement Kernel for OpenFlow Networks, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 121–126. doi:10.1145/2342441.2342466.

[67] D. Mazières, D. Shasha, Building Secure File Systems out of Byzantine Storage, in: Proceedings of the Twenty-first Annual Symposium on Principles of Distributed Computing, PODC '02, ACM, New York, NY, USA, 2002, pp. 108–117. doi:10.1145/571825.571840.

[68] J. Collings, J. Liu, An OpenFlow-Based Prototype of SDN-Oriented Stateful Hardware Firewalls, in: IEEE 22nd International Conference on Network Protocols (ICNP), 2014, pp. 525–528. doi:10.1109/ICNP.2014.83.

[69] B. Anwer, T. Benson, N. Feamster, D. Levin, J. Rexford, A Slick Control Plane for Network Middleboxes, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 147–148. doi:10.1145/2491185.2491223.

[70] H. Hu, W. Han, G.-J. Ahn, Z. Zhao, FLOWGUARD: Building Robust Firewalls for Software-defined Networks, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 97–102. doi:10.1145/2620728.2620749.

[71] S. K. Fayazbakhsh, V. Sekar, M. Yu, J. C. Mogul, FlowTags: Enforcing Network-wide Policies in the Presence of Dynamic Middlebox Actions, in: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 19–24. doi:10.1145/2491185.2491203.

[72] T. Xing, D. Huang, L. Xu, C.-J. Chung, P. Khatkar, SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment, in: Second GENI Research and Educational Experiment Workshop (GREE), 2013, pp. 89–92. doi:10.1109/GREE.2013.25.

[73] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, N. Zeldovich, Delegating Network Security with More Information, in: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking, WREN '09, ACM, New York, NY, USA, 2009, pp. 19–26. doi:10.1145/1592681.1592685.

[74] P. Kampanakis, H. Perros, T. Beyene, SDN-based solutions for Moving Target Defense network protection, in: IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014, pp. 1–6. doi:10.1109/WoWMoM.2014.6918979.

[75] E. G. Silva, L. Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville, A. Schaeffer-Filho, Capitalizing on SDN-Based SCADA Systems: An Anti-Eavesdropping Case-Study, in: 15th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2015), Ottawa, Canada, 2015, pp. 165–173.

[76] J. H. Jafarian, E. Al-Shaer, Q. Duan, Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 127–132. doi:10.1145/2342441.2342467.

[77] J. R. Ballard, I. Rae, A. Akella, Extensible and Scalable Network Monitoring Using OpenSAFE, in: Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking, INM/WREN'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 8–8.

[78] R. Smeliansky, SDN for network security, in: First International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014, pp. 1–5. doi:10.1109/MoNeTeC.2014.6995602.

[79] C. Schlesinger, A. Story, S. Gutz, N. Foster, D. Walker, Splendid Isolation: Language-Based Security for Software-Defined Networks, in: Proceedings of Workshop on Hot Topics in Software Defined Networking, 2012, pp. 79–84.

[80] Z. Abaid, M. Rezvani, S. Jha, MalwareMonitor: An SDN-based Framework for Securing Large Networks, in: Proceedings of the 2014 CoNEXT on Student Workshop, CoNEXT Student Workshop '14, ACM, New York, NY, USA, 2014, pp. 40–42. doi:10.1145/2680821.2680829.

[81] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, M. Tyson, FRESCO: Modular Composable Security Services for Software-Defined Networks, in: Network and Distributed System Security Symposium (NDSS), 2013, pp. 1–16.

[82] S. Kumar, T. Kumar, G. Singh, M. S. Nehra, OpenFlow switch with intrusion detection system, International J. Schientific Research Engineering & Techonology (IJSRET) 1 (2012) 1–4.

[83] D. Li, X. Hong, J. Bowman, Evaluation of security vulnerabilities by using ProtoGENI as a launchpad, in: IEEE Global Telecommunications Conference (GLOBECOM 2011), IEEE, 2011, pp. 1–6.

[84] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, M. Yu, SIMPLE-fying Middlebox Policy Enforcement Using SDN, in: Proceedings of the ACM SIGCOMM 2013, New York, NY, USA, 2013, pp. 27–38. doi:10.1145/2486001.2486022.

[85] S. Son, S. Shin, V. Yegneswaran, P. Porras, G. Gu, Model checking invariant security properties in OpenFlow, in: IEEE International Conference on Communications (ICC), IEEE, 2013, pp. 1974–1979.

[86] R. Braga, Braga, E. Mota, Mota, A. Passito, Passito, Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow, in: Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks, LCN '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 408–415. doi:10.1109/LCN.2010.5735752.

[87] S. A. Mehdi, J. Khalid, S. A. Khayam, Revisiting Traffic Anomaly Detection Using Software Defined Networking, in: Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, RAID'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 161–180.

[88] K. Giotis, G. Androulidakis, V. Maglaris, Leveraging SDN for Efficient Anomaly Detection and Mitigation on Legacy Networks, in: Third European Workshop on

Software Defined Networks (EWSDN), 2014, pp. 85–90. doi:10.1109/EWSDN.2014.24.

[89] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, Flowvisor: A network virtualization layer, OpenFlow Switch Consortium, Tech. Rep.

[90] R. Kloti, Openflow: A security analysis, in: IEEE Proceedings of the Workshop on Secure Network Protocols (NPSec), 2013, pp. 1–6.

[91] S. Scott-Hayward, G. O'Callaghan, S. Sezer, SDN security: A survey, in: IEEE SDN for Future Networks and Services (SDN4FNS), IEEE, 2013, pp. 1–7.

[92] Y. Zhang, S. Natarajan, X. Huang, N. Beheshti, R. Manghirmalani, A Compressive Method for Maintaining Forwarding States in SDN Controller, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 139–144. doi:10.1145/2620728.2620759.

[93] W. Zhou, L. Li, W. Chou, SDN Northbound REST API with Efficient Caches, in: IEEE International Conference on Web Services (ICWS), 2014, pp. 257–264. doi:10.1109/ICWS.2014.46.

[94] H. Egilmez, S. Dane, K. Bagci, A. Tekalp, OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks, in: Asia-Pacific Signal Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012, pp. 1–8.

[95] A. Akella, K. Xiong, Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN), in: IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC), 2014, pp. 7–13. doi:10.1109/DASC.2014.11.

[96] J. Huang, Q. Duan, Q. Chen, Y. Sun, Y. Tanaka, W. Wang, Guaranteeing End-to-end Quality-of-service with a Generic Routing Approach, ACM SIGAPP Applied Computing Review 14 (2) (2014) 8–22. doi:10.1145/2656864.2656865.

[97] P. Xiong, H. Hacigumus, J. F. Naughton, A Software-defined Networking Based Approach for Performance Management of Analytical Queries on Distributed Data Stores, in: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, SIGMOD '14, ACM, New York, NY, USA, 2014, pp. 955–966. doi:10.1145/2588555.2593681.

[98] W. Wendong, Q. Qinglei, G. Xiangyang, H. Yannan, Q. Xirong, Autonomic QoS management mechanism in Software Defined Network, China Communications 11 (7) (2014) 13–23. doi:10.1109/CC.2014.6895381.

[99] C. Cleder Machado, L. Zambenedetti Granville, A. Schaeffer-Filho, J. Araujo Wickboldt, Towards SLA Policy Refinement for QoS Management in Software-Defined Networking, in: IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), 2014, pp. 397–404. doi:10.1109/AINA.2014.148.

[100] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, G. Vaszkun, On QoS Support to Ofelia and OpenFlow, in: European Workshop on Software Defined Networking (EWSDN), 2012, pp. 109–113. doi:10.1109/EWSDN.2012.26.

[101] H. Egilmez, S. Civanlar, A. Tekalp, A distributed QoS routing architecture for scalable video streaming over multi-domain OpenFlow networks, in: 19th IEEE International Conference on Image Processing (ICIP), 2012, pp. 2237–2240. doi:10.1109/ICIP.2012.6467340.

[102] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. Cunha de Lucena, R. Raszuk, Revisiting Routing Control Platforms with the Eyes and Muscles of Software-defined Networking, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 13–18. doi:10.1145/2342441.2342445.

[103] S. Shin, V. Yegneswaran, P. Porras, G. Gu, AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-defined Networks, in: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, CCS '13, ACM, New York, NY, USA, 2013, pp. 413–424. doi:10.1145/2508859.2516684.

[104] L. Mchale, J. Case, P. Gratz, A. Sprintson, Stochastic Pre-classification for SDN Data Plane Matching, in: IEEE 22nd International Conference on Network Protocols (ICNP), 2014, pp. 596–602. doi:10.1109/ICNP.2014.95.

[105] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, W. Chou, A roadmap for traffic engineering in SDN-OpenFlow networks, Computer Networks 71 (0) (2014) 1 – 30.

[106] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, V. Maglaris, Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments, Computer Networks 62 (0) (2014) 122 – 136.

[107] R. Ramos, M. Martinello, C. Esteve Rothenberg, SlickFlow: Resilient source routing in Data Center Networks unlocked by OpenFlow, in: IEEE 38th Conference on Local Computer Networks (LCN), 2013, pp. 606–613.

[108] J. Alcorn, C. Chow, A framework for large-scale modeling and simulation of attacks on an OpenFlow network, in: 23rd International Conference on Computer Communication and Networks (ICCCN), 2014, pp. 1–6. doi:10.1109/ICCCN.2014.6911848.

[109] T. Benson, A. Anand, A. Akella, M. Zhang, MicroTE: Fine Grained Traffic Engineering for Data Centers, in: Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies, CoNEXT '11, ACM, New York, NY, USA, 2011, pp. 8:1–8:12. doi:10.1145/2079296.2079304.

[110] M. Belyaev, S. Gaivoronski, Towards load balancing in SDN-networks during DDoS-attacks, in: First International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC), 2014, pp. 1–6. doi:10.1109/MoNeTeC.2014.6995578.

[111] S. Raza, G. Huang, C.-N. Chuah, S. Seetharaman, J. P. Singh, MeasuRouting: A Framework for Routing Assisted Traffic Monitoring, IEEE/ACM Transactions on Networking (TON) 20 (1) (2012) 45–56. doi:10.1109/TNET.2011.2159991.

[112] A. Wang, Y. Guo, F. Hao, T. Lakshman, S. Chen, Scotch: Elastically Scaling Up SDN Control-Plane Using vSwitch Based Overlay, in: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, CoNEXT '14, ACM, New York, NY, USA, 2014, pp. 403–414. doi:10.1145/2674005.2675002.

[113] D. Venmani, D. Zeghlache, Y. Gourhant, Demystifying Link Congestion in 4G-LTE Backhaul Using OpenFlow, in: 5th International Conference on New Technologies, Mobility and Security (NTMS), 2012, pp. 1–8. doi:10.1109/NTMS.2012.6208711.

[114] A. Passito, E. Mota, R. Bennesby, P. Fonseca, AgNOS: A Framework for Autonomous Control of Software-Defined Networks, in: IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), 2014, pp. 405–412. doi:10.1109/AINA.2014.114.

[115] J. Li, S. Berg, M. Zhang, P. Reiher, T. Wei, Drawbridge: Software-defined DDoS-resistant Traffic Engineering, in: Proceedings of the 2014 ACM Conference on SIGCOMM, New York, NY, USA, 2014, pp. 591–592. doi:10.1145/2619239.2631469.

[116] S. Sun, L. Han, S. Cho, S. Han, J. Wang, B. Paillassa, Performance optimization of media distribution in overlay networks using OpenFlow, in: International Conference on Information Networking (ICOIN), 2014, pp. 276–281. doi:10.1109/ICOIN.2014.6799481.

[117] M. Shtern, R. Sandel, M. Litoiu, C. Bachalo, V. Theodorou, Towards Mitigation of Low and Slow Application DDoS Attacks, in: IEEE International Conference on Cloud Engineering (IC2E), 2014, pp. 604–609. doi:10.1109/IC2E.2014.38.

[118] F. de Oliveira Silva, M. Goncalves, J. de Souza Pereira, R. Pasquini, P. Rosa, S. Kofuji, On the analysis of multicast traffic over the Entity Title Architecture, in: 18th IEEE International Conference on Networks (ICON), 2012, pp. 30–35. doi:10.1109/ICON.2012.6506529.

[119] S. Laga, T. Van Cleemput, F. Van Raemdonck, F. Vanhoutte, N. Bouten, M. Claeys, F. De Turck, Optimizing scalable video delivery through OpenFlow layer-based routing, in: IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–4. doi:10.1109/NOMS.2014.6838378.

[120] X. Tu, X. Li, J. Zhou, S. Chen, Splicing MPLS and OpenFlow Tunnels Based on SDN Paradigm, in: IEEE International Conference on Cloud Engineering (IC2E), 2014, pp. 489–493. doi:10.1109/IC2E.2014.20.

[121] H. Rodrigues, I. Monga, A. Sadasivarao, S. Syed, C. Guok, E. Pouyoul, C. Liou, T. Rosing, Traffic Optimization in Multi-layered WANs Using SDN, in: IEEE 22nd Annual Symposium on High-Performance Interconnects (HOTI), 2014, pp. 71–78. doi:10.1109/HOTI.2014.23.

[122] R. Bennesby, E. Mota, P. Fonseca, A. Passito, Innovating on Interdomain Routing with an Inter-SDN Component, in: IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), 2014, pp. 131–138. doi:10.1109/AINA.2014.21.

[123] R. Krishnan, D. Krishnaswamy, D. Mcdysan, Behavioral Security Threat Detection Strategies for Data Center Switches and Routers, in: IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW), 2014, pp. 82–87. doi:10.1109/ICDCSW.2014.19.

[124] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, P. Castoldi, Effective flow protection in OpenFlow rings, in: Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference (OFC/NFOEC), 2013, pp. 1–3.

[125] A. Voellmy, H. Kim, N. Feamster, Procera: A Language for High-level Reactive Network Control, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48.

[126] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, D. Walker, Frenetic: A Network Programming Language, Proceedings of the 16th ACM SIGPLAN international conference on Functional programming 46 (9) (2011) 279–291. doi:10.1145/2034574.2034812.

[127] R. Alvizu, G. Maier, Can open flow make transport networks smarter and dynamic? An overview on transport SDN, in: International Conference on Smart Communications in Network Technologies (SaCoNeT), 2014, pp. 1–6. doi:10.1109/SaCoNeT.2014.6867771.

[128] P. Pupatwibul, A. Banjar, A. Sabbagh, R. Braun, Developing an application based on OpenFlow to enhance mobile IP networks, in: IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops), 2013, pp. 936–940.

[129] S. Namal, I. Ahmad, A. Gurtov, M. Ylianttila, Enabling Secure Mobility with OpenFlow, in: IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1–5.

[130] Y. Li, H. Wang, M. Liu, B. Zhang, H. Mao, Software defined networking for distributed mobility management, in: IEEE Globecom Workshops (GC Wkshps), 2013, pp. 885–889. doi:10.1109/GLOCOMW.2013.6825101.

[131] J. Schulz-Zander, N. Sarrar, S. Schmid, Towards a Scalable and Near-sighted Control Plane Architecture for WiFi SDNs, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 217–218. doi:10.1145/2620728.2620772.

[132] S. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Communications Magazine 51 (2) (2013) 136–141. doi:10.1109/MCOM.2013.6461198.

[133] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, ElasticTree: Saving Energy in Data Center Networks, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 17–17.

[134] A. AL Sabbagh, P. Pupatwibul, A. Banjar, R. Braun, Optimization of the OpenFlow controller in wireless environments for enhancing mobility, in: IEEE 38th Conference on Local Computer Networks Workshops (LCN Workshops), 2013, pp. 930–935. doi:10.1109/LCNW.2013.6758534.

[135] J. Wang, X. Chen, C. Phillips, Y. Yan, Energy efficiency with QoS control in dynamic optical networks with {SDN} enabled integrated control plane, Computer Networks 78 (2014) 57 – 67.

[136] T. Pfeiffenberger, J. L. Du, Evaluation of software-defined networking for power systems, in: IEEE International Conference on Intelligent Energy and Power Systems (IEPS), 2014, pp. 181–185. doi:10.1109/IEPS.2014.6874175.

[137] N. M. Sahri, K. Okamura, Fast Failover Mechanism for Software Defined Networking: OpenFlow Based, in: Proceedings of The Ninth International Conference on Future Internet Technologies, CFI '14, ACM, New York, NY, USA, 2014, pp. 16:1–16:2. doi:10.1145/2619287.2619303.

[138] K. Phemius, M. Bouet, OpenFlow: Why latency does matter, in: IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013, pp. 680–683.

[139] Z. Zhu, H. Li, K. Pan, C. Yu, F. Chen, D. Li, Centralized Flat Routing, in: International Conference on Computing, Management and Telecommunications (ComManTel), 2014, pp. 52–57.

[140] Z. Cai, A. L. Cox, T. S. E. Ng, Maestro: A System for Scalable OpenFlow Control, Tech. rep., TSEN Maestro-Technical Report TR10-08 (2011).

[141] K. Nguyen, Q. T. Minh, S. Yamada, A Software-Defined Networking Approach for Disaster-Resilient WANs, in: 22nd International Conference on Computer Communications and Networks (ICCCN), 2013, pp. 1–5. doi:10.1109/ICCCN.2013.6614094.

[142] G. Sun, G. Liu, H. Zhang, W. Tan, Architecture on mobility management in OpenFlow-based radio access networks, in: IEEE Global High Tech Congress on Electronics (GHTCE), 2013, pp. 88–92. doi:10.1109/GHTCE.2013.6767247.

[143] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, A. W. Moore, OFLOPS: An Open Framework for Openflow Switch Evaluation, in: Proceedings of the 13th International Conference on Passive and Active Measurement, PAM'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 85–95.

[144] B. Stephens, A. L. Cox, S. Rixner, Plinko: Building Provably Resilient Forwarding Tables, in: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, ACM, New York, NY, USA, 2013, pp. 26:1–26:7. doi:10.1145/2535771.2535774.

[145] S. Jeon, C. Guimarães, R. L. Aguiar, SDN-based Mobile Networking for Cellular Operators, in: Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture, MobiArch '14, ACM, New York, NY, USA, 2014, pp. 13–18.

[146] T. Mahmoodi, S. Seetharaman, Traffic Jam: Handling the Increasing Volume of Mobile Data Traffic, IEEE Vehicular Technology Magazine 9 (3) (2014) 56–62.

[147] M. Borokhovich, L. Schiff, S. Schmid, Provable Data Plane Connectivity with Local Fast Failover: Introducing Openflow Graph Algorithms, in: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, HotSDN '14, ACM, New York, NY, USA, 2014, pp. 121–126. doi:10.1145/2620728.2620746.

[148] S. Zhang, C. Kai, L. Song, SDN based uniform network architecture for future wireless networks, in: International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2014, pp. 1–5. doi:10.1109/ICCCNT.2014.6963056.

[149] A. Y. Ding, J. Crowcroft, S. Tarkoma, H. Flinck, Software defined networking for security enhancement in wireless mobile networks, Computer Networks 66 (0) (2014) 94 – 101, leonard Kleinrock Tribute Issue: A Collection of Papers by his Students. doi:http://dx.doi.org/10.1016/j.comnet.2014.03.009.

[150] A. Lara, B. Ramamurthy, K. Nagaraja, A. Krishnamoorthy, D. Raychaudhuri, Cut-through switching options in a MobilityFirst network with openflow, in: IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS), 2013, pp. 1–6. doi:10.1109/ANTS.2013.6802877.

[151] N. A. Jagadeesan, B. Krishnamachari, Software-Defined Networking Paradigms in Wireless Networks: A Survey, ACM Computing Surveys 47 (2) (2014) 27:1–27:11. doi:10.1145/2655690.

[152] M. Karimzadeh, A. Sperotto, A. Pras, Software Defined Networking to Improve Mobility Management Performance, in: Monitoring and Securing Virtualized Networks and Services, Vol. 8508 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2014, pp. 118–122.

[153] F. Giust, M. Liebsch, Internet-Draft - Deployment of Control-/Data-Plane separation in DMM, Tech. rep., Internet Engineering Task Force (IETF) (2015).

[154] K. Sun, Y. Kim, Internet-Draft - Use case analysis for supporting flow mobility in DMM, Tech. rep., Internet Engineering Task Force (IETF) (2015).

[155] H. Yang, K. Sun, Y. Kim, Internet-Draft - Routing Optimization with SDN, Tech. rep., Internet Engineering Task Force (IETF) (2015).

[156] S.-M. Kim, H.-Y. Choi, P.-W. Park, S.-G. Min, Y.-H. Han, OpenFlow-based Proxy mobile IPv6 over software defined network (SDN), in: IEEE 11th Consumer Communications and Networking Conference (CCNC), 2014, pp. 119–125.

[157] M. Ramadas, S. Ostermann, B. Tjaden, Detecting anomalous network traffic with self-organizing maps, in: In Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (LNCS), Springer Verlag, 2003, pp. 36–54.

[158] M. Menth, M. Duelli, R. Martin, J. Milbrandt, Resilience Analysis of Packet-Switched Communication Networks, IEEE/ACM Transactions on Networking, IEEE/ACM Transactions on 17 (6) (2009) 1950–1963. doi:10.1109/TNET.2009.2020981.

[159] S. Jain, K. Fall, R. Patra, Routing in a Delay Tolerant Network, in: Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '04, ACM, New York, NY, USA, 2004, pp. 145–158. doi:10.1145/1015467.1015484.

[160] C. Aikens, Facility location models for distribution planning, European Journal of Operational Research 22 (3) (1985) 263 – 279.

[161] M. Pease, R. Shostak, L. Lamport, Reaching Agreement in the Presence of Faults, Journal of the ACM (JACM) 27 (2) (1980) 228–234. doi:10.1145/322186.322188.
URL `http://doi.acm.org/10.1145/322186.322188`

[162] Open Networking Foundation, OpenFlow Switch Specification - Version 1.3.0, available at: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Accessed: August 2014 (June 25, 2012).

[163] A. Marnerides, A. Schaeffer-Filho, A. Mauthe, Traffic anomaly diagnosis in Internet backbone networks: A survey, Computer Networks 73 (0) (2014) 224 – 243. doi:http://dx.doi.org/10.1016/j.comnet.2014.08.007.

[164] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, Network function virtualization: Challenges and opportunities for innovations, IEEE Communications Magazine 53 (2) (2015) 90–97. doi:10.1109/MCOM.2015.7045396.

AppendixD ACCEPTED PAPER – NOMS 2016

Anomaly traffic detection and classification mechanisms need to be flexible and easy to manage in order to detect the ever growing spectrum of anomalies. Detection and classification are difficult tasks because of several reasons, including the need to obtain an accurate and comprehensive view of the network, the ability to detect the occurrence of new attack types, and the need to deal with misclassification. In this paper, we argue that Software-Defined Networking (SDN) form propitious environments for the design and implementation of more robust and extensible anomaly classification schemes. Different than other approaches from the literature, which individually tackle either anomaly detection or classification or mitigation, we present a management framework to perform these tasks jointly. Our proposed framework is called ATLANTIC and it combines the use of information theory to calculate deviations in the entropy of flow tables and a range of machine learning algorithms to classify traffic flows. As a result, ATLANTIC is a flexible framework capable of categorizing traffic anomalies and using the information collected to handle each traffic profile in a specific manner, e.g., blocking malicious flows.

- **Title –**
  ATLANTIC: A Framework for Anomaly Traffic Detection, Classification, and Mitigation in SDN
- **Conference –**
  IEEE/IFIP Network Operations and Management Symposium (NOMS 2016)
- **Qualis –**
  A2
- **Date –**
  April 25, 2016

ATLANTIC: A Framework for Anomaly Traffic Detection, Classification, and Mitigation in SDN

Anderson Santos da Silva, Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, Alberto Schaeffer-Filho
Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
Email: {assilva, jwickboldt, granville, alberto}@inf.ufrgs.br

Abstract—**Anomaly traffic detection and classification mechanisms need to be flexible and easy to manage in order to detect the ever growing spectrum of anomalies. Detection and classification are difficult tasks because of several reasons, including the need to obtain an accurate and comprehensive view of the network, the ability to detect the occurrence of new attack types, and the need to deal with misclassification. In this paper, we argue that Software-Defined Networking (SDN) form propitious environments for the design and implementation of more robust and extensible anomaly classification schemes. Different than other approaches from the literature, which individually tackle either anomaly detection or classification or mitigation, we present a management framework to perform these tasks jointly. Our proposed framework is called ATLANTIC and it combines the use of *information theory* to calculate deviations in the entropy of flow tables and a range of *machine learning* algorithms to classify traffic flows. As a result, ATLANTIC is a flexible framework capable of categorizing traffic anomalies and using the information collected to handle each traffic profile in a specific manner, *e.g.,* blocking malicious flows.**

Keywords—Software-Defined Networking, Network Management, OpenFlow, Anomaly detection

I. Introduction

Computer networks must be resilient and properly deliver the communication services expected by their users [1]. The detection of an ever increasing number of anomalies in network traffic is the key task to achieve resilience. Anomaly detection in traditional computer networks is difficult to achieve because the points of observation are spread along distributed forwarding devices. With the advent of Software-Defined Networking (SDN) [2] [3], however, anomaly detection can be performed at the logically centralized spot created by the SDN controller. SDN in general and the OpenFlow protocol [4] in particular allow building more reliable, extensible, and manageable networks where new network functions can be more easily deployed. However, SDN-based networks are not free of abnormal traffic that can affect the resilience of the network. Still, SDN can facilitate the design of anomaly detection and traffic classifications systems because of several reasons: (*i*) SDN offers a more comprehensive view of the network, (*ii*) SDN supports the easy collection of flow statistics, and (*iii*) SDN includes a dedicated management plane to coordinate dynamic reconfiguration actions.

To the best of our knowledge, there is no framework capable of managing anomaly detection, classification and mitigation in a coordinated manner in SDN environments. We advocate that such a framework should perform these tasks jointly, be fully extensible to accommodate different types of anomalies, and rely on modular software abstractions. To address these issues, in this paper, we introduce the ATLANTIC (*Anomaly deTection and machine LeArNing Traffic classifICation for software-defined networking*) framework for detection, classification, and mitigation of traffic anomalies in SDN-based networks. Anomaly detection and classification are performed in two complementary phases: (*i*) a *lightweight phase*, in which low computation cost methods are executed more frequently to quickly spot potentially malicious flows, and (*ii*) a *heavyweight phase*, where such flows are analyzed and classified according to their abnormal behavior. To instantiate our framework, we employ an information theory approach based on entropy analysis [5] in the *lightweight phase*, whereas in the *heavyweight phase* a machine learning algorithm based on Support Vector Machine (SVM) [6] is used to leverage historical knowledge about past anomalies and to classify the abnormal traffic.

Our main contributions are: (*i*) a strategy that obtains global network information without additional costs to network administrators, such as additional sensors; (*ii*) an architecture to combine several types of anomaly detection, classification, and mitigation techniques in a flexible manner, while avoiding high resource utilization; (*iii*) a publicly available application of how SDN can provide sophisticated software-based management solutions (represented by the lightweight and heavyweight phases) to tackle legacy network problems, such as managing classification techniques. We have developed a prototype system as a proof-of-concept. Our prototype has been implemented in Python and is publicly available in GitHub[1]. We evaluate the instantiation of our ATLANTIC framework to manage an SDN-based environment consisting of 11 switches organized according to the topology of the Federal University of Rio Grande do Sul campus network. In our experimental evaluation, we observed performance, accuracy, and overhead of ATLANTIC, considering distributed denial of service (DDoS) and port scanning attacks.

The remaining of this paper is organized as follows. In Section II, we present background and review related work. In Section III, we introduce ATLANTIC. In Section IV, we present our evaluation and associated results, including a performance analysis of the framework. In Section V, we conclude this paper presenting final remarks and future work.

[1]https://github.com/AndersonSanSilva/ATLANTIC

II. Background and Related Work

Anomaly detection and traffic classification in traditional networks are research areas that have been intensively investigated for years [7] [8] [9]. In SDN-based networks, however, the amount of work on this subject is still much less prominent. SDN-based as well as traditional networks are susceptible to abnormal traffic that can harm network operations and management, *e.g.,* Distributed Denial of Service (DDoS) attacks [10], [11]. As such, abnormal traffic must be detected, classified, and mitigated. Before introducing the ATLANTIC framework, we first present in this section background information on the key anomaly detection techniques and the related work.

Because of the emergence of new traffic behaviors, strategies to detect and classify anomalous traffic are necessary so to protect the network against malicious attacks. In traditional networks, machine learning has been widely used for traffic classification and anomaly detection [12]. These techniques can be divided into two main classes: *supervised learning* and *unsupervised learning*. Supervised learning techniques, such as Naive Bayes [13] and Support Vector Machine (SVM) [6], are suitable for classifying data samples into a range of known attacks. However, supervised learning is unable to handle new types of attacks. SVM typically achieves high classification accuracy [12] and thus it is commonly chosen to compose anomaly detection systems. On the other hand, unsupervised learning techniques, such as K-means [8] and Expectation Maximization [12], are suitable for detecting new types of attacks. However, in general, they need human input to determine the classes of the sampled data, which is generally grouped by similarity. Despite the high accuracy and performance obtained with some techniques, machine learning algorithms tend to suffer from several limitations: *(i)* the difficulty of determining the best set of discriminators to classify flows [13]; *(ii)* the availability of labeled training data for classification [8] [14]; *(iii)* the trade-offs between different machine learning algorithms regarding accuracy and performance [14]; *(iv)* the sheer amount of traffic data that makes it difficult to handle and to promptly detect malicious activities [15] [10]; *(v)* the availability of a high amount of resources, such as management systems and middleboxes, to collect traffic information [16]. Further discussion on machine learning and its use for network traffic classification is presented by Nguyen and Armitage [12].

Further, techniques based on Information Theory have also been used in traditional networks for anomaly traffic detection [15] [17]. These techniques use probability and statistic theory to model the entropy, *i.e.*, the mean information present in some set of traffic features, to detect when disturbances occur in the network. In particular, entropy can be used to model a high-level view of the flows observed in the network, and enables the monitoring of distributions of flow features with reduced computational cost. By analyzing the entropy information within a time interval it is possible to detect deviations that indicate an anomaly. Past research efforts indicate that entropy is a suitable, low-cost, and accurate technique to monitor traffic behavior changes [5]. Moreover, the combination of entropy and machine learning can be used for traffic classification [18].

Recent research efforts have indicated that SDN is suitable for the implementation of sophisticated software solutions and that anomaly detection schemes can benefit from the SDN architecture [11] [19] [20]. However, different than these works, we propose a framework that jointly coordinates anomaly detection, classification and mitigation tasks. SDN can assist to overcome legacy challenges related to anomaly detection, classification and mitigation because its software abstractions enhance the network visibility and management [21]. More concretely, *(i)* the OpenFlow protocol can natively collect primitive traffic statistics about traffic flows; and *(ii)* the network controller can be aware of network topology, traffic profiles, and forwarding behavior without relying on middleboxes. Our work is encouraged by the lack of an integrated framework that combines and manages a large set of techniques related to detection, classification and mitigation of network anomalies.

III. A framework for anomaly detection, classification and mitigation in SDN

In this paper, we advocate that anomaly detection and traffic classification can take advantage of SDN/OpenFlow characteristics. To demonstrate this, we introduce ATLANTIC, an anomaly detection, classification and mitigation framework that allows an administrator to flexibly reconfigure the operation of its building-block components and algorithms. In this section, we discuss the requirements of our framework and explain the main principles behind it. In addition, we present ATLANTIC in details and describe its main components. In particular, we show an overview of our traffic classification process and discuss our proposed architecture.

A. Framework Requirements

A framework for anomaly detection and traffic classification should be capable of orchestrating several different modules, such as those responsible for traffic monitoring, classification, and mitigation. We argue that the required functionality for such a framework should be placed in the management plane of the SDN architecture, and take into account the following aspects:

- **Comprehensive view of the network -** To perform traffic monitoring and analysis, the framework must be able to retrieve detailed and unrestricted information about the network and traffic flows. As opposed to applications sitting in the application plane of SDN – which make use of the Northbound API to request network resources to the controller – our framework uses the Management Interface to gain access to information about flows from all applications, and uses this information to manage traffic anomalies.
- **Human intervention -** The network administrator must be able to interact and monitor the operation of the anomaly detection and traffic classification framework, observing logs and reconfiguring its operation whenever necessary. For example, an administrator might update parameters or replace some component functionality to increase performance or accuracy of classification.
- **Flexible network configuration -** Several types of configurations can assist the task of anomaly mitigation, such as the definition of proactive and reactive path instantiation, deployment of specific or generic

flow rules in flow tables, and management of flow parameters such as timeout and data rate. Our framework must be able to instruct the network controller to change its behavior regarding certain events and flows as they are deemed anomalous.

We anticipate that our management framework must be modular and support customization, *i.e.*, it should be possible to update components with a more sophisticated algorithm or strategy whenever needed. For example, a *network driver* (see Section III-C) may need to be customized to collect information from different types of individual network controllers or from a large set of distributed controllers. Using the management plane, these decisions can be taken by administrators to achieve more appropriate configurations.

B. Lightweight and Heavyweight Processing

ATLANTIC comprises two operational phases: a *lightweight processing phase*, responsible for traffic monitoring and anomaly detection; and a more *heavyweight processing phase*, consisting of anomaly classification and mitigation. Next, we explain these phases in details and how they are combined to support robust anomaly management. Figure 1 summarizes the interplay between the lightweight and heavyweight processing phases.

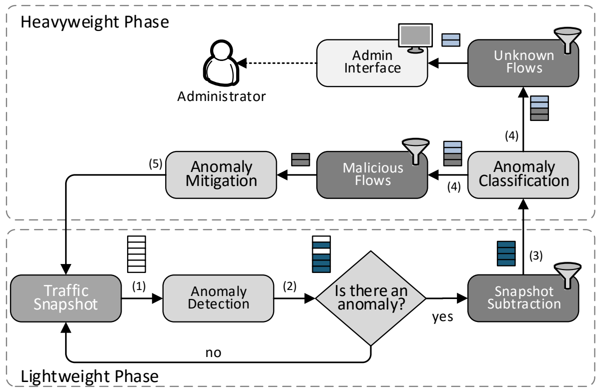


Fig. 1. Framework management process

1) Lightweight Processing Phase : Several techniques can be used to extract network traffic profiles, for example, by performing packet sampling using sFlow [17]. A limitation of this approach is the high memory consumption to obtain fine-grained traffic information and packet inspection. To rapidly perform lightweight anomaly detection, our framework benefits from the characteristics of SDN and uses the control plane to obtain a snapshot of existing traffic flows, including information about traffic counters and matched packet headers. Based on the traffic snapshot collected (arrow 1 in Figure 1), lightweight anomaly detection mechanisms are employed to detect deviations from the "normal" traffic pattern.

We apply entropy analysis to detect variations in the distribution of certain flow features observed along consecutive traffic snapshots. For example, consider two consecutive snapshots t_1 and t_2. If the entropy of the flow features in t_1 is approximately equal to the entropy of the same features observed in t_2, then it is safe to assume that no significant traffic changes have occurred between the two consecutive snapshots. However, if there is a large difference in the entropy calculated for a given flow feature between two snapshots, this might indicate an anomaly (arrow 2). If by subtracting the flows in t_2 from the flows in t_1 we obtain a non-empty result, this indicates which flows are responsible for the entropy change. Flows that are responsible for the entropy change in this stage can only be considered suspicious, and are thus selected for further categorization using a proper classification scheme (arrow 3). It is important to emphasize that our framework is designed so that any snapshot-based anomaly detection scheme can be employed. We chose to use entropy analysis for the *Lightweight Processing Phase* because it can be executed very often and permits fast detection of disturbances in the network [17].

2) Heavyweight Processing Phase: Our framework comprises the occasional need to execute complementary heavy processing classification mechanisms to categorize traffic flows. In this paper, we consider traffic classification mechanisms based on machine learning, which indeed take a considerable amount of resources to execute but can produce very accurate results in terms of traffic classification [12]. Our framework allows both supervised and unsupervised machine learning mechanisms. With supervised mechanisms, a model is generated to internally organize data obtained from previous malicious activity to automatically categorize traffic flows as either malicious or benign. Unsupervised mechanisms, on the other hand, are interesting to be used to analyze and organize information about traffic features, even if they cannot identify whether there is a threat or not. As a result, the framework allows flows to be categorized into either malicious, benign, or unknown, according to their traffic profiles. Malicious flows are sent for mitigation, whereas unknown flows need to be manually analyzed by a human administrator (arrow 4).

For every flow that is signalized as malicious, an action must be taken so to avoid network disruption or performance degradation. For example, commands can be sent back to the network control plane to instruct the devices closer to the source of the malicious traffic to drop packets of flows deemed malicious. After mitigation actions are performed, the framework returns to its initial traffic monitoring and snapshot collection step (arrow 5). For flows signalized as unknown, the administrator can use the information obtained during classification, for example, to create new models for the supervised mechanisms to identify the new traffic pattern in a future round of anomaly detection. Note that this heavyweight phase is expected to be executed less frequently than the lightweight one. In addition, heavy classification mechanisms only need to deal with a subset of the full traffic snapshot, because of the subtraction performed in the lightweight phase. The more the administrator interacts with the framework inserting new information about traffic patterns to be automatically identified, the more efficient the detection will be.

C. Anomaly Traffic Classification

The basic components of our anomaly traffic classification framework are depicted in Figure 2. The *Statistical Layer* is responsible for collecting traffic flow statistics and comprises the following components: *Statistics Manager*, *Features Selector*, and *Network Driver*. The information generated by the *Statistical Layer* is delivered to the *Classification Layer*, which comprises the following components: *Anomaly Monitor*, *Flow Classifier*, and *Flow Manager*. Next, we describe these components in details.

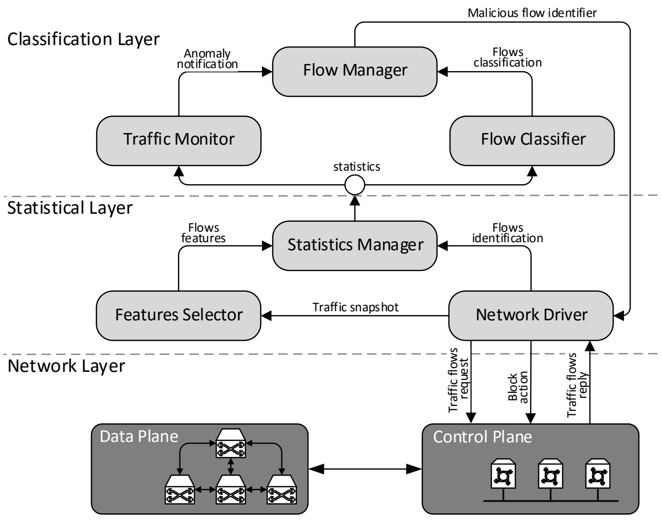


Fig. 2. Overview of the anomaly classification framework

1) Network Driver: The Network Driver operates by sending a request to the network controller every p seconds in order to query the status of all flows in the data plane. The parameter p can be adjusted accordingly, to avoid degrading network performance. After receiving flow information from the network controller, the *Network Driver* parses and organizes relevant data, such as the flow identifier, packet headers, and flow counters. The result of one request produces a traffic snapshot, *i.e.*, a data structure summarizing all flows currently existing in the network. Note that we construct a *flow-id* using the following 5-tuple, which is also used as our flow definition:

$$\langle srcip, dstip, srcport, dstport, protocol\rangle$$

2) Features Selector: After receiving the last saved traffic snapshot produced by the *Network Driver*, the *Features Selector* extracts basic flow features that describe the profile of network flows. The basic set of features is defined by the OpenFlow specification [4] and is represented by:

$$\langle packet\_count, byte\_count, duration\rangle$$

This component initiates the selection of relevant features. Frequently, the best choice for a flow profile model is a set of features that can accurately be used to discriminate a flow class with minimum computational cost. The issue of finding the optimal set of features to describe a flow can be addressed by manual inspection, or with the assistance of techniques such as PCA (*Principal Component Analysis*) and genetic algorithms [12]. This component can benefit from extensions to the controller API, which may enable additional information about flows to be exported.

3) Statistics Manager: The Statistics Manager summarizes all data collected by the *Network Driver* and by the *Features Selector* in order to derive statistical information that is used by the classification algorithms. In general, raw features have limited contribution to classification schemes without post-processing and statistical interpretation. After the features for one traffic snapshot are obtained, the *Statistics Manager* calculates *(i)* mean, *(ii)* standard deviation, *(iii)* coefficient of variance, and *(iv)* minimum and *(v)* maximum values as statistical discriminators to describe the data collected for each flow.

4) Anomaly Monitor: The main objective of this component is to monitor traffic flows using their statistics and basic flow features to detect potential anomalies. To exemplify how this component can behave, we use entropy analysis to detect changes in traffic features. In particular, this is calculated according to *Shannon's entropy* definition. Considering that a traffic snapshot is an alphabet, then the mean information $H(X)$ for some subset of features can be calculated using the available flows. We chose to calculate the entropy based on IP address and transport port features because they have been demonstrated to be accurate for the detection of DDoS attacks and worm propagations [11]. Every time that a new entropy E is calculated for a given snapshot, it can be classified as anomalous in the following way: considering that M represents the mean entropy observed in the network and S the standard deviation associated, then E will be an anomalous entropy if it is not within the interval $[M - S, M + S]$.

5) Flow Classifier: Different algorithms can be used for flows classification. When this component receives a set of feature statistics, a range of classification schemes (*e.g.*, machine learning algorithms) can run independently over the flow features. Note that this component is responsible for defining the class of each specific flow, by deciding which class is the most frequent when considering all algorithm outputs. Currently, we apply K-means [8] for clustering and Support Vector Machine (SVM) [22] for classification. We consider these algorithms suitable to initially exemplify our classifier because one can complement the results offered by the other, and the union of their outputs can be easily performed. Still, the *Flow Classifier* is customizable and can be extended with additional algorithms[2].

6) Flow Manager: Events from the *Anomaly Monitor* and *Flow Classifier* are sent to the *Flow Manager* to indicate if an anomaly has been identified for a specific *flow-id*. Thus, the *Flow Manager* is responsible for deciding the mitigation actions to be taken when a malicious flow is identified. In this paper, we consider that a *'Malicious flow identifier'* message is sent to the *Network Driver*, indicating that the flow identified as malicious should be blocked. The *Network Driver* component further uses the *'Block action'* message to install firewall rules in the data plane and then modify how this plane handles the malicious flow. An example of action that could be taken by the *Network Driver*, as an alternative to dropping packets associated with malicious flows, is to forward such packets to another component (*e.g.*, a deep-packet inspector).

[2]In our implementation, the training phase for SVM is performed using simulated attack traces ($\sim$100 malicious flows) in order to provide a range of samples to the classification schemes. Also, we use the R tool (https://www.r-project.org) to provide a reliable implementation of such machine learning algorithms.

IV. Framework Evaluation

In this section, we present an experimental evaluation of ATLANTIC. The experiments were divided in two scenarios featuring different attacks: a worm propagation and a large-scale DDoS attack. We analyzed the amount of memory used by ATLANTIC and the processing time needed to provide anomaly detection, classification and mitigation. Our evaluation includes the study of *(i)* the performance of the lightweight phase; *(ii)* the performance of the heavyweight phase; and *(iii)* the classification accuracy of the overall framework.

A. Testbed and Simulation Profile

ATLANTIC was implemented on top of the Floodlight controller[3]. Floodlight provides a JSON-based REST API that is suitable for implementing the communication between the network controller and ATLANTIC. We run the experiments using the Mininet emulator. We used a topology of a campus network. It is a partially mesh topology consisting of 100 hosts and 11 switches. We chose this scenario because of recent malicious attempts to attack similar environments [23].

Table I describes the background traffic used in our experiments. When the simulation is set up, two services are configured: a video streaming server (following a lognormal distribution) and a Web server (following an exponential distribution) [24]. These services enable hosts to receive streaming over HTTP or send requests for Web pages, thus generating traffic related to file transfer. These traffic profiles have been validated in [24]. We generate these traffic profiles using VLC[4] for video streaming and SimpleHttpServer[5] to emulate an HTTP server.

TABLE I. Background traffic profile used in the experiments

Parameter	Value
Number of hosts	100
Number of switches	11
Number of servers	2 (HTTP and Streaming)
Number of attack flows	3500
Traffic profile	Video: 75 %, Web: 25 %
Host behavior	Exponential Distribution ($\lambda = 0.033$,mean = 30 s)

In order to simulate the user behavior, we set up a scenario where users watch a video for a certain amount of time, pause it, and then access a few Web pages. For each group of 6 hosts requesting HTTP traffic, there is 1 host requesting video streaming traffic. Network anomalies related to malicious activities are generated with the *scapy* tool,[6] which enables the generation of realistic malicious attacks, such as port scanning and DDoS. The importance of these attacks has increased thanks to recent uses of DDoS to compromise campus communications [23]. Next, we explain the behavior of our simulated attacks.

- **Port scanning -** A malicious host can use port scanning to discover a set of open ports in a remote host. Open ports can be used to exploit vulnerabilities in a target system or be used in worm propagations [25]. We simulate a malicious user that chooses a random host to start its attack to a server in the network. The attack consists of sending several TCP connection packets to ports ranging from 0 to 65536. When an open port is found, a notification is generated and this port can be used for worm propagations. Typically, *port scanning* generates packets in the network with a fixed IP address, but with varying transport protocol port.
- **DDoS attack -** We also defined a *DDoS attack* [11] scenario. A DDoS attack in SDN can be used to overflow with a large amount of fake flows a specific switch's flow table or to overload the network controller by producing several `packet_in` messages. Frequently, these attacks result in a range of source IP addresses accessing a single target IP. In this case-study, we create a SYN flood attack, *i.e.*, a malicious machine chooses a server (HTTP or streaming) to send multiple TCP SYN packets to service ports offered by this machine (8080 for the VLC streaming and 8000 for the HTTP server). Given that there are services running in those ports, the server will process a request, allocate resources to handle it, and send an ACK to the requesting machine. Afterwards, the attacker uses source IP spoofing, *i.e.*, it sends TCP SYN packets with the source IP address of another machine, thus causing the erroneous receipt of an unsolicited ACK and leaving the server with an open TCP connection.

B. Lightweight Anomaly Detection Evaluation

In this section we demonstrate the operation of the lightweight phase when it is instantiated with an entropy-based anomaly detection scheme to monitor the network in the presence of malicious anomalies. Additionally, we analyze the performance of this phase regarding memory usage and processing time.

Initially, the *Network Driver* communicates with the controller to request traffic information. There are two possible bottlenecks in this approach: the traffic snapshot transmission time and the amount of memory needed to store this information. To understand these issues, we monitored the number of flows generated in the network while users were accessing HTTP pages and video streaming during a few minutes. Next, we started a DDoS attack and monitored the amount of new traffic flows. The transmission time required to export this information to our framework and the amount of memory needed were observed. The polling interval was set to 0.5s in order to obtain a fine-grained view of the network traffic. According to Figure 3(a), the transmission time related to traffic snapshot when an attack is not happening remains under 1.07s. Around the 180^{th} snapshot, the DDoS attack starts and increases this transmission time to 1.28s in the worst case. Furthermore, according to Figure 3(b), the size of network snapshots increase from 32 to 600 kilobytes with approximately 4,400 flows, including malicious and benign.

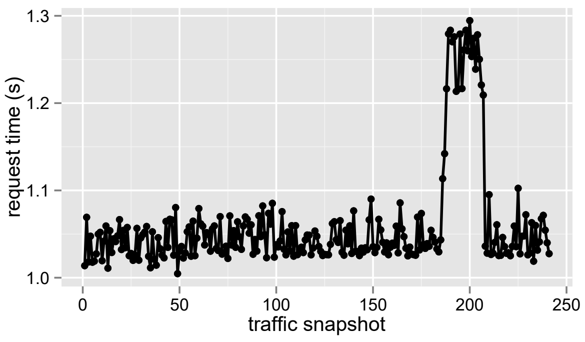
We argue that our simulations can realistically reflect the size of a large-scale campus scenario. In particular, it has been shown in [24] that campus scenarios comprise around 800 traffic flows in mean. We also analyzed individual flow rules

[3]http://www.projectfloodlight.org/

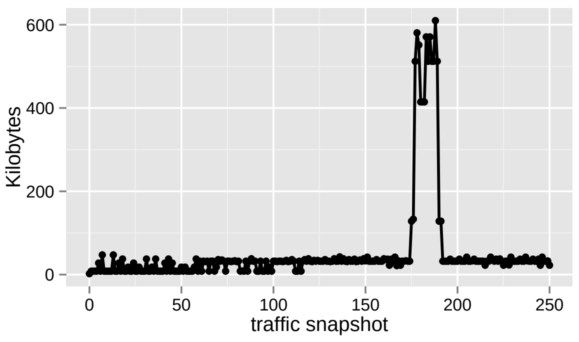
[4]http://www.videolan.org/vlc/

[5]https://docs.python.org/2/library/simplehttpserver.html

[6]http://www.secdev.org/projects/scapy/



(a) Traffic snapshot requesting time



(b) Traffic snapshot memory used

Fig. 3. Resources usage to request and store a traffic snapshot

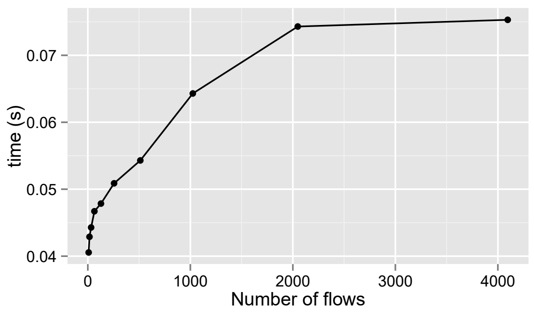


Fig. 4. Processing time of entropy calculation

in ATLANTIC and verified that a single flow rule uses around 136 bytes and takes less than 0.0008s to be transmitted. Thus, in order to simulate the campus scenario presented in [24], ATLANTIC would use 108,800 bytes (106.25 Kb) to store a traffic snapshot and 0.64s to transmit this information.

Entropy mean values associated with traffic features[7] can be used to detect network anomalies. The performance of the *Flow Monitor* itself is related to entropy calculation time, which grows linearly accordingly to the number of flows in a specific traffic snapshot. Figure 4 indicates that the entropy calculation delay for 4,000 flows is 0.075s. We conclude that this type of traffic monitoring is suitable for ATLANTIC mainly because it is fast and accurate for detecting traffic deviations when we analyze, for example, the source IP entropy of a traffic snapshot. Figure 5(a) illustrates the variation in entropy of the destination port when a port scanning attack occurs. Around the 180^{th} snapshot, the average entropy that was around 0.55 rises to almost 1, indicating an anomaly. It is still possible to observe in Figure 5(b) the changes in the entropy caused by the DDoS attack. Around the 280^{th} simulation snapshot, the DDoS attack stops, thus causing the entropy to revert back to normal (between 0.8 and 0.9).

When changes in the entropy of a specific feature are detected, the *Flow Classifier* component needs to determine the nature of existing flows in the network. The performance of this component is discussed in the next sub-section.

C. Heavyweight Anomaly Classification Evaluation

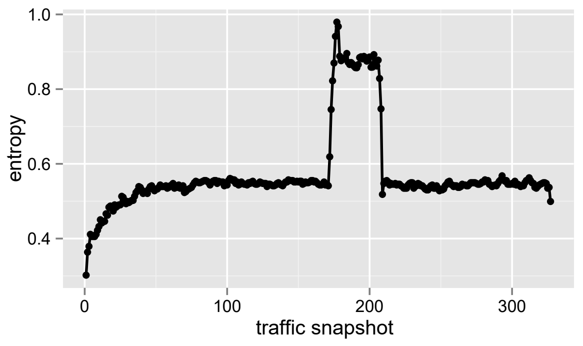
Each time a traffic deviation is detected, a notification is generated and the ATLANTIC framework enters into its heavyweight phase. The first action taken by the *Flow Classifier* is the classification of all remaining flows after the subtraction of the current traffic snapshot from the last one. Due to space limitations, we only show the results for the DDoS scenario.

To classify flows into separate classes representing DDoS and normal traffic, we instantiate the *Flow Classifier* with two well-known algorithms: K-means and SVM. Using K-means, similar flows are clustered together such that each cluster represents a type of traffic profile observed in the network. After this, the SVM algorithm can determine the classification of flows in each cluster. Note that each traffic snapshot may contain HTTP and streaming flows with different profiles, such as short-lived HTTP flows and long term video streaming flows. Figure 6(a) and Figure 6(b) summarize the amount of active flows in the simulation, as well as the flows signalized as malicious and subsequently blocked. Based on the joint classification offered by K-means and SVM, it is possible to calculate metrics such as *precision (PPV)* and *accuracy (ACC)*, which allow assessing the quality of the classification achieved by these algorithms[8]. In particular, it is possible to apply K-means using different values of k in order to find the optimal configuration and, jointly with SVM, find which combination achieves better classification metrics. Figure 7 illustrates our results. It can be observed that SVM presents accuracy of 88.7% and precision of 82.3%. The simulations were executed 35 times until the error rate was less than 0.01. The results obtained are very similar to the values expected from traditional networks, as presented in [26].

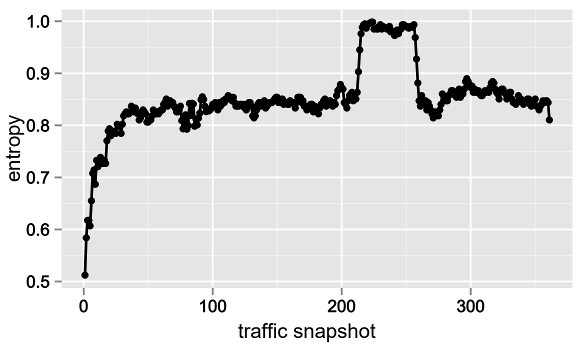
After obtaining the classification, the *Flow Manager* is notified and then it is able to block malicious flows and restore the entropy back to normal. To block these malicious occurrences in the network, the *Flow Manager* can instruct

[7]We performed entropy analysis on all the features of a flow, but due to space constraints we only show results for destination ports and destination IP addresses in the first and second case-study, respectively.

[8]We computed the standard metrics commonly used in the evaluation of machine learning algorithms – TPR: sensitivity or true positive rate; SPC: specificity or true negative rate; NPV: negative predictive value; FPR: false positive rate; FDR: false discovery rate; F1: f1 score (harmonic mean of precision and sensitivity)

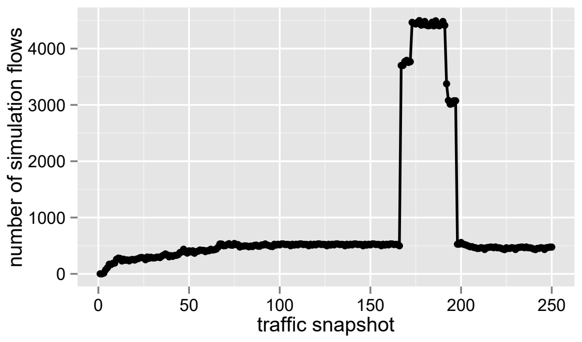


(a) Port scanning entropy

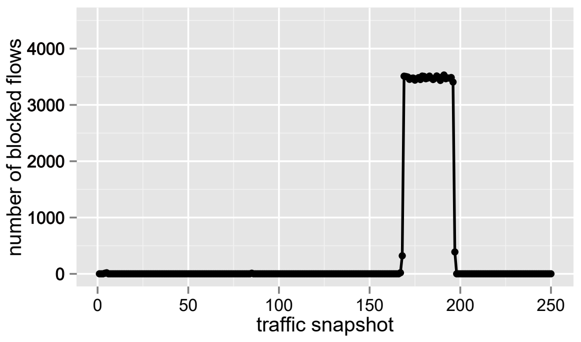


(b) DDoS entropy

Fig. 5. Entropy observed including benign and malicious flow



(a) Overall flows



(b) Blocked flows

Fig. 6. Number of simulation flows

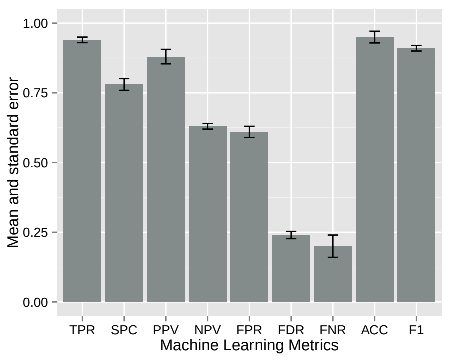


Fig. 7. Machine Learning metrics for SVM

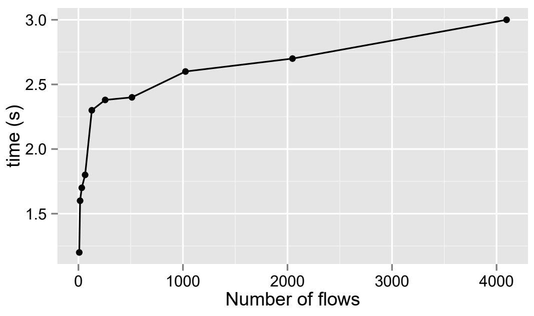


Fig. 8. Processing time of Heavyweight Phase

the *Network Driver* to use a firewall application or to use the OpenFlow drop action installed on the data plane. This solution can block external malicious attacks from other networks.

The heavyweight phase lasts around 3 seconds in our simulations (Figure 8). When we compare this with the processing time of the lightweight phase (which takes around 0.07 seconds), we can more clearly appreciate the benefits of using more frequently the lightweight phase instead of always using the heavyweight phase to classify every traffic snapshot.

D. Discussion

We advocate that ATLANTIC meets the management requirements listed in Section IV.A. The lightweight step performs traffic monitoring using a global network view, demonstrating that detailed traffic information can be easily obtained. This contributes toward the comprehensive network view requirement needed for accurate anomaly detection.

The design of ATLANTIC allows the network administrator to monitor and modify the operation of its components. This characteristic contributes to a more tailored anomaly classification, which can rely on human intervention when the automated components are not able to protect the network.

Additionally, using the network controller, ATLANTIC can orchestrate all flows in the network, sampling or eventually blocking a particular flow whenever needed.

V. Conclusions and Future Work

This paper proposed ATLANTIC, a framework for anomaly detection, classification and mitigation in SDN-based networks. Our framework comprises a lightweight phase responsible for monitoring traffic flows and a heavyweight phase responsible for anomaly classification and mitigation. As a result, traffic anomalies can be categorized and the information collected can be used to handle each traffic profile in a specific manner, such as blocking malicious flows.

In our experiments, the lightweight monitoring scheme enabled ATLANTIC to detect malicious activities without overloading the network, taking about 0.075s to collect and analyze traffic information consisting of 4400 flows in a topology with 100 switches. The heavyweight phase uses a machine learning algorithm (*i.e.*, SVM) which took less than 3s to classify traffic flows, demonstrating that ATLANTIC performs well even in the presence of DDoS attacks. Moreover, ATLANTIC executes the lightweight phase more frequently than the heavyweight phase, thus minimizing the overhead of the overall anomaly detection scheme. Most importantly, our results show how a sophisticated anomaly detection framework can be built over SDN.

As part of our future work, we aim to evaluate the use of different algorithms for traffic classification and entropy analysis to enforce network protection. We intend to investigate new mitigation strategies, such as the use of rate limiters, and new classification schemes, such as the combination of several network classifiers using meta-learning techniques (*e.g.*, stacking, and bayesian networks).

References

[1] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and Survivability in Communication Networks: Strategies, Principles, and Survey of Disciplines," *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, Jun. 2010.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[3] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, February 2013.

[4] Open Networking Foundation, "OpenFlow Switch Specification - Version 1.0.0 (Wire Protocol 0x01)," December 31 2009, available at: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Accessed: August 2014.

[5] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, and H. Zhang, "An Empirical Evaluation of Entropy-based Traffic Anomaly Detection," in *8th ACM SIGCOMM Conference on Internet Measurement (IMC)*, New York, USA, 2008, pp. 151–156.

[6] R. Yuan, Z. Li, X. Guan, and L. Xu, "An SVM-based Machine Learning Method for Accurate Internet Traffic Classification," *Information Systems Frontiers*, vol. 12, no. 2, pp. 149–156, Apr. 2010.

[7] T. Ide, S. Papadimitriou, and M. Vlachos, "Computing Correlation Anomaly Scores Using Stochastic Nearest Neighbors," in *7th IEEE International Conference on Data Mining (ICDM)*, 2007, pp. 523–528.

[8] J. Erman, M. Arlitt, and A. Mahanti, "Traffic Classification Using Clustering Algorithms," in *SIGCOMM Workshop on Mining Network Data (MineNet)*, New York, NY, USA, 2006, pp. 281–286.

[9] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet Traffic Classification Demystified: Myths, Caveats, and the Best Practices," in *ACM CoNEXT*, Madrid, Spain, 2008, pp. 11:1–12.

[10] W. Wang and S. Gombault, "Efficient Detection of DDoS attacks with Important Attributes," in *3rd International Conference on Risks and Security of Internet and Systems (CRiSIS)*, Oct 2008, pp. 61–67.

[11] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow," in *IEEE 35th Conference on Local Computer Networks (LCN)*, Washington, DC, USA, 2010, pp. 408–415.

[12] T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[13] A. W. Moore and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques," in *ACM SIGMETRICS*, New York, NY, USA, 2005, pp. 50–60.

[14] N. Williams, S. Zander, and G. Armitage, "A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 5–16, Oct. 2006.

[15] A. Lakhina, M. Crovella, and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," in *ACM SIGCOMM*, New York, NY, USA, 2005, pp. 217–228.

[16] Y. Yu, M. Fry, A. Schaeffer-Filho, P. Smith, and D. Hutchison, "An Adaptive Approach to Network Resilience: Evolving Challenge Detection and Mitigation," in *8th International Workshop on the Design of Reliable Communication Networks (DRCN)*, 2011, pp. 172–179.

[17] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Computer Networks*, vol. 62, pp. 122 – 136, 2014.

[18] B. Agarwal and N. Mittal, "Hybrid Approach for Detection of Anomaly Network Traffic using Data Mining Techniques," *Procedia Technology*, vol. 6, pp. 996 – 1003, 2012, 2nd International Conference on Communication, Computing and Security (ICCCS).

[19] Y. Zhang, "An Adaptive Flow Counting Method for Anomaly Detection in SDN," in *9th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '13. New York, USA: ACM, 2013, pp. 25–30.

[20] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in Internet Society NDSS, 2013.

[21] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting Traffic Anomaly Detection Using Software Defined Networking," in *14th International Conference on Recent Advances in Intrusion Detection (RAID)*, Berlin, 2011, pp. 161–180.

[22] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[23] J. J. Santanna, R. van Rijswijk-Deij, A. Sperotto, R. Hofstede, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - An Analysis of DDoS-as-a-Service Attacks," in *14th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, Canada, May 2015, pp. 243 – 251.

[24] P. H. Isolani, J. A. Wickboldt, C. B. Both, J. Rochol, and L. Z. Granville, "Interactive Monitoring, Visualization, and Configuration of OpenFlow-based SDN," in *14th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ottawa, Canada, May 2015, pp. 207–215.

[25] A. Schaeffer-Filho, A. Mauthe, D. Hutchison, P. Smith, Y. Yu, and M. Fry, "PReSET: A Toolset for the Evaluation of Network Resilience Strategies," in *13th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Ghent, Belgium, May 2013, pp. 202–209.

[26] X. Li, F. Qi, D. Xu, and X. Qiu, "An Internet Traffic Classification Method Based on Semi-Supervised Support Vector Machine," in *IEEE International Conference on Communications (ICC)*, Kyoto, Japan, June 2011, pp. 1–5.