Journal of Reliable Intelligent Environments
https://doi.org/10.1007/s40860-022-00171-8

REVIEW

# A comprehensive survey on SDN security: threats, mitigations, and future directions

Yassine Maleh[1] · Youssef Qasmaoui[2] · Khalid El Gholami[1] · Yassine Sadqi[3] · Soufyane Mounir[1]

Received: 9 October 2021 / Accepted: 20 January 2022
© The Author(s), under exclusive licence to Springer Nature Switzerland AG 2022

**Abstract**
Nowadays, security threats on Software Defined Network SDN architectures are similar to traditional networks. However, the profile of these threats changes with SDN. For example, a denial-of-service attack on a centralized controller that manages a large network of several network devices (routers, switches, etc.) is more destructive than a targeted attack against a router. A spoofed SDN controller could allow a hacker to control an entire network, while a spoofed router could only harm the proper functioning of the traffic routed through that router. The SDN is facing these new security challenges, especially on securing the SDN architecture itself. SDN security is ensured at all these levels based on three-layer architecture and programming interfaces, which poses several challenges. The SDN's security challenges are expected to grow with the progressive deployment. This paper aims to provide a comprehensive review of state of the art, accompanied by categorizing the research literature into a taxonomy that highlights each proposal's main characteristics and contributions to the SDN's different layers. Based on the analysis of existing work, we also highlight key research gaps that could support future research in this area.

**Keywords** Software Defined Network · SDN threats · SDN attacks · Control plane · Data plane · Application plane · SDN mitigations

## 1 Introduction

Today, the Internet, or so-called computer networks, is a key component of businesses, schools, and homes. The Internet connects firms, knowledge and families on a global scale. However, over the last 20 years of traditional computer networks, we have seen the emergence of limitations such as dependence on unchanged protocols, distributed control, hardware that is difficult to reprogram [130], complex and unwieldy networks [128]. The need for new devices with networking capabilities and the Internet of Things (sensors, cameras, alarms, among others) brings challenges to network management. In this sense, the concept of Software-Defined Networking (SDN) emerged [153, 154], a new paradigm for networks that makes it possible to break the ossification barrier. This concept allows this vertical integration to be broken, removing the responsibility from network equipment (such as routers and switches) in controlling the network, leaving only the task of managing the routing of data on the network. In the SDN concept, network control is no longer distributed among each network device, but logically centralized, making network control manageable via software, and thus acquires the ability to program the network. In this way, the responsibilities can be organized in plans, where the management of the network with a high level of abstraction is the responsibility of the management plane (management via software), the concretization of this management in terms of rules with a low level of abstraction for the instruction of the network

✉ Yassine Maleh
yassine.maleh@ieee.org

Youssef Qasmaoui
youssef.qasmaoui@ieee.org

Khalid El Gholami
k_elgholami@usms.ma

Yassine Sadqi
y.sadqi@usms.ma

Soufyane Mounir
s.mounir@usms.ma

[1] Laboratory laSTI, ENSAK, USMS University, Beni Mellal, Morocco

[2] Laboratory IR2M, FSTS, UH1 University, Settat, Morocco

[3] Laboratory LIMATI, FPBM, USMS University, Beni Mellal, Morocco

Published online: 08 February 2022

equipment is the responsibility of the control plane, and the routing of the data that travels on the network is the responsibility of the data plan. In a classical network model, these three planes are associated with the network equipment (also called vertical integration), but these planes are disassociated (known as vertical integration breakdown). This allows the separation of network policy definitions and their respective implementations on network devices. SDN has attracted significant attention from academia and industry. For example, industry professionals have recently created the Open Network Foundation, an industry-driven organization to promote SDN and define standardizations on the topic. Another initiative, but from an academic perspective, comes from creating the Open-Flow Network Research Center, focusing on SDN-related research.

This paradigm shift (traditional network approach vs. SDN) has generated a spectrum of new challenges (e.g. how to make these paradigms compatible, how to provide the SDN paradigm in the most diverse networking scenarios that exist today, etc.), as well as bringing back several problems found in traditional networks, such as performance, resilience, scalability, reliability and security.

With SDN, network intelligence is externalized and managed by an external device called a "controller" that manages the functions of the control plane. This intelligent controller entity can be on one or more distributed physical or virtual machines. The network configuration will be done by programming the controller via an open Application Programming Interface (API) called Northbound API. The communication between the controller and the network devices is done via a secure channel through the Southbound API (e.g., OpenFlow).

The enthusiasm of digital players, such as Google and Microsoft in deploying SDN in their data centers, gives great prospects to the concept of SDN, which is starting to become a reality.

In other words, the goal of SDN is to enable networks to be agile, flexible and programmable to make them easy to manage. The SDN concept is now globally recognized as an architecture that opens up networks to applications and offers many advantages. Despite its advantages, the SDN faces several threats, such as the denial of service (DoS) attack, considered the primary threat to these networks. Because of the centralized management in the SDN architecture, this type of attack can easily overload the switches' SDN controller and switch tables (TCAMs), resulting in a critical degradation of network performance. Research topics based on traditional security techniques are numerous to protect networks from different attacks. Still, techniques using the SDN paradigm to improve security are new, promising and widely discussed in the literature. Hence, it has become necessary to analyze the recent SDN security threats and propose an up-to-date and detailed classification, which considers current and future threats related to the three layers of SDN [16].

The main goal of this work is to provide readers with a comprehensive revision of state-of-art proposals for the development and evolution of SDN security. This paper mainly aims to make the following contributions:

- It analyses the SDN security threats.
- It studies and analyses the SDN security mitigations and solutions for the different SDN layers in-depth.
- An introduction to the open and emerging challenges in SDN security to suggest and point out the directions for future research efforts and proposals.

The structure of this paper is as follows. Section 2 describes the research methodology. Section 3 presents the survey methodology. Section 4 reviews the related SDN threats and vulnerabilities. Section 5 presents the most important SDN mitigation techniques for each SDN Layer classification. Section 6 discusses and compares the literature review. Section 7 proposes some future open directions. Finally, Sect. 8 concludes the paper.

## 2 Background and context

The concept of Software-Defined Networking (SDN) has emerged as a new networking paradigm that makes it possible to break the ossification barrier. This concept allows the breaking of this vertical integration, taking away the responsibility of network equipment (such as routers and switches) in controlling the network, leaving only the task of managing the routing of data on the network. In the SDN concept, network control is no longer distributed among each network device, but logically centralized, which makes network control manageable via software, and thus acquires the ability to program the network [63–65]. The responsibilities can be organized in plans, where the management of the network with a high level of abstraction is the responsibility of the control plane. The concretization of this management in terms of rules with a low level of abstraction for instructing the network equipment is the responsibility of the control plane, and the routing of data that travels on the network is the responsibility of the data plane. In a classical network model, these three planes are associated with the network equipment (vertical integration). Still, these planes are disassociated (known as vertical integration breakdown) [25]. This allows the separation of network policy definitions and their respective implementations on network devices. SDN has attracted significant attention from academia and industry. For example, industry professionals have recently created the Open Network Foundation [151], an industry-driven organization promoting SDN and defining standardizations on the

topic. Another initiative, but from an academic perspective, comes from creating the OpenFlow Network Research Center [35, 25, 131], focusing on SDN-related research. This paradigm shift (traditional network approach vs. SDN) has generated a spectrum of new challenges (e.g., how to make these paradigms compatible, provide the SDN paradigm in the most diverse networking scenarios that exist today, etc.). It also brings back several problems in traditional networks, such as performance, resiliency, scalability, reliability, and security. This provides the opportunity for related areas to solve the challenges that SDN imposes. In this sense, security is one area where SDN presents complex open challenges [139].

## 2.1 SDN architecture

Common network infrastructures generally use equipment (switches, routers, etc.) whose configuration depends on the manufacturers. Once deployed, it is challenging to evolve these infrastructures by adopting new protocols or adding and removing equipment. Thus, the idea of Software Defined Networks (SDNs) emerges.

SDN is a new paradigm for defining network equipment's behavior using control software [3]. The data plane establishes the network's equipment and connections, while the control plane defines its behavior. Consequently, SDN separates the data plane and the control plane.

The purpose of SDN technology is to offer some open interfaces that enable the development of software that can manage the connectivity supplied by a collection of network resources and the flow of network traffic through them, along with possible examination and adjustment of traffic that may transit in the network [115, 116]. The ONF (Open Networking Foundation) defined three main principles of SDN [115, 116]:

- Separating the controller and data planes:
- Decouple the control plane from the data plane. However, control must be executed within the data plane system.
- A central control: This aspect will provide a high overview of the network and enhance decision-making instead of local control; therefore, change latency is less significant.
- Exposure of abstract network resources and state to external applications.

These principles are clarified by the SDN architecture built around three principal layers: Application plane layer, Controller plane layer, and Data plane layer.

### 2.1.1 Data plane

This layer defines the functionality of the data plane. The data plane consists of network equipment such as switches, routers, etc., including packet routing components and APIs [95]. A network device is an entity that receives packets in its ports and performs one or more network functions (e.g., a packet received on a port can be forwarded, deleted, or its header modified for a specific action) [148]. Figure 1 Shows the Flow Table Entry architecture.

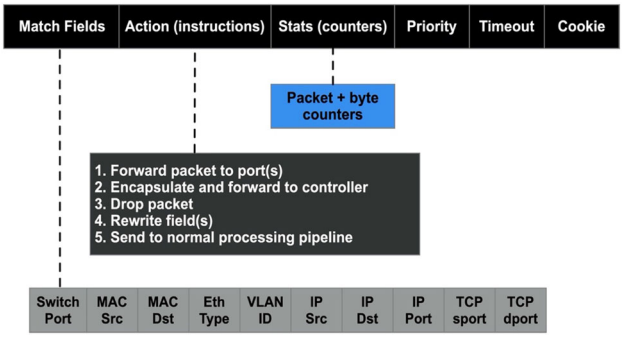


**Fig. 1** Flow table entry architecture

### 2.1.2 Control plane

The control layer of the SDN architecture consists of a software controller called the SDN controller that centralizes all network intelligence. Considered the most important component of the SDN concept, the SDN controller is a software program comparable to a network operating system, responsible for manipulating the flow table of a switch through the Southbound API. In other words, the SDN controller is responsible for managing and configuring the network by installing appropriate packet routing rules on the network devices (switches, routers) at the infrastructure layer through the Southbound API. This programming interface is crucial, because it offers the possibility of innovations for programmers and accelerates the development and implementation of network services. OpenFlow is the most widely used southbound. It allows the SDN controller to add, modify, and delete entries in the forwarding table of an OpenFlow switch. A secure channel connects the controller to an OpenFlow switch. Using this channel as a connection interface, the controller manages the OpenFlow switches, receives packets from the OpenFlow switches and sends packets to the OpenFlow switches.

Several controllers have been developed, most of which are open source and support the OpenFlow protocol. These controllers differ in their programming languages, the version of OpenFlow supported, the techniques used such as multi-threading, and the performance such as throughput. Below is a non-exhaustive list of SDN controllers:

- Nox [53]: the first publicly available controller written in C language, the use of a The only thread in its core limits its

deployment. Several NOX bypasses have been later proposed, notably the Nox in its multithreaded version called NOX-MT [136].

- QNOX [65], NOX supporting Quality of Service (QoS), FortNOX [105], an extension of NOX implementing a conflict detection analyzer in the flow rules caused by applications, and finally, the POX Controller [107], a NOX-based controller in python language, whose purpose is to improve the performance of the original NOX controller.
- Maestro (Maestro, n.d.): uses multithreading technology to perform bottom parallelism, keeping a simple programming model for application developers. It achieves its performance by distributing tasks from the core to the threads. Besides, Maestro can process requests from multiple streams through a single runtime task, increasing its efficiency.
- Beacon (Beacon, n.d.): developed in Java by Stanford University, it is also based on multi-threaded technologies and is multiplatform. Its modular architecture allows the manager to run only the services desired.
- SNAC (SNAC, n.d.): uses a web application to manage network rules. A language of Flexible rules definition and easy-to-use interfaces have been integrated to configure network devices and control their events.
- Floodlight (Floodlight, n.d.): is a variant of Beacon characterized by its simplicity and performance. It has been tested with physical and virtual OpenFlow switches. Many developers now supported and improved, including Intel, Cisco, HP, Big switch, and IBM.
- McNettle [139]: it is an SDN controller programmed by Nettle [137], which is a DSL (Domain Specific Language) integrated into Haskell, and allows programming of networks using OpenFlow. McNettle operates in multi-core servers that share their memory to achieve global visibility, high performance, and low latency.
- RISE [59]: designed for large-scale network experiments, RISE is a Trema-based controller (Controller, n.d.). The latter is a framework programmed in Ruby and C. Trema that provides an integrated test and debugging environment, including tools for development.
- Ryu (Ryu, n.d.): is a framework developed in Python. It allows the separation between domains without using a VLAN. It supports up to the latest version of OpenFlow 1.5.
- Opendaylight (OpenDaylight, n.d.): is an open-source project implemented in java and cross-platform. It supports the OSGi Framework [54] for local programming of the controller and REST [44] bidirectional. ConteXteam, IBM, NEC, Cisco, Plexxi, and Ericsson are very active in this project.

#### 2.1.3 Application plane

Service providers and operators can use applications to configure, control, and monitor their network in SDN. It is the highest layer of the SDN architecture where management rules are defined and implemented. These rules are a set of applications such as IDS, firewalls, load balancing… con¸cues to provide a specific service to the network. This layer brings application automation through APIs and allows interaction and manipulation of network devices' behavior through the SDN controller. Thanks to the application layer, applications benefit from the visibility of network resources in a specific way. For example, for discovering links or network topology, it is necessary to make a coupling between the applications and the underlying network devices such as switches and routers.

In the SDN approach, the control plane provides a global view of the network and information about the network elements to the applications via the controller's southbound API. As mentioned earlier, OpenFlow is the de facto standard API for implementing network functions as OpenFlow applications.

Network configuration and management enable real-time configuration and ease of administration, network monitoring, network troubleshooting, network policies, and security. With this layer, from a security perspective, various services can be implemented on top of the OpenFlow controller as security applications. The application can include SDN-specific applications belonging to SDN providers themselves. However, it can also have a third-party application developed to meet specific needs such as applications related to network automation to better align with the needs of the applications running on it. Those applications often present a high-risk rate, specifically third-party applications. The threats generally target the control plane because of their high value. It is mandatory to secure access to the control data provided by the SDN controller.

Figure 2 shows the architecture of the typical SDN network as described by the Open networking foundation.

## 3 Research methodology

This research investigates the following hypothesis: It is possible to balance security and performance in the control plane communication of an SDN environment, between controllers and switches, with a security level. It does not compromise the network's security. The following research questions will be answered to support the hypothesis: 1. What are the main vulnerabilities, attacks, and existing solutions to provide security in SDN environments? This question aims to understand the current security scenario in SDN to provide a

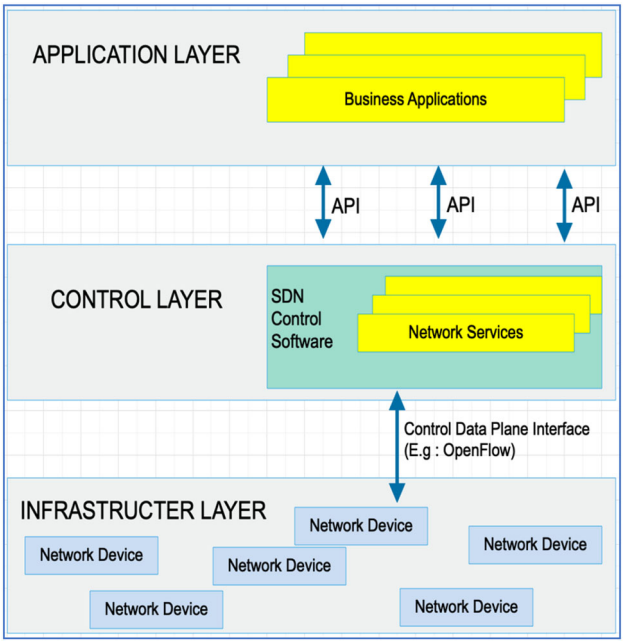


**Fig. 2** SDN Architecture with tree layers (Application plane, Control plane, Data plane)

framework of knowledge on the subject to define the requirements to support the hypothesis. None of the existing studies in the literature on security in SDN used a well-defined methodology to cover the entire scope of works involving the topic. Although several works, which aim to organize the research on the topic, have already been developed [3, 47, 25, 119].

In this section, we present the methodology used in this survey. To assist in developing the research proposal, a systematic mapping study (SMS—Systematic Mapping Study) [102] was conducted on the current state of security in SDN. An SMS is a common methodology to help conduct research, such as in the medical area. Since 2007, its applicability in the Computing area has grown substantially, especially in Software Engineering [15].

According to Petersen [102], an SMS provides a categorization structure, offering a visual summary, mapping, and results. This requires less effort, while providing an overview with a better selection of information.

Initially, SMS was geared towards software engineering and was usually recommended for research areas with a range of relevant and high-quality early-stage studies. As security in SDN is early and high-quality studies are emerging, the SMS can contribute to categorizing these studies.

The SMS was first organized by defining the research questions. It was necessary to think about the opportunities that security in SDN produces. The primary categories related to the topic were defined: vulnerabilities, attacks, and defense mechanisms. To highlight the necessity of creating an up-to-date and detailed survey of SDN threats and mitigations, we propose two research questions that we will try to address in this paper:

**RQ1: What are the main threats and vulnerabilities in SDN?**

**RQ2: What are the main approaches to provide security in SDN?**

To select the papers relating to these research questions, we rely on a three-stage selection procedure: (1) identifying search terms, (2) selection of sources, and (3) identifying and applying inclusion/exclusion criteria for selected papers:

(1) **Search terms:** This stage aims to select the most appropriate keywords to provide our review. Also, synonyms, alternative spellings of the main elements of the field of threats, and mitigations in the SDN context are included to find relevant papers. Therefore, keywords such as *"SDN threats", "SDN vulnerabilities"," SDN attacks"," SDN Security", "SDN security issues", SDN mitigations"," SDN security solutions"* were used for finding relevant papers.

(2) To define the search string, the Boolean operations "OR" was used to select optional words and synonyms and "AND" to choose the terms related to population, intervention, and effect, resulting in the search string (("software-defined network" OR "software-defined networking" OR "SDN) AND ("security" OR "secure" OR "attack" OR "vulnerability" OR "threats" OR "integrity" OR "mitigations")).

(3) **Selection of sources:** Table 1 shows the popular digital academic databases selected for searching at this stage. The articles from journals and conference proceedings are considered for our review. Some references for downloaded articles were looked upon, and publications related to interest were also included. Other databases were not used due to either the complexity (or impossibility) of instantiating the search string, presenting a high index of repeated articles with the IEEE and ScienceDirect databases, or performing only one indexing (like Google Scholar).

(4) **Inclusion/exclusion criteria:** Table 2 summarizes the inclusion–exclusion criteria for the SMS.

## 4 SDN security threats and attacks

### 4.1 SDN security threats and attacks classification

The SDN has many advantages over traditional networks due to the separation between the control plane and the data plane, making the networks programmable, flexible, and scalable.

**Table 1** Digital databases used in the SMS

| Online databases # | Online database | URL |
|---|---|---|
| 1 | IEEE | http://ieeexplore.ieee.org/ |
| 2 | ScienceDirect | http://www.sciencedirect.com/ |
| 3 | ACM | http://www.acm.org/ |
| 4 | Wiley | http://www.wiley.com/ |
| 5 | SpringerLink | http://link.springer.com/ |
| 6 | Hindawi | https://www.hindawi.com/ |
| 7 | Taylor and Francis | http://taylorandfrancis.com/ |

**Table 2** Summary of the inclusion–exclusion criteria for selection papers

| Criteria | Rational |
|---|---|
| Inclusion 1. The early-stage study should propose a solution for security in SDN | The SDN is our reference field. Thus, we need papers that directly proposed threat classification for SDN |
| Inclusion 2. The early phase study must describe attacks or vulnerabilities in SDN | Both academic and industrial taxonomies are essential to this study |
| Inclusion 3: Publication period | Papers published between 2010 and 2020 |
| Exclusion 1. A study that does not focus on SDN security | The emphasis of this paper is only on taxonomies that discuss web security issues |
| Exclusion 2. Non-English papers | The papers that did not publish in English are excluded |
| Exclusion 3. The study has not been published between 2010 and 2020 | The papers will be excluded after a full-text reading. If the focus is not on SDN threats and mitigations |

However, this has led to other security issues unique to the SDN [55]. Although the idea seems to have convinced many cloud and network service providers of its many benefits, it cannot be widely adopted if its security challenges are not addressed. Targeted attacks can disrupt the SDN software controller, and since the management subsystem is placed on the outside, it becomes the most vulnerable point of the entire system. Once every packet was routed separately, and no single control center could be easily attacked, a controller failure would disrupt the whole network. Such a center is an attractive target for attackers. However, if the central control fails, the switches are likely to go offline in route calculation mode and start working with the old technology—albeit slowly, without breaking the connection. Therefore, device designers are unlikely to abandon proven technologies completely and will initially use "pure" SDN/OpenFlow.

The objects for a targeted attack in SDN have become more than for a classic IP network: the controller, the communication channels between the controller and the switches, the switches themselves, and the communication channels for data transmission. IP protocol is focused on the fact that routers monitor the load and performance of the device's channels. In SDN, the controller monitors the network's state, which receives data on channels' failure and load "from outside". That means there is a delay in receiving information and making decisions, which does not contribute to reliability [126]. It is possible that by using this protocol, attackers will be able to insert devices that they control into the data flow route, allowing them to intercept network traffic and conduct other types of attacks. Also, the network management protocol can be dangerous, because it does not assume "intelligence" in the switch, so sending multiple commands to devices that already support OpenFlow could disrupt the network or a specific application. Incidentally, hardware manufacturers have already begun to implement support for OpenFlow in their products, and users who had not planned to build their SDN may be vulnerable to the unauthorized use of this technology, so companies will have to monitor the presence of the protocol OpenFlow in their network.

Another SDN issue is applications running on the controller and reconfiguring the network to meet an individual application or virtualization management system's requirements. At this point, it is an open question as to how these applications will compete for network resources in real time. Developing such network applications is complex, and there is doubt whether cloud users will do so optimally. This means that it is highly likely that at some point, some applications will monopolize network resources, effectively shutting down all other applications. Thus, there is another point at which application developers can go wrong and implement the network optimization algorithm incorrectly. Finding this error is problematic, because it only appears under certain conditions and high loads. Theoretically, configuration errors should be detected by the software that manages the network, but this requires SDN technology to be supported by the manufacturers of such software. In any case, the data center operator must monitor user applications and restrict them from monopolizing network resources.

With the development of SDN networks and the technologies involved, the study of security in these networks has become a significant concern, especially regarding their application in large network infrastructures. The challenges faced with SDN are that its vulnerabilities lie precisely in

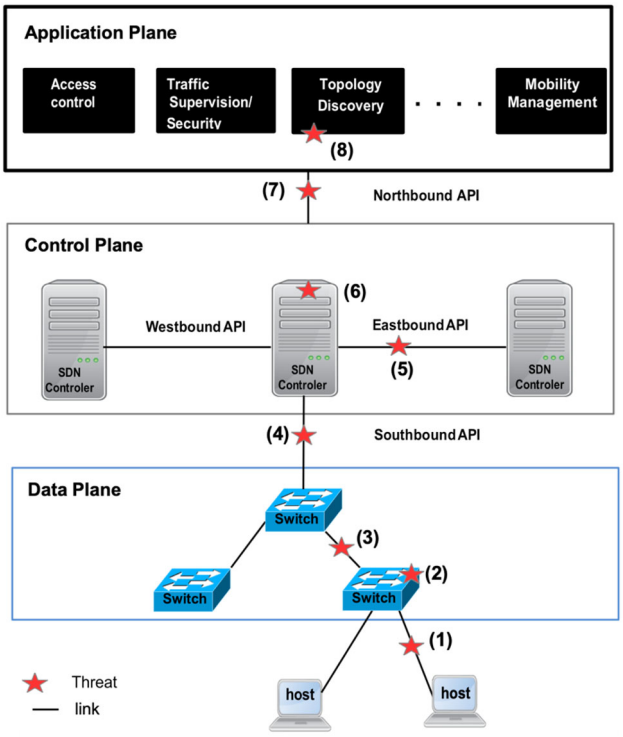


**Fig. 3** Threats analysis in SDN by layer

its main characteristics: software-driven network programming and centralization of logical control. This is because the SDN paradigm focuses on the operational characteristics of separating the control and data layer, leaving security as an external concern. These challenges need to be highlighted to take appropriate security measures proactively. Security analysis by several authors [3, 37, 122] in the literature has shown that the SDN suffers from many security threats. As shown in Fig. 3, there are several threats and vulnerabilities on the different layers and interfaces of the SDN network architecture. In this section, we will list threats to all layers and interfaces of the SDN.

#### 4.1.1 The SDN switch

An SDN switch is generally a separate device composed of related hardware and software, vulnerable to attacks. An example of vulnerability is the size limitation of flow tables.

#### 4.1.2 On the network equipment

With OpenFlow, an OpenFlow switch must buffer data flows until a flow rule is obtained from the SDN controller. In SDN networks, the controller installs the flow rules in the flow tables of a switch proactively or reactively. However, a switch has a limited number of flow tables in which the flow rules are installed. Since switches' decision-making capability has been externalized in the SDN concept by separating the control plane from the data plane, OpenFlow switches' main security challenge is distinguishing authentic flow rules from false ones. Another challenge is the number of flow entries that a switch can handle. This makes an OpenFlow switch vulnerable to saturation attacks (for example), because it has limited resources for buffering unsolicited, high-volume flows, whether UDP or TCP. A hacker can attack an SDN switch by manipulating its behavior to disrupt exchanges in the network. This can introduce fake traffic or flow rules on the switch with potentially disastrous consequences (data theft, traffic detour, overloading of the controller and neighboring switches, packet deletion) [149]. For example, removing a flow rule can interrupt communication between host machines. It can also lead to the abandonment of network packets or the generation of numerous messages to the SDN controller, which can cause a denial of service (DoS). Furthermore, it should be noted that in the SDN concept, the SDN controller has a direct implication on the SDN switches, because if the controller is out of order, the whole network is compromised.

#### 4.1.3 On the link between network equipment

The SDN architecture, exchanges on the links between SDN switches are not encrypted. This flaw allows hackers to intercept the information exchanged over these links, compromising network security.

#### 4.1.4 On the link between the controller and the network equipment

With SDN, if a switch does not receive a flow routing order from the controller, either due to the SDN controller's failure or a disconnection, the data plane becomes out-of-service. Therefore, the link between the switch and the controller must be protected, because it can potentially become a prime target for hackers to attack the network. Furthermore, it should be noted that the OpenFlow protocol in version 1.3 does not necessarily include data encryption with the SSL or TLS protocol on the link between the controller and the network equipment. The configuration of SSL or TLS is complex, which is why many vendors have ignored SSL or TLS support in their OpenFlow switches [47]. The lack of encryption on this interface and the lack of switch authentication at the controller level make the entire SDN network vulnerable to different attacks, such as Man-in-The-Middle, eavesdropping, or malicious access to flow rules occur.

#### 4.1.5 On the link between controllers

The links between controllers are subject to several threats. To ensure redundancy between them, the controllers

exchange updated information. A hacker can exploit the lack of encryption on this link to access and share updated information [84].

#### 4.1.6 On the controller

The threats specific to the SDN are the separation between the control plane and the data plane and introducing a logically centralized controller. The controller is the main component of an SDN network that provides control of the network. Therefore, it can be targeted to compromise the network because of its pivotal role. Applications implemented above the controller in the SDN architecture can be threats. Therefore, the SDN controller must face this challenge to authenticate and authorize applications before use. It is also necessary to separate the different applications according to their security implications before accessing network information. A compromised machine connected to the same network as an SDN controller can allow a hacker to quickly launch an attack on the SDN controller and take full control of the network [60]. Besides, the controller translates network commands that can be exploited to introduce an aggregated attack on the network. The SDN controller can also become a bottleneck to manage all incoming network flows.

Another big challenge for implementing currently available controllers is to specify the number of network devices managed by a controller to cope with latency constraints. If the number of flows on the controller increases, it is very likely that latency will increase, implying that more processing power is required at the controller. Limiting the controller's processing capabilities can cause a total network failure. It is possible to use several controllers to prevent one controller from becoming a critical point. However, it is shown in Yustus Eko Oktian et al. [96], that several controllers are not a solution to prevent the controller from becoming a critical point. The reason given in this article is that if the controllers supporting the load of the failed controller exceed their flow processing capabilities, this may cause a cascade failure of the controllers. Distributed DoS and DoS attacks on the SDN controller are critical security challenges. These types of threats can make a network resource unavailable to legitimate users. As proof, a DoS attack on an SDN controller is demonstrated by Zhang et al. [156–159].

#### 4.1.7 On the link between the applications and the controller

Unlike the Southbound API, the Northbound API is not standardized and suffers from vulnerabilities. A lack of a mechanism to ensure trust on the Northbound API interface used for communication between applications [95] and the controller can introduce threats. In other words, a weak authentication method between applications and the controller and inappropriate authorization allow a hacker to launch, for example, an identity theft attack or unauthorized access to applications. Thus, the hacker can use a malicious application to ask the controller, for example, to disconnect an application creating a flow modification problem. It can also send many requests to the controller to create a bottleneck and overload the processor. Various APIs for the Northbound interface can increase security threats due to built-in vulnerabilities, which decrease reliability between the controller and multiple applications.

#### 4.1.8 On applications

Hackers generally use applications to take control of the controller. This is facilitated because the applications and the controller software are most often located on the same physical machine. Thus, during exchanges between the applications and the controller, malicious code can be introduced into the control software via the Northbound API, which is also vulnerable. With unreliable applications, combined with the APIs' intrinsic vulnerabilities, a hacker can control the network by introducing his own rules on the controller. The centralized management of a network through a controller is fundamental for future networks, especially the Internet of Things [18]. However, it also opens up new security challenges. The challenges are numerous, from the risk of a critical point in the SDN controller to interface vulnerabilities and the need for authentication and authorization of network applications. Moreover, the list of these security challenges is expected to increase as the SDN is progressively deployed. Table 3 below summarizes the main security issues that exist in the three layers of the SDN.

DoS is within the vulnerabilities category, because the selected studies focused on its vulnerabilities, although it is an attack. In it were grouped the papers involving DoS and DDoS attacks. The selected papers focus on exploiting the Controller's difficulty or a switch running with resource exhaustion (CPU and memory usage). The ease of increasing network latency with packet flooding and the difficulty of identifying the real devices that exist in the network, expanding the network's size with fake devices in a way that makes network management unfeasible. The vulnerabilities related to the Access Control category exploit the possibility of accessing SDN components based on access permission policies and delegations problems. The Authentication category presents vulnerabilities to prospect the problem of impersonation: the difficulty of identifying whether an entity in the network is itself (be it a Controller device, switch, or even network administrator). The Confidentiality Breach category organizes studies in which the focus is on accessing sensitive information. Flow tables and flow rules are examples of information hijacked due to vulnerabilities in this

**Table 3** Major security threats in SDN

| SDN plane/layer | Proposal | Threat type | Description |
|---|---|---|---|
| Application plane | [33, 92, 94, 127, 140] | Lack of authentication and authorization | No robust authentication and authorization mechanism for applications |
| | [82, 109, 142] | Fraudulent insertion of flow rules | Malicious or compromised applications can generate false flow rules, and it is complicated to verify if it comes from a compromised application |
| | [46, 62, 70, 99, 127] | Lack of access control and traceability | Difficulty in implementing access control and traceability on third-party applications |
| Control plane | [4, 9, 11, 22, 27, 39, 52, 152] | Denial of Service attacks (DoS) | The centralization of network intelligence on a controller and the controller's limited processing resources are the main reasons for DoS attacks |
| | [23, 41, 69] | Unauthorized access to the controller | No powerful mechanism to implement access control on an application |
| | [17, 148, 156–159] | Interception | Centralizing intelligence in a single entity can pose scalability and availability issues |
| Data plane | [6, 75, 81, 100, 113] | Misconfiguration | The flow tables of OpenFlow switches can store a limited number of flow rules |
| | [20, 26, 110, 162] | Controller compromise | The data plan depends on the control plan that ensures its security. The security of the data plan depends on the security of the controller |
| | [1, 2, 36, 80, 83, 84, 93, 117, 120] | Man-in-the-middle attack | Due to the optional use and the complexity of TLS protocol |

category. The Misconfiguration category refers to vulnerabilities caused by network administrators, who intentionally or unknowingly inject misconfigurations into the network. The Integrity Violation category deals with the possibility of inserting spoofed information into the SDN network. Studies in this category modify the original data, such as packet data and flow rules. In the Controller Failure category, problems caused by Controller malfunction are related. This group of vulnerabilities studies the causes and consequences of this malfunction. The Network Update category focuses on the time lapse between network configuration changes, such as flow rules, firmware, etc. If a device depends on the device's configuration being updated during this period, conflicts can occur and open security holes. Finally, the lack of TLS category studies identifies the impacts of not using TLS to secure communication between SDN devices [111]. Although the TLS protocol is a possible solution, concerns are raised about how network devices manufacturers have implemented TLS. For example, the Openflow protocol specification is weak in this regard [83]. It only suggests using TLS in communication between devices without providing more details about how this use should occur.

Attacks directly related to SDN are rare, due to the early stage of research related to security in SDN. However, some attacks were found in the literature and are represented in Table 4. The identified attacks were Control Plane Saturation, which occurs when an attacker continuously injects requests for flows, to flood the communication between the Controller and the switch, Interception, which occurs when the attacker captures the control plane traffic, to divert the traffic, inspect the data or modify it. In addition to these attacks, poisoning attacks were identified, where an attacker injects packets on the network, to flood the data plane, resulting in a DoS; the Flow Table Overflow attack, where this category of attacks aims to produce new flows, in such an amount that exceeds the capacity of the flow tables, and thereby cause a DoS on switches. This can occur via a controller or a Men-in-the-Middle (MitM) attack [155]. Finally, the Fingerprinting attack aims to obtain information about the Controller, routes

**Table 4** Comparison of attacks in SDN networks

| Attack Type | SDN network level that is under attack | The aspect of security, which is affected by the attack | | |
|---|---|---|---|---|
| | | Availability | Confidentiality | Integrity |
| DDoS | Data transfer and control | X | | |
| DoS | Data transfer and control | X | | |
| Compromised controller | Data transmission and management control | X | X | X |
| Add-ons | Data transfer and management | | X | X |
| Man-in-the-middle | Data transmission, management<br>Data communication-control link | | X | X |
| Blackhole | Data transmission, management, data link-management | X | X | |
| Listening | Data transfer, and management | | X | |

or key devices on the network, such as servers, for example, from variations in network metrics, such as through the time difference when traversing packets over the network.

## 4.2 SDN threat analysis

### 4.2.1 DDoS/DoS attacks

SDN completely changes the architecture and intra-communicative aspects of components in the network—thus creating a completely new platform for attackers, who carry out attacks, disrupting the security of data networks. The networks operating under the SDN paradigm still have the same security requirements as the traditional ones, as they can also be used for transferring private and confidential information [117]. This makes it necessary to develop a similar level of security as in traditional networks, but for protection against threats of a different nature [73]. This section examines some of these significant threats and demonstrates the importance of protection against them in the SDN network. Numerical types of common DDoS attacks can be performed in the SDN environment. Still, the most widespread here is a variation of the attack using corrupted flow records, which attackers can use to attack the controller and compromise its availability. By blocking the controller with notes on the flow selection decision, the attacker causes the controller's computing resources to be remonetized. The controller will not be able to confront any legitimate notes it receives. Nationalizing the attack on the centralized control point (i.e., the controller), the intruder makes the entire measure significantly unusable for use. Although, data flows in the network can still function normally, even with the controller "collapsed". In the system, after the end of the terms of validity of the rules in their tables, the network devices advise the controller on the new rules in the network. The latter, in turn, will not be able to review the notes. However, if the network devices do not receive answers to their notes, they reset their routing and commutation tables or use archaic versions of these tables. As a result of such actions, the network may "fall apart".

On the level of data transfer, it is possible to use the created records that consume all the space in the flow tables to flood other devices. This makes the overpressure appliances unwilling to add lightweight flow records to their tables. This means that the devices cannot add new flow records to their tables, leaving the measure in a disjoint state. One of the key problems related to the devices on the data transfer level within the architecture defined by the software is that the devices cannot distinguish between legitimate and compromised flow records. This flaw allows attackers to carry out successful DoS attacks on data transfer level, flooding stream buffer of switches with memos of compromised character. Even though the adversary may focus on an individual path to the data and reduce its availability, it is much more likely that the attack will be carried out on the controller itself. It is much easier to create and expand access to the system. The prospect of this can be potentially ruinous, especially in manufacturing environments, where the most often used services will be inaccessible to customers and employees. In addition, the attacker can plan and carry out further attacks, which may be aimed at destroying the integrity and confidentiality of data on the network. For these reasons, DDoS/DoS attacks mentioned above are considered among the most important types of attacks.

### 4.2.2 Controller compromise

The controller can be seen as a centralized "brain" of SDN. It controls the entire network from a single point, making it probably the most important component of SDN architecture. A perpetrator who succeeds in compromising the controller essentially has control over the entire dimension [76]. The ability to control the controller's actions allows attackers to manipulate the flow records in any way they

choose, e.g., by disabling certain packets before reaching the final nodes, redirecting packets to their own "wanted" nodes in the infrastructure. Because of this, the attacker may compromise a particular network device and make it function as a "man-in-the-middle" or "blackhole/greyhole" node. "blackhole/greyhole". This will allow the attacker to potentially take down, change, or read the contents of any package received by him [118]. Another possibility is that the attacker successfully registers a non-legitimate controller at the control level of the SDN network. Using this malware controller, the attacker can influence the availability of other controllers, change the rules, and effectively disable/manipulate the operation of add-ons at the add-on level. Although the centralized nature and ability to coordinate information in a single moment can be beneficial for the measurable administrators and programmers, it can be used to attack the integrity of message management or to obtain sensitive information about the add-on, the presence of important services for system users, and the confidentiality of user information used by programs on the level of add-ons. Any attack that targets a controller in the SDN architecture can have potentially devastating effects. To successfully restrain the effect of the compromised controller, the protection system must focus on ensuring the authenticity of the controller before allowing it to make any changes to the measure.

#### 4.2.3 Add-ons

As a result of using SDN technology to integrate third-party add-ons, malicious add-ons arise. Add-ons that demonstrate malicious behavior in the SDN environment can lead to catastrophic consequences similar to those when a controller is compromised. Add-ons engaged in deep packet inspection can lead to potential risks for the network, since they can have the ability to indirectly control the entire network through the information they have collected during packet inspection. Authentication and authorization of add-ons to work in the SDN environment are important. The increased amount of data and the method used to centralize it make it possible for malicious add-ons to threaten the integrity and confidentiality of the information about the user and the measure to which they have access.

Securing the front-end interface is challenging, because every add-on that uses it may need access to a unique subset of information from the controller. To successfully control this, it is necessary to enforce a specific policy on access to information. This ensures that the addressee declares what information they require and can only access it. This can ensure that supplements do not steal sensitive information and use information from other programs. You should also ensure authenticity before the add-on can communicate with the controller.

#### 4.2.4 Attacks on control level communication and data transfer

Another key area of SDN, which presents opportunities for attack, is the link between control and the level of data transfer. The OpenFlow specification makes the use of TLS (Security Layer Transport) obligatory [121], which makes it a weak point and susceptible to various attacks, such as man-in-the-middle and blackhole. A man-in-the-middle attack occurs when a malicious node is placed between the controller and devices at the data transmission level. Instead of transmitting messages directly to the controller (or vice versa), the man-in-the-middle hub can maneuver and check the volume of packets [56]. A blackhole attack can also be performed, in which the device is placed between the target device and the controller and simply drops any packets it receives without forcing them on the controller. This leads to the breakdown of network communications and the inaccessibility of services to easy users.

If the attacker succeeds in establishing himself as a mediator between control and data transmission, it can potentially disrupt the entire network. A man-in-the-middle attack is directed at the integrity of controlling communications between network devices at the data transmission level and the controller. An intruder can change the control messages and create a way of shaping the network to give him certain advantages. On the other hand, a blackhole attack attacks the availability of network services. Suppose all the communications between the network devices and the controller are not forwarded through a malicious hub. In that case, the link will inevitably fail, and the devices at the data transmission level will not access the controller when needed. This link between the control level and the data transmission level is certainly a weak point, and is the target for the attack by the attackers. Therefore, it must be well protected before the SDN network is used in the production process.

#### 4.2.5 Listening attacks

Attackers attempting to gain illegal access to the SDN network or disable access to the services it provides can use eavesdropping attacks (illegally taking packets transmitted through the connection) on certain connections in the network [13]. This can allow them to collect important information, which can then carry out more insecure attacks. Attacks that use the auditory method have been performed in traditional merged-network settings for a long time-deviceless archetypes are particularly weak to such attacks through their transmission through the air. However, in the context of SDN, listening can be used to check the packets used for communication between the control level and the data transmission level, or exclusively at the data transmission level. At the level of data transmission, the device

that is used in the listening mode, integrated into the OpenFlow commutator, can be exploited by an intruder (who could compromise the commutator), to verify the packets transmitted by external commutators, which allows the attackers to see the important information. In some senses, eavesdropping, which occurs at the control and transmission levels, is mostly a passive attack and does not directly affect the data's availability, confidentiality, or integrity. However, it allows for further attacks that degrade these aspects of security.

In the context of eavesdropping, which is carried out in the link between the levels of the addressee and the management, the confidentiality of the information can be compromised. An intruder may be able to look into the information related to a certain user, if they can listen to the connection, transmitting important data. This makes eavesdropping a particularly insecure type of attack—confidentiality must be ensured before critical data, which contains confidential information, can be ignited in the SDN environment.

#### 4.2.6 SDN attacks comparison

The table below summarizes the above survey of attacks on the SDN network. Also, in this table, we can see the levels of the SDN network, which are affected by these or other types of attacks, and the specific security aspects they can potentially compromise, as shown in Table 4.

Every attack investigated in this section was still quite widespread for traditional networks. Most interestingly, with the emergence of new attack platforms for SDN architecture changes, attack variations emerge exclusively to SDN. Even though the traditional variants of each of these types of attacks can be more effectively resolved due to the centralized nature of SDN, the modified variants of conventional attacks, which are attached to SDN, pose the greatest threat and require the greatest care in working with the security of SDN. The following section will familiarize you with the proposed solutions for the above attacks and evaluate their effectiveness.

## 5 SDN security mitigations

This section discusses some of the protection methods suggested for these particular threats and offers some general solutions to ensure the security of both the control level and the level of data transmission.

### 5.1 Mitigations techniques at the application plane

The SDN control layer interfaces between network equipment and applications to reduce network complexity from applications. Thus, the SDN facilitates applications that can extract network statistics and packet characteristics data via the controller to implement new security services. Several tools and network programming languages have been proposed in the literature to secure the application layer, such as Procera [138], Frenetic, and NetCore [88]. OFTesting, a Python-based OpenFlow application, is used by this application layer to debug and automate OpenFlow programs.

SDN applications need to follow these applications to claim the network specifications regarding the information. To dress this issue, one of the critical solutions is permOF [144], which is a permission system that provides privilege control for OpenFlow controllers and applications running above the SDN control plane. PermOF is based on 18 permissions to be applied by the controller APIs. It also provides a custom isolation framework that maintains various priorities for applications and isolates control traffic from data traffic to achieve complete resource isolation and access control. The read permission defines what information the application can retrieve from the network through the controller. The write explains whether an application can modify or not state the controller or switches, notification handle permission of notifying the application if a specific event acquires, and the system category manages access to the operating system's local resources.

Fake flow rules remain the first and most critical issue to deal with, any flow rules residing within the controllers are legitimate and expose the controller to propagate false and misleading routing information, hence the importance given to solve this issue among the researcher community. Many types of research propose FLOVER, a model-checking system that verifies the flow policies against the security policies and makes a decision that is transferred to the controller.

In the SDN architecture, FLOVER [132] is executed as an OpenFlow application, but logically, it is located parallel to the controller. The controller is modified to request FLOVER's permission on every new flow rule generation or modification. Primarily, the controller communicates flow tables and security policies to FLOVER, which contains critical security information to the network preconfigured by the network administrator, so FLOVER can evaluate requested changes accordingly and respond to them. FLOVER also uses the controller to access the current state, network information, and statistics, like flow rule tables on the switches. Figure 4 shows the architecture of FLOVER alongside the communication between the Open Flow controller, OV switches, and the FLOVER application.

Flover formalizes two key components of the Flover framework:

(a) Non-bypass property representation:
A non-bypass security property states a feature within a specific flow rule set. A non-bypass security property is a first-order logic involving a universal quantifier, conditions and an action. An action is either forward or drop.

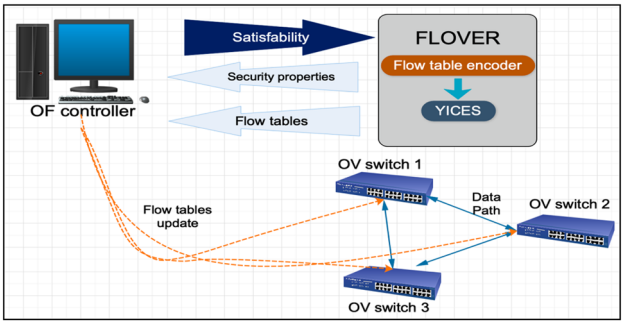


**Fig. 4** FLOVER framework architecture

The conditions part of a non-bypass security property combines boolean formula overflow rule set fields. Each field's condition is represented with a boolean expression stipulating a range of non-negative integers because every field consists of several bits with a length that varies from 3 to 64. The maximum number of fields is a max of 15.

(b) Flow rule set representation

Every flow entry (flow rule) has a flow table number specifying where it goes. Every flow rule is attached to conditions over fields and a set of actions. The requirement for each field is a set of positive integers, which supports the wildcard Formula. Each flow rule's action set requires forwarding, drop or goto as a finishing action. The action set for a flow rule can have several set commands that alter the packet corresponding to the flow entry condition. No match represents that a flow table has no corresponding flow rule. When there is no matching rule for an incoming packet, an OF switch sends the OF controller packet and requests a new flow rule in the default setting. Thus, for packet p, the possible final action set of FlowRule in FlowTable can be modelled using the following logic forms.

$$\text{FlowRule}_i \in \text{FlowTable}_k,\ \text{Cond}_i(p) = \bigwedge_{j=0}^{n} F_j(p) \in [i\text{Low}_{i,j}, i\text{High}_{i,j}],$$

$$\text{FlowRule}_i(p) = \{a | a = \text{action}_{\text{final}} \text{ of FlowRule}_i \ s.t\ \text{Cond}_i(p) = \text{true}\}.$$

In [142] the authors propose PERM-GUARD, a system for managing and authorizing flow rules. PERM-GUARD uses an authentication/authorization model to verify the controller's flow rules' validity through identity-based signature. This solution effectively filters out unauthorized flow rules created by valid applications and traces their creator quickly and accurately.

PERM-GUARD provides the following security functionalities:

- Managing the flow-rule-production authorizations set of an authorized SDN application,

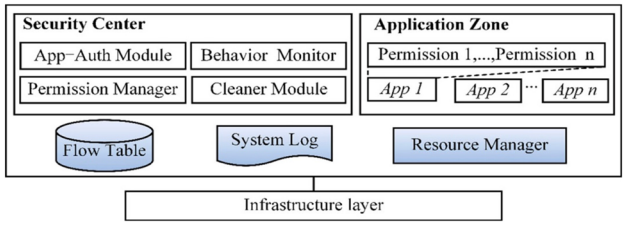


**Fig. 5** PERM-GUARD-flow rule production-permission authentication scheme

- Verifying the authenticity of flow rules,
- Authenticating the information of an SDN application that generates flow rules insertion requests.
- Monitoring irregular behaviors of malicious SDN applications.

PERM-GUARD uses four security elements in the controller to accomplish all these functionalities, shown in Fig. 5.

- App-Auth Module. The app-auth module validates the application's identity that generates flow rules insertion. Fake flow rules generated by unregistered applications will be handled as invalid and dropped. If the flow rules source is a valid application, then the App-Auth module verifies the generator's flow-rule-production permissions. The new flow rule will be recognized if it is has been inserted in an authorized manner; else, it will be taken care of as an invalid rule and rejected.
- Behavior Monitor Module. The behavior monitor component saves unusual behaviors of local and remote SDN applications that interact with the SDN controller, e.g., unauthorized attempts to add or alter flow rules on the SDN controller, prohibited registrations on the controller, etc. The behavior monitor typically periodically sends the monitoring results to the permission manager component.
- Permission Manager Module. The permission manager module configures the flow-rule-production permissions set of an application depending on its behavior records sent by the behavior monitor module. The permission manager will send every application's latest flow-rule-production permission set to the app-auth module permissions after it is done with modification.
- Cleaner Module. The cleaner module purges the information in a temporary buffer created by PERMGUARD to block system cache information from being illegally exploited.

The lack of third-party application implementation of any authentication and authorization mechanism compromises the normal network behavior. FRESCO [89], a security-specific application development framework for Open Flow networks, was proposed. FRESCO is a security application

development platform that facilitates API scripts' exportation, which helps security experts develop threat detection logic and security monitoring as programming libraries. Moreover, the FRESCO programming framework was presented to help attain rapid design and modular composition of various security mitigation and detection modules using Open Flow.

It is composed of a FRESCO application layer and a security application core. The FRESCO application layer provides a development environment and resource controller that allows developers to use the FRESCO scripting language to compose security functions. In contrast, the FRESCO security application core is integrated with the OpenFlow controller to ensure policy compliance and avoid conflicts. By integrating FRESCO on a NOX-type SDN controller, the authors have demonstrated that FRESCO does not introduce overloading and creates security functions quickly with fewer code lines.

In Wang et al. [142], the authors proposed a modular SDN security-control communication architecture, KISS, with innovative solutions in key distribution and secure channel support. Besides, they suggest iDVV, the device's integrated verification value, a code protocol for generating a deterministic secret code but indistinguishable from chance. This allows local but synchronized generation/verification of keys at both channel ends, even by message. IDVV should significantly contribute to the robustness and simplicity of authentication and secure communication problems in SDN.

Integrated device verification values (iDVVs) are sequentially generated to protect and authenticate requests between two networking devices. The generator is constructed such that the indistinguishability-from-random and determinism properties are in its output series. As a result, the same sequence of random-looking secret values is generated on both ends of the channel, enabling the stable, decentralized generation/verification of per-message keys. However, there is no way an opponent can know, predict or produce an iDVV if the seed and key initial values and the generator state are kept secret.

To satisfy the needs of SDN, the iDVV generation is made versatile. In other words, an iDVV is a specific secret value created by an A device (for example, a forwarding device) that can be checked locally by another B device (e.g., a controller). Therefore, iDVVs can be generated: (a) on a message basis; (b) for a sequence of messages; (c) for a specific time interval; and (d) for a single session of communication. The main benefits of iDVVs are their low cost and the fact that they can be produced locally, i.e., without any prior arrangement needing to be formed. The iDVV generation is more straightforward and faster than standard KDF algorithms and related solutions such as HKDF (RFC 5869).

As discussed before, the combination of two SDN devices, e.g., forwarding devices such as switches and SDN controller, occurs through KDC, under the protection of the secret keys gained from registration (Kkf, resp. Kkc). The association protocol's result is the supply of two random secrets to both devices: a seed seedij, and an association key keyij. The iDVV mechanism is bootstrapped by implementing these two secret values in both the controller and the switch, to run the iDVV generation algorithms. Note that setting up and generating an iDVV values is executed in a deterministic way to achieve locally at both ends. Nevertheless, as iDVVs will be implemented as keys by cryptographic primitives such as encryption functions or MAC, they must be fuzzy from random. Hashing primitives are typical choices for iDVV algorithms since they provide indistinguishable-from-random variables if one or more input variables are identified only by the source and the destination. This demonstrates why seed and association keys are vital to be encrypted and therefore known only to the connecting devices. Besides, all of the variables seed, key, and idvv in the algorithms below should have the same length to avoid information leakage, which the 256 bits are chosen at iDVV design. After the bootstrap with the first idvv value, the idvv_second function is invoked on-demand (over, synchronously at the two sides of the channel) to autonomously produce authentication or encryption keys used for securing the communications. The key stays the only constant shared secret between the devices. The seed progresses to a new indistinguishable from random value every time idvv_next is called to create a new iDVV. The new seed results from a hashing primitive H over the actual seed and actual idvv (line 2). The new idvv, the output of function idvv_next, results from a hashing primitive H over the combination of the new seed and association key.

Stable communication protocols are ambiguous in the iDVV mechanism. They can be used in various ways, in several protocols, as a key-per-message or key-per-session, etc. The only main issue about creating iDVV is holding it 1We omit the device-identifying subscriptions in readability variables. 3 The two ends of the channel are aligned. Therefore, in this respect, we are presenting some suggestions.

SDN controller is responsible for handling all network requests and events. As the size of the network keeps growing, the controller can reach a state of a bottleneck. Benchmarking the SDN NOX [53] controller shows that it can support 30 k demand. This rate can be a severe issue for networks with high requests.

To assuage this matter, leveling parallelism in multicore systems was proposed by Tootoonchian et al. [135]. Their approach showed that minor alterations to the NOX controller increase its performance by greatness on a single core. It means that a single controller can support a further, more extensive network, given adequate controller channel bandwidth with acceptable latency. More improvement can be achieved by reducing the number of requests sent to the controller using DIFANE [53] individual switches, called

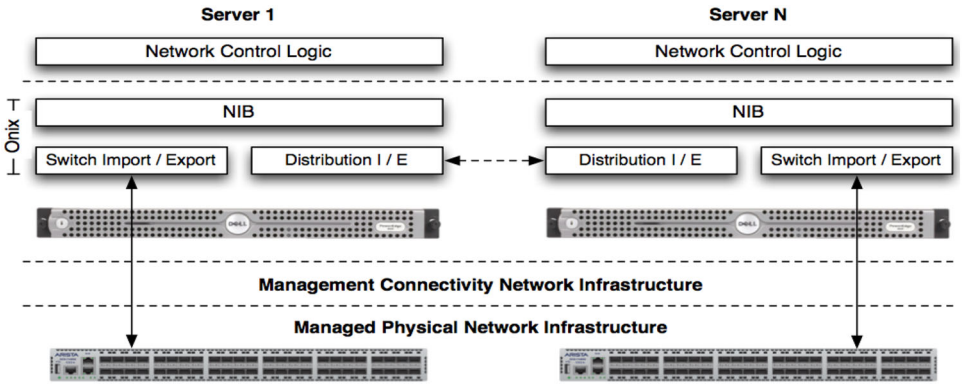


**Fig. 6** Two ONIX controller coordinating and their views of the underlying network state

authority switches. These switches are used to discharge the controller's tasks and handle the packets in the data plane. Most micro-flows are held in the data plane to reduce the need to request the control plane and increase the scalability. The Onix controller is a control platform for large-scale networks. The Onix project is built on NOX. NOX names a centralized network control operating system developed by Nicara and introduced to the community in 2009, built-in C++. NOX uses the centralized controller to manage, and the switches implement the management NOX [71]. The Onix architecture is shown in Fig. 6.

There are four components in a network controlled by Onix, and they have very distinct roles.

- Physical infrastructure: This component contains network switches and routers, also any other network components (such as load balancers) that support an API allowing Onix to read and write. These network components do not have to run any programs further than that mandatory to support this interface and accomplish basic communications. The state controls the element's comportment (such as forwarding table records).
- Connectivity infrastructure: The connectivity between the physical networking equipment and Onix (The control part) transits the connectivity infrastructure. This control passage may be used both in-band (where the control management shares the same forwarding equipment's as the data traffic on the network) or out-of-band (where a distinct physical network is implicated in handling control traffic). The connectivity infrastructure needs to support two-way communication between the Onix controller, and the open-flow enabled switches and optionally maintain convergence on link breakdown. Standard routing protocols such as IS–IS and OSPF are appropriate for establishing and maintaining forwarding operations in the connectivity infrastructure.
- Onix: Onix is a distributed system that turns on a cluster of one or more physical servers, each of which may run several Onix instances. Onix provides the control logic and programmatic access to the network (reading and writing network state). An Onix instance is also responsible for disseminating network status to other cases within the cluster to scale to extensive networks (millions of ports) and provide the necessary resilience for production deployments.

  Control logic: On top of Onix's API, the network control logic is implemented. This control logic defines the desired network behavior; Onix provides the primitives required to access the appropriate network state.

Many other solutions to the scalability issue were presented in the literature, such as HyperFlow [135], consisting of a physically distributed and logically centralized control platform.

The HyperFlow framework is a C++ NOX application created to ensure a single network-wide view of all controllers. Each controller runs an instance of the HyperFlow application. The implementation of HyperFlow includes minor modifications to the core controller code, primarily to provide sufficient hooks for intercepting commands and serializing events, as shown in Fig. 7.

Each controller runs NOX atop the HyperFlow program, subscribing to the Control, data, and canal in the publish/subscribe framework (depicted with a cloud). Events are released to the data channel, and periodic advertisements are sent to the control channel. Controllers publish commands targeting a controller directly to its channel. In the source controller, the answers to the commands are written.

Hyper flow operates as described below:

- Initialization: Upon launching NOX, the HyperFlow program launches the WheelFS client and storage services, subscribes to the data and control channels of the network, and begins regularly advertising itself in the control channel. The advertising message contains the controller's information, including the switch identifiers it controls. The advertising interval among controllers in a network must be greater than the highest round-trip time.

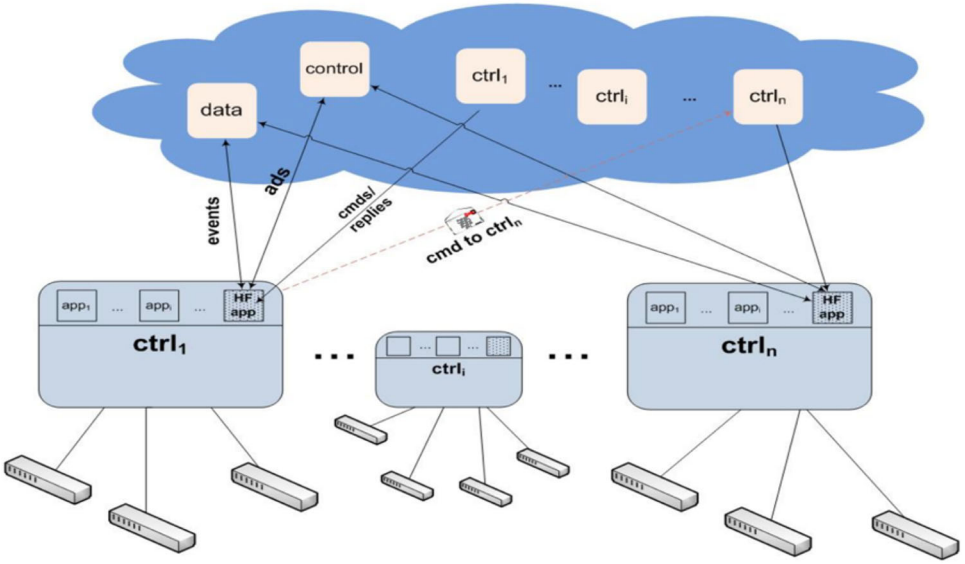


**Fig. 7** A high-level overview of HyperFlow

- Publishing events: The HyperFlow framework collects all of the built-in (OpenFlow message events) NOX events and the events recorded with HyperFlow by the applications. It then selectively serializes (using the Boost serialization library) and publishes those created locally and affects the controller's state. To this end, applications must be configured to tag incidents that impact their status. Besides, the parent event of any non-built-in event they cause should be defined by applications. This allows HyperFlow to track and propagate each high-level event back to the underlying lower level event instead. Using this approach, we ensure that the number of propagated events is restricted by the number of OpenFlow message events that the local controller produces. The name of the published messages includes the source controller identifier and the publisher's local event identifier. This method effectively partitions the message namespace between controllers and prevents writing conflicts.
- Moreover, Hyperflow has a cached copy of a message (file) never becomes stale. Therefore, Hypeflow instructs WheelFS to alleviate consistency by using semantic signals. We note that this requires that the network operator establish a replication policy suitable for network configuration.
- Both message forms include the publisher controller ID of the publisher (ctrlid). Events and commands also include the publisher's locally generated event identifier (eventid). Commands also include the switch's identifier for which the command is intended. Copies of stored files are stored and retrieved from nearby controllers as soon as possible.
- Replaying events: The HyperFlow framework replays all published events since source controllers selectively filter out and only publish the events required to recreate the application state on other controllers with apps' support. The HyperFlow program deserializes and fires it upon receiving a new message on the network data channel or the controller's channel.
- Redirecting commands targeted to a non-local switch: A controller can control only the switches connected to it directly. The HyperFlow application intercepts when an OpenFlow message is about to be transmitted to those switches and publishes the command to the network control channel to program a switch that is not under the controller's direct Control. The name of the published message implies that it is a command and includes the source controller's identifier, the destination switch's identifier, and the identifier of the local command (similar to the event message identifier).
- Proxying OpenFlow messages and replies: The HyperFlow application collects command messages that target a switch under its Control (identified in the name of the message and sends them to the destination switch. The message name includes all identifiers for the controller. The HyperFlow program maps the message transaction identifiers (xid) and the source controller identifiers to route the replies back to the source controller. The HyperFlow application tests the locally generated xid OpenFlow message events of the controller. If the event xid is located in the map of the xidcontroller, the event is prevented from being further processed and released to the network's data channel. Upon receipt, the source controller picks up and replays the incident.
- Health checking: The HyperFlow program listens to advertisements for the controller in the network control channel. If a controller does not re-advertise itself, it is believed to have failed for three promotional periods. For every switch attached to the failed controller, the HyperFlow application

fires a switch leave case. Upon controller failure, HyperFlow configures the failed controller-associated switches to connect to another controller. Alternatively, four nearby controllers may serve to take over the IP address as a hot standby for each other.

Shin et al. [123] proposed IRIS-HiSA, a cluster architecture for the distributed controller. Its main objective is to support uninterrupted load balancing and failover with horizontal scalability, as is done in existing work. Still, one of IRIS-HiSA's distinctive features is to provide transparency between the data plane and the switches. Thus, the switches do not need to know the controller cluster's internal details, and they simply access the same way a single controller is accessible.

IRIS-HiSA has two main goals:

1. Provide high scalability and availability: IRIS-HiSA utilizes a pool of distributed controller instances to provide the SDN controller with ample scalability and availability. All controller instances run in active mode for high scalability, and the traffic loads among the controller instances are well balanced to optimize the use of computational and networking resources. For high availability, seamless failover is provided by IRIS-HiSA. The connected switches are seamlessly migrated to the other controller instances, even though some controller instances go down.
2. Allow transparent access to the controller cluster: The IRISHiSA controller cluster appears as a single logical controller for straightforward access to a controller cluster by covering the controller cluster's internal information. Using the controller cluster's representative IP address, switches may connect to the controller cluster without knowing the number of instances of the controller or the address of each instance of the controller. As IRIS-HiSA offers smooth load balancing and failover, internal information such as load balancing and failover need not be understood by the switch.

As shown in Fig. 8. The IRIS-HiSA architecture consists of a pool of examples of controllers, a HiSA controller, and a HiSA switch. The HiSA switch connects both the controller instances and the HiSA controller, and there is no logical restriction to the number of controller instances.

By tracking the state of both the controller instances and switches, the HiSA controller supports load balancing and failover.

An OpenFlow capable switch is a HiSA switch, and it makes the controller cluster seem to be a single logical controller. For example, switches connect to the controller cluster via a HiSA switch using the controller cluster's representative IP addresses. It queries a HiSA controller with a PACKET IN

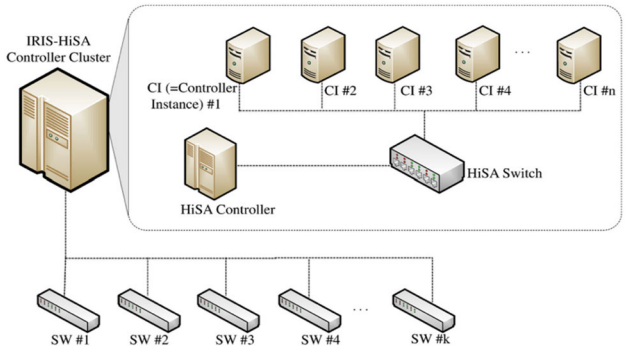


**Fig. 8** The architecture of the IRIS-HiSA controller cluster

message, because a HiSA switch does not have any matching rules for link setup messages like an SYN packet. Upon receiving a PACKET IN message, the HiSA controller uses the load balancing algorithm to assign the control instance to manage the new link. Changing the IP/MAC packet header also sends flow rules to the HiSA switch, which directly links the switch and a particular instance of the controller.

The HiSA controller also enables switches to be seamlessly transferred to another controller if the controller's instance to which the switches are attached fails. Therefore, the controller cluster appears as a single logical controller fitted with good robustness and high scalability from a switch perspective.

SDN applications access the controller for a wide range of reasons. It is primordial to ensure that these applications work within their respective perimeter with the legitimate functional requirement. For that, securing the controller from the malicious application must be provided.

SE-Floodlight [3] controller was proposed as an extended version of the Floodlight controller. SE-Floodlight introduced several security enhancement methods:

- Privilege separation by adding a secure programmable northbound API enables SE-Floodlight to work as a truly independent mediator between the application plane and the data plane.
- A runtime integrity validator of the modules that generate flow rules. The runtime copy of the local flow-rule producer is compared to the original image installed by the administrator.
- A Rule Reduction (ARR) algorithm that manages inline rule-conflict detection.
- Role-based conflict resolution by comparing the authoritative roles of the producers of the conflicting rule
- PACKET_OUT Control: the administrator can block packet control generated by OpenFlow apps.
- Security Audit: is a subsystem that tracks all security events

**Table 5** A summary of control layer mediation policies for data flows initiated from the application layer to data plane (A to D) and from the data plane to application layer (D to A)

| Flow direction | Data exchange operation | Mediation policy | Minimum authorization |
|---|---|---|---|
| 01: A to D | Flow rule mod | RCA (Section IV-C) | APP |
| 02: D to A | Flow removal messages | Global read | APP |
| 03: D to A | Flow error reply | Global read | APP |
| 04: A to D | Barrier requests | Permission | APP |
| 05: D to A | Barrier replies | Selected read | APP |
| 06: D to A | Packet-In return | Selected read | APP |
| 07: A to D | Packet-Out | Permission | SEC |
| 08: A to D | Switch port mod | Permission | ADMIN |
| 09: D to A | Switch port status | Permission | ADMIN |
| 10: A to D | Switch set config | Permission | ADMIN |
| 11: A to D | Switch get config | Permission | APP |
| 12: D to A | Switch config reply | Selected read | APP |
| 13: A to D | Switch stats request | Permission | APP |
| 14: D to A | Switch stats report | Selected read | APP |
| 15: A to D | Echo requests | Permission | APP |
| 16: D to A | Echo replies | Selected read | APP |
| 17: D to A | Vendor features | Permission | ADMIN |
| **18: A to D** | Vendor actions | Permission | ADMIN |

SE-Floodlight incorporates a Security Compliance Kernel (SEK) into the Floodlight Controller to mediate all data exchange operations between the application layer and the data plane. The SEK applies the application to the data plane mediation scheme shown in Table 5 for each process. In Table 5, the Minimum Authorisation column identifies the minimum function delegated to an application to perform the procedure. As discussed in the previous Section, SE-Floodlight implements a hierarchical authorization system with three default authorization functions. The lowest authorization function, APP, is primarily intended for traffic engineering applications (non-security-related) and provides ample permission for most such applications for flow control. SEC's Security Authorization function is designed for applications implementing security services. The highest authorization function, ADMIN, is intended for applications like the operator console app.

The mediation service aims to provide a configurable permission model for a given SE-Floodlight deployment. The collection of roles can be expanded, and their permissions can be customized for each newly specified function.

The default mediation policy allocated to each available contact between the application layer and the data plane is provided in column 3 of Table 5. First, any OpenFlow application's inherent objective is the ability to define or override existing flow policies within a switch (row 1). However, as defined in Section IV-C, SE-Floodlight incorporates Rule-based Conflict Analysis (RCA) to ensure that each candidate flow rule submitted does not conflict with an existing flow rule whose author is aligned with a higher authority than the candidate rule's author.

In OpenFlow, a rule conflict occurs when the candidate rules permit or disables a network flow otherwise forbidden (or permitted) by the current rules. Conflicts are either direct or indirect within OpenFlow. An immediate dispute exists when the candidate rule contravenes an existing rule (e.g., it forwards packets between A and B while the candidate rule drops them). If the candidate rules logically pair with pre-existing rules that contravene a current rule, an indirect dispute arises.

Event alerts used to monitor the status of the flow table provide a second class of operations. These operations do not alter network policies but provide the knowledge required to make informed flow management decisions for traffic engineering applications. The default permission model describes these operations as two types of public reading. The Global Read reflects events in the data plane streamed to all applications involved in receiving them (rows 2 and 3). Chosen read operations apply to individual events for which an application can register to receive switch state-change alerts via the controller (rows 5, 6, 12, 14, and 16). Selected read alerts are responses to permission-protected behaviors that will be addressed next. The Packet-In notification is an exception. It is received in response to the flow rule's insertion (vetted via RCA) that activates the switch to alert the application when the packets meet the flow rule requirements are received or when no matching flow rule is found.

The third class of behaviors includes those needing specific permission (Rows 4, 7–11, 13, 15, 17, and 18). Such activities either make direct adjustments to the network flow policies enforced by the switch or allow the operator to the SEK empowers an OpenFlow stack from the bottom of Fig. 9 to serve multiple applications in parallel, implements control layer arbitration when apps generate contradictory flow logic and impose the limitations of application permission. The SEK extends Floodlight with five key components.

A Role-based Source Authentication module provides digital signature validation for each flow-rule insertion

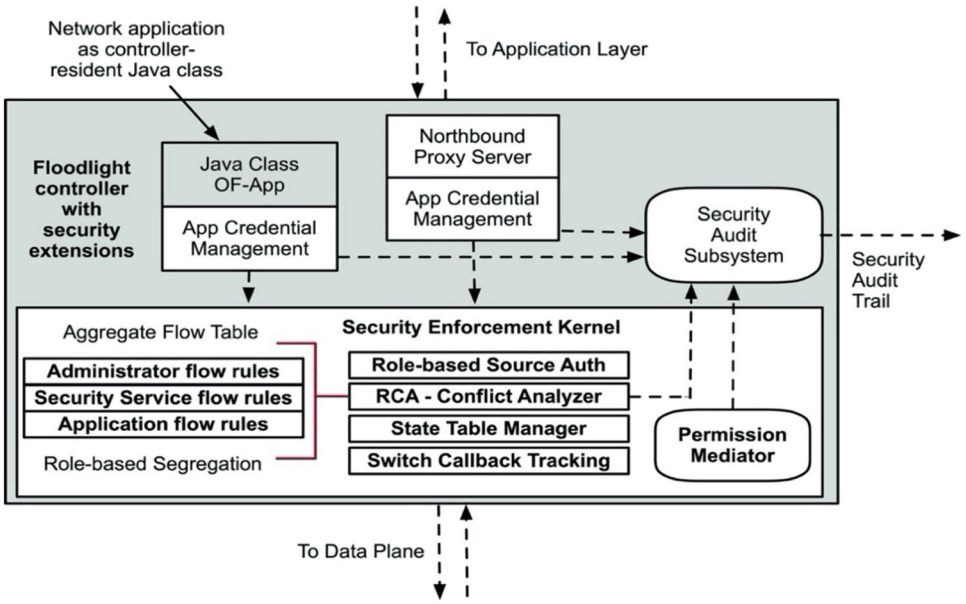


**Fig. 9** Floodlight controller with security extensions

request to temporarily restrict the priority of a candidate flow rule based on the application's operating position.

Giotis et al. propose in [51], a method combining the OpenFlow protocol specifications and the standard sFlow traffic monitoring technology (sFlow.org, 2017). This method provides an effective mechanism for detecting anomalies and mitigating their impact on SDN environments in real time. The solution consists of three main modules. A collector module periodically collects information and flow statistics installed in the switch tables of the switches. The authors use sFLow, a flow monitoring mechanism, while using packet sampling to accomplish this task. This module periodically (every 30 s) sends sampled packets from the collected flows to an anomaly detection module to identify potential attacks based on the flow information. When an anomaly is detected, this detection module communicates with a mitigation module responsible for taking the necessary [148] countermeasures to mitigate the effects of the attacks on the SDN network. This module distributes and installs the appropriate switching rules in each switch in the network to neutralize the identified attacks by blocking malicious traffic. The results have demonstrated that the proposed solution successfully detects DDoS attacks, worm propagation, and port scan attacks. Using the sFlow packet sampling capacity, the solution reduces the communication required between the switches and the SDN controller and reduces the controller's processing load. However, the systematic collection of data from the entire flow table, containing thousands of entries, is unsuitable for high network traffic environments. Besides, this solution does not solve the problem of switch table exhaustion during the attack. Similarly, this technique does not consider false positive alerts since it directly blocks traffic considered malicious.

In the situation categorized as Lack of TLS, [148] proposed resolving the interception category attack with anomaly detection. Due to the lack of TLS, an attacker can perform a MitM attack. This paper provides an IDS that organizes all the flow rules of the switches into a global table and monitors the information regarding the global path of a flow, the time for a packet to travel along the path of a flow, the number of packets and bytes that cross the path of a flow, among others. Based on the comparison of this information, the IDS detects the existence and location of a MitM by the time the attacker uses to receive, process and resend packets over the network. The SMS results indicated several topics where significant efforts are being invested in addressing them (DoS, for example). Several other issues are in the early stages of exploration, opening up an extended field of research.

In the Misconfiguration vulnerability category, the work of [90] is faced with managing Firewall policies in an SDN formed by distributed controllers, where a new point policy can impact the policies of other controllers. A pre Firewall algorithm is proposed to solve the problem, which receives a policy and compares it with a set of policies in use, where it tries to detect anomalies in the policy. Two types of anomalies are possible: the shadowing anomaly, where a policy field corresponds to the field of a policy that is in the set of policies, in a full, inclusive, or partial way, and the priority of the compared policies are not equal; and the correlation anomaly, which occurs in the same way as the shadowing anomaly, but the priorities of the policies are similar. In the shading case, depending on the policies' actions (allow or deny) and priorities, the new rule may be added, partially added, or eliminated. In the correlation case, the new policy can be added. In contrast, the old policy is deleted or split through

a policy splitting function, which returns new policies with the correlation comparison's disjoint parts [48].

On the other hand, solve the Misconfiguration problem with a model checking solution. This work verifies if the configurations defined in the P4 language meet a set of properties. The proposed solution provides an assertion language that allows programmers to specify their properties by annotating programs written in P4. This proposed language enables the invariants to specification, and once these invariants have been annotated, the program is symbolically executed, with the assertions being checked while all paths are traversed. To do this, the developer first annotates the assertions in the P4 code. Then, the P4 program is translated into a C language-based model. Some routing rules can be inserted into the translator during this process to restrict the verification for a given network configuration. The generated model is then verified by a symbolic engine, which tests all execution paths, looking for failures in the assertions [48].

The Misconfiguration category is also addressed by Kim et al. [68], who proposes a policy management solution, where converts high-level policies into low-level policies through the NETCONF protocol. A high-level policy generator has proposed a converter of policy information, defined by the network administrator through a graphical interface, into an XML format, and a parser of these rules for NETCONF. In the case of vulnerabilities involving access control or the lack thereof, the work of Zou et al. [164] describes northbound APIs' abuse due to the lack of an access control system multi-tenant architecture. For this, an access control system is proposed, formed by an authorization API, responsible for protecting northbound APIs' sharing in multi-tenant networks. The system is composed of a permission manager and an intermediary for runtime access. The permission manager abstracts permissions in a three-level structure, providing a permissions description language, with interfaces for permissions configuration and network connectivity verification of authorized users. The runtime access intermediary intercepts all calls to APIs to ensure validity and maintain user network topology information. To solve problems involving authentication, the work of Allouzi and Khan [10] discusses the lack of authentication that exists in the Openflow protocol so that both a controller and a switch can trust each other, every time one of the devices requests access to sensitive resources of the other device.

For this purpose, an extension of the handshake protocol of the Openflow specification is proposed by including a Negotiation Request message, which indicates which resources the device needs access to. If it is a sensitive resource, credentials are exchanged; otherwise, the resource is instantly released. For this, the resources are managed through an access control policy, represented in the Disjunctive Normal Form (DNF), where, for example, for an Rc1 resource of a Controller to be accessed. First either the Rs2 switch resource or the Rs3 resource must have been previously authorized for that Controller. G. Wu et al. [146] proposed a switch migration scheme by adopting a noncooperative game to improve the control plane scalability in SDN by designing a novel load balancing monitoring scheme to detect the load imbalance between controllers and trigger migrating switches. The authors then used a noncooperative game among controllers to decide switching migration to maximize profits.

In the Information Exposure category, the work of Conti et al. [31] proposes a secure path solution. In this sense, the work addresses the access to the flow table of a switch through several ways, such as connecting to an unprotected switch port, taking advantage of a poor switch configuration, such as a weak password, or misconfiguration, using the RTT variation to infer information about the flow tables, taking advantage of the lack of TLS in the control channel, exploiting backdoors inserted by device manufacturers or governments, or compromising the switch. With access to flow tables, the attacker can perform a series of attacks, such as infecting the network with a worm, scanning the network, performing a DoS, obtaining information about the mechanisms of detection and prevention of anomalies, through the injection of packets in the network and analysis of changes in flow tables, and correlating the settings of access control mechanisms with the flow rules to circumvent these mechanisms. As a security mechanism, the obfuscation of some data from the flow rules during the path of packets to the network output switch is proposed. The attacker cannot detect that the flow rules were impacted by its action and cannot obtain relevant information from the flow tables to perform attacks.

### 5.2 Mitigation techniques at the data plane

The network equipment responsible for routing data in the SDN concept must be secured from malicious applications that can install or modify rules on these _network elements. Therefore, security solutions for the infrastructure layer have been proposed in the literature to address these challenges.

Ahmad et al. [67] proposed a tool called VeriFlow, which allows to search for malicious two-way rules inserted by SDN applications on an Open-Flow switch. It also dynamically prevents fraudulent rules from reaching OpenFlow switches to maintain the integrity of the flow rules. The authors tested the tool on Mininet, an Open-Flow-compatible network simulation environment. The results show that VeriFlow can detect a new transfer rule in milliseconds by tracking routing data. FortNox [105] employs a security enforcement kernel (SEK) to enforce flow controls for active defense against various threats. FortNox is an enhancement engine responsible for executing and avoiding rule conflicts from separate security authorizations. FortNox uses two protection mechanisms:

- Rule prioritization, which ensures that any new flow rule contradicts the rules produced by applications, is overridden because of the highest priority: FortNOX defines three authorization roles by default among applications that make flow rule insertion requests. These roles may be enlarged with sub-roles, as needed when implemented. The first role is IT administrators, whose rule insertion requests are the highest priority within FortNOX's conflict resolution scheme, besides the highest flow rule priority traits sent to the switch. Second, security applications are allocated with a different authorization role. These security applications generate flow rules that may further restrict the administrator's static network security policy depending on newly perceived runtime threats, such as a malicious flow, infected internal equipment, a blacklist-worthy external host, or a surfacing mischievous combined traffic pattern. Flow insertion requests generated by security applications are defined with a flow rule priority under administrator-defined flow rules. Finally, non-security-related OF applications are assigned the minor priority. Roles are implemented through a digital signature structure, in which FortNOX configuration is preset with the public keys of several rule insertion sources. FortNOX increases NOX's flow rule insertion interface to include a digital signature per-flow request. If an ancient OF application does not decide to sign its flow rules, those rules are assigned the default role and priority of a common OpenFlow application.
- The conflict detection algorithm affects each new flow rule: to detect a divergence between a newly inserted OpenFlow rule and the existing OpenFlow rule set, the source IP and destination IP addresses, their ports, and wild cardmembers, all rules get converted, including the candidate rule, into a representation called alias reduced rules (ARRs), and then perform the conflict analysis on these ARRs. An alias reduced rule is simply a derivation of the flow rule in which we expand the rule's match criteria to incorporate set operation transformations and wildcards explicitly. An initial alias set is created, containing the first rule's IP addresses, network masks, and ports (where 0 (zero) represents any port). If the rule's action causes a field substitution via a set action, the resultant value is added to the alias set, which is then used to alter the criteria side of the ARR. Pairwise analysis of the candidate ARR to the actual ARRs representing the active ruleset. If there is an intersection between the source and addresses, the combination of the respective sets is used as the following rule's alias set.

VAVE [150] for Virtual source Address Validation Edge is a preventive protection program tested on an OpenFlow NOX controller to mitigate DoS attacks caused by IP address spoofing. When a new packet that does not match any rule in a flow table on a switch arrives, it will be sent to the controller for source address validation, or IP address spoofing may be detected. If the controller detects IP spoofing, it creates a rule in the flow table to stop the specific flow from that source address.

The failure resilience and recovery mechanisms change when the control scheme is decoupled from the data scheme. It is also shown that the path between an SDN controller and the switches is proportional to connectivity loss. Therefore, Zhang et al. [160] proposed an approach to shorten the path between the switches and the controller, to improve content availability for security applications, enabling fast recovery and security analysis.

FlowChecker [7] is a configuration verification tool to identify OpenFlow rules' inconsistencies in one or more switches. In other words, FlowChecker checks the consistency of the different switches and validates the accuracy of the flow table of the configuration deployed by the new services and protocols. It can be used as a centralized OpenFlow controller to receive OpenFlow applications' requests to analyze, verify or debug the configuration.

SDNsec [115] is a security extension to ensure packet routing rules' traceability at the SDN data plane level. In other words, ensuring that the flow rules are correctly applied at the infrastructure layer. The authors have proposed a mechanism to ensure that switches transmit packets according to the controller's instructions on the one hand, and on the other hand, to validate the packet path to allow the controller to reactively verify that the data plane has followed the specific rules. This solution guarantees consistency in updating the flow rules so that the data plan's behavior is well defined during reconfigurations.

*AVANT-GUARD,* proposed by Shin et al. [125], is a defense platform against saturation attacks on the link between the data plane and the control plane. AVANT-GUARD allows the control plane to be more resistant and scalable against attacks that cause the control plane's saturation, such as TCP-SYN (DoS) flood attack. In this work, the author aims to avoid saturation of the switch-to-controller bandwidth and increase the controller response rate to requests from the data plane switches. Shin et al. implement two new modules in the data plane; the connection migration and the actuating trigger. The first module protects the controller from saturation during TCP-SYN flood attacks by limiting the flow requests sent to the control plane. It inspects TCP sessions at the data plane before sending them to the controller. The second module, called "actuating trigger", increases the controller's reactivity and the data plane (the OF switches). It uses triggers to collect network statistics and report all existing conditions in the switches (the storage capacity of the TCAM tables, the number of installed flow rules, the load in the queues, etc.) to the controller to respond to threats. The results show that AVANT-GUARD security extensions reduce the control plane load and use the

switch flow policy table in the data plane. However, only TCP flows are considered. The proposed approach is not designed to prevent attacks based on other protocols such as UDP or ICMP.

Moreover, to ensure strict security, the packets in the flow interact with the controller only after passing the mechanism for establishing a TCP connection. This method can help detect malicious flows, but it does not consider the data plane's processing load when attacking DoS. This solution can increase the response time for legitimate flows. In addition, to implement this solution, it is necessary to make changes to the OpenFlow switch specification to manage the new modules' operations.

Security Enhanced-Floodlight (SE-Floodlight) (Phillip [98, 106]) extends the Floodlight SDN controller that aims to create an ideal secure control layer for SDN. The SE-Floodlight controller provides privilege separation mechanisms by adding a secure Northbound API to the controller that mediates between applications and the data plane. Introducing a new OpenFlow audit subsystem on SE-Floodlight allows tracking all security events occurring at the control layer. Gao et al. [49, 50] proposed a solution, categorized in Middleboxes, to protect the network against attacks of the Poisoning and Flow Table Overflow categories, where the proposed solution is composed of six functional modules: traffic monitor, host status monitor, controller, control application abstraction, attack detection, and audit server. The traffic monitor module functions as the data plane and runs on network hardware. It processes and monitors both inbound and outbound traffic based on the flow rules in its flow table. The host status monitor module is an application that monitors host information. This application provides host information to the control abstraction module to enable application-level management and perform accurate attack detection. The control application abstraction module is an intermediate layer between the Controller and the control applications. This module collects host and network traffic information and associates each packet with host information. In addition, the control application abstracts the Controller implementation language to a high-level language, providing user-friendly interfaces to update network security policies dynamically. The attack detection module is a pre-installed control application, which identifies malicious traffic based on hosts and network traffic. The audit server is a device to detect whether a poisoning attack is attacking the control plane. The audit server checks the flow rules in the traffic monitor. The server collects host and traffic information periodically and uses the same database and attack detection algorithms as the attack detection module to check the legality of the flow rules. Chen et al. [24] combat the poisoning attack through a security architecture formed by several middleboxes (Firewall, IPS, IDS) distributed over the network, in the form of virtualization of the network functions. Another work of the DoS category aims to protect the network specifically against Poisoning and Fingerprinting attacks through a security Framework [49, 50]. The work proposes to combat the attacks through two modules: a traffic agent and a global vision agent. The traffic agent acts between the data and control planes, processing some traffics, so the Controller's workload. For this purpose, the traffic agent identifies and filters packets from a DoS attack and injects a delay in some packets (thus eliminating the possibility of the attacker discovering some information based on traffic time). Besides, the agent verifies the existence of hosts by sending probe packets (thus combating Poisoning attacks). The global view agent is a Controller application that provides global network information to the traffic agent and passes on valid flows from the traffic agent to other Controller applications. Alasadi et al. [8] proposed to combat Poisoning attack through a security architecture that decreases host discovery messages, avoiding DoS attacks that exploit messages such as ARP and DHCP. The controller and the servers of these types of service share the responsibility of answering these messages, depending on the type of service requested. For the same service, several input ports are directed to the same output port, thus reducing the number of messages circulating in the control plane, and consequently, the flows stored in the flow tables. With this solution, broadcast messages are eliminated since each service has a predetermined server as a destination.

### 5.3 Mitigation techniques at the Control Plane

Control plane security solutions can be applications or approaches to protect the control plane from threats such as DoS, DDoS, malicious applications. The SDN controllers' programmability makes it very easy for network administrators to install security applications on network equipment through the controllers, using the Northbound APIs. As a result, the applications inform the controllers to operate the network devices they control as security policy enforcement devices.

To deal with these types of attacks, especially denial of service attacks on an SDN controller, it is necessary to analyze the traffic characteristics of the flows stored in OpenFlow switches. Braga et al. presented in Braga et al. [19] a detection method that consists of self-organizing maps (SOMs) to identify abnormal/injected flows, as shown in Fig. 10. The proposed solution consists of three modules:

- A flow Collector module that is responsible for gathering flow from switches.
- A feature Extractor module that extracts relevant data distinguishing DDOS attacks.

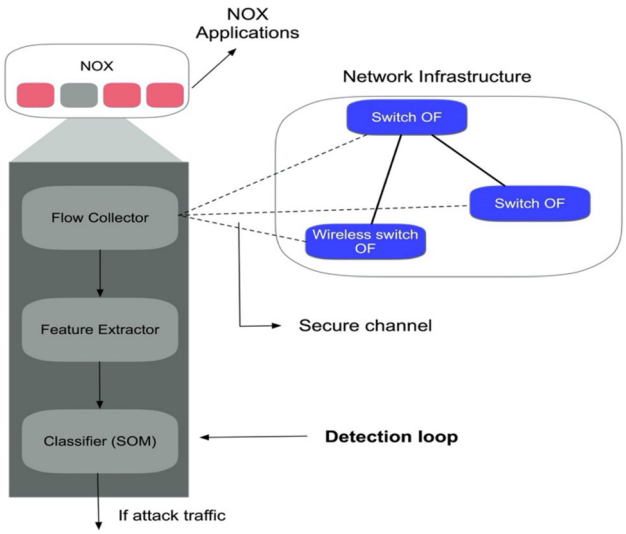


**Fig. 10** Self-organizing maps (SOMs) SOMs detection process

- A classifier module analyses extracted data then classifies them as normal or abnormal.

To improve the processing capacity of the controllers, McNettle [139] has been proposed. McNettle is a multiple processor core SDN controller developed for large networks supporting resource-intensive control algorithms. It is performance efficient and scalable to handle up to 46 processor cores. Network administrators can program the McNettle extension with a high-level programming language as needed.

Nguyen et al. [93] presented an extension to the SDN controller. It is an automatically secure controller from attacks on tracking services.

The extension relies on three factors as follows:

- Port Manager: It is responsible for determining the host producing the traffic. It also detains a list of hosts mapped with corresponding MAC addresses.
- Host Probing: Its primary role is verifying if the host is reachable or not by generating an ICMP echo request.
- Host Checker: To verify if the host can be migrated and prevent ARP poisoning.

FloodGuard is a defense platform against DoS attacks in SDN networks proposed by Wang et al. [141]. The authors' objective is to avoid saturation of the switch tables of the transmission plane switches and controller overload. They implement two supervision and flow management modules in the controller to achieve these objectives. The first is a proactive switch rule analysis module that generates and installs switch rules during the DoS attack. The second is a packet migration module that manages flooded traffic without losing legitimate traffic. When a DoS attack is detected, the packet migration module redirects packets to a cache installed in the data plane. Subsequently, the Proactive Switching Rules Analysis module will generate switching rules based on the current state, such as the data in the network and the links' state. At the same time, the data plane cache starts managing the already collected packets by sending "packet_in" messages to the controller with a limited send rate using the round-robin algorithm (Round Robin—RR) as a packet scheduling technique (Luc and De Mey 2017). When the flow rules are ready, the controller assigns them to the switches. The results prove that this technique effectively decreases the bandwidth utilization rate between the controller and the switch and avoids the saturation of its TCAM memory. However, FloodGuard does not avoid overloading the control plane, because it will increase the controller's processing load.

DISCO is a multi-domain controller proposed by Phemius et al. [104] to provide security functions at the control layer for distributed heterogeneous networks. It is composed of an inter-domain control module and an intra-domain control module. The inter-domain control module supervises and manages the data flow priorities between the domains to calculate and transfer flow paths with different priorities. Thus, the Inter-Domain Control Module can dynamically redirect or stop the flow to deal with attacks on the control plane. The intra-domain control module manages the exchanges between controllers. It has a transceiver and agents to search for neighboring controllers and provide a control channel between controllers. The agents exchange network information through the communication channel provided by the transceiver module.

SLICOTS is another solution proposed by Mohammadi et al. [86] to mitigate the impact of TCP-SYN flooding attacks in SDN networks. It is presented as a security module implemented in the control plane. This module monitors all incoming TCP requests to the network to detect and prevent TCP-SYN flooding attacks and block malicious hosts. SLICOTS installs temporary flow rules for each TCP connection request during the TCP connection process. After validating a connection, it establishes permanent flow rules between the client and the server. In addition, SLICOTS blocks the attacker who sends a large number of half-open TCP connections. The proposed solution is an extension module to the OpenDaylight controller [97]. Simulation results showed that SLICOTS is an effective solution to mitigate SYN flooding attacks' impact by reducing the attack detection time, minimizing the flow table's size and the controller load. This solution offers a low response time for legitimate requests. However, during a DDoS attack, the time spent before malicious traffic is detected and blocked can overload the controller and the data plane. Besides, switch TCAM tables may be saturated during the detection period due to many temporary flow rules installed. Lei Wei and

Carol Fung [143] proposed FlowRanger, a flow storage system that can detect and mitigate DoS attacks. FlowRanger is implemented in the SDN controller. This system consists of three main components. The first is a trust value management component that calculates each packet-in message's trust value by verifying its source. The second one is responsible for managing the controller's file wait, which will assign switching rules to each incoming packet. This component puts the packet-in messages in file in priority according to its trust value already calculated by the first component. And the third is the message scheduling component, which processes messages according to the "Weighted Round Robin" (WRR) strategy (Brocade Communications Systems 2016). The FlowRanger technique can reduce the impact of DoS attacks on network performance by ensuring that legitimate flows are served first in the controller. However, this solution does not prevent controller overload and switch table overload.

SDN's centralized nature is revealed to be a single point of breakdown that can be exploited by one of the Internet's most old, high risk and significant security threats known as Distributed Denial of Service (DDoS) attacks. A DDoS attack is a dispersed and harmonized attack that begins from multiple network devices. Essentially, this attack's strategy is to send a considerable volume of spoofed IP packets from disparate points to make the network resources unattainable to legitimate users. Over recent years, the attackers have become more intelligent and continually enhance and use advanced DDoS attack methods to inflict more economic and financial costs. Kuerban et al. [74] offered a FlowSec approach to prevent DoS attacks on the SDN controller. FlowSec calculates the collected controller bandwidth statistics automatically. If the attack is found, the switch will be enforced to reduce traffic using the Floodlight module (Floodlight, n.d.), responsible for collecting switch statistics. Then, Suh et al. [134] designed a Content-oriented Networking Architecture (CONA) and clarified how it works on the NetFPGA-OpenFlow platform. An access router can determine which content gets pulled from attacked hosts in CONA. The CONA service receives the hosts' content request and responds with corresponding content. Thusly, CONA can block threats that consume network resources like DDoS attacks. Since the controller is a single-point failure, TCP SYN FLOOD threats (Type of DDoS attack) can attempt to compromise the controller. Fichera et al. [43] introduced OPERETTA. It is an OpenFlow-based resolution to TCP SYN FLOOD threats. OPERETTA allows TCP SYN packets to reach into the controller and block bogus connection requests. Dridi et al. [39] suggested SDN-Guard. The main objective of this work is to design and develop a new solution that would protect the SDN network against DDoS attacks and mitigate their impact on network performance. This solution can simultaneously reduce controller load, congestion of the link between the switch and the controller, and avoid overloading the switch tables of OpenFlow switches. As a result, SDN-Guard is designed to simultaneously alleviate the problems that have already been moved by dynamically managing flow paths, switch rule usage time in the switches and switch rule aggregation. Figure 11 presents SDN-Guard architecture. SDN-Guard's decisions are made based on the flow threat probability provided by an Intrusion Detection System (IDS) that analyzes flows and detects activities malicious in real-time.

De Assis et al. [14] proposed a stand-alone DoS / DDoS defensive approach for SDN called Game Theory (GT) Game Theory, which is an analytical tool that can model negotiation situations and deal with many problems in different areas -Holt-Winters for Digital Signature (HWDS), which combines detection and anomaly identification provided by an HWDS system with a decision-making model based on GT. Yunhe et al. introduced in Porras et al. [98] a system named Software-Defined Anti-DDoS divided into four modules detection, a Trigger Module, Detection Module, Traceback Module, and Mitigation Module, respectively. These modules correspond to four states:

- **Init State**: It is the primary state of the SD-Anti-DDoS system. A packet_in trigger is used to program attack detection in this state. If no abnormal event were found, it would remain in the Init State. Else, the system will run a Detection State.
- **Detection State**: This state handles the detection of a DDoS attack. The attack detection will start to verify the existence of a DDoS attack in the network.
- **Traceback State**: If the detection module detects a DDoS attack, the Traceback State will start, and the system will try to trace the attack path and the attack origin switch.
- **Mitigation State**: The Mitigation module will stop the threat from its source and clean the infected switch from malicious flow entries.

Figure 12 Illustrate the four states of the above Anti-DDoS attacks.

Another DoS Category work aims to protect the network from Flow Table Overflow, and Control Plane Saturation attacks through a security architecture to resist attacks in SDN cloud environments [34]. The Flow Table Overflow attack is prevented by inspecting packets arriving at the switches. Control Plane Saturation attack is prevented through a multiple Controller architecture, minimizing Poisoning attacks. The architecture also has an authentication system to mitigate a malicious user from entering the network. This authentication occurs through a digital signature with the SHA3 hash algorithm. The Controllers' location is defined through the composition of a genetic algorithm and a population-based meta-heuristic algorithm (Cuckoo Search).

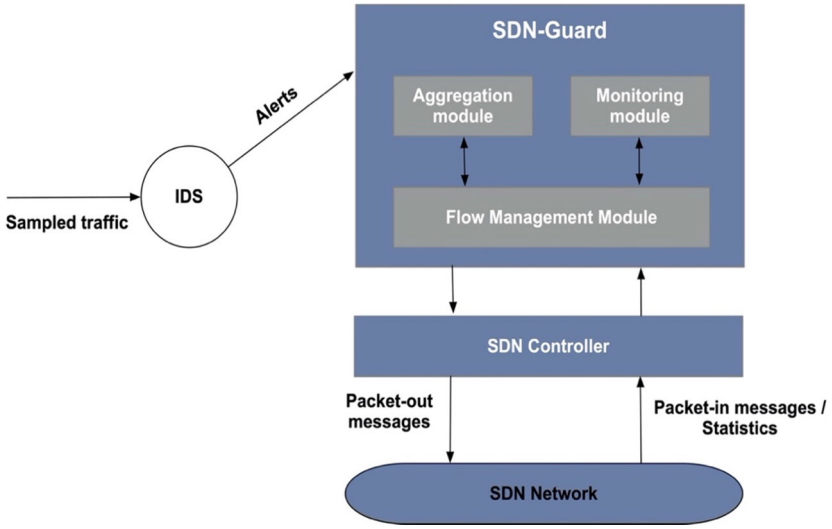


**Fig. 11** SDN-Guard architecture

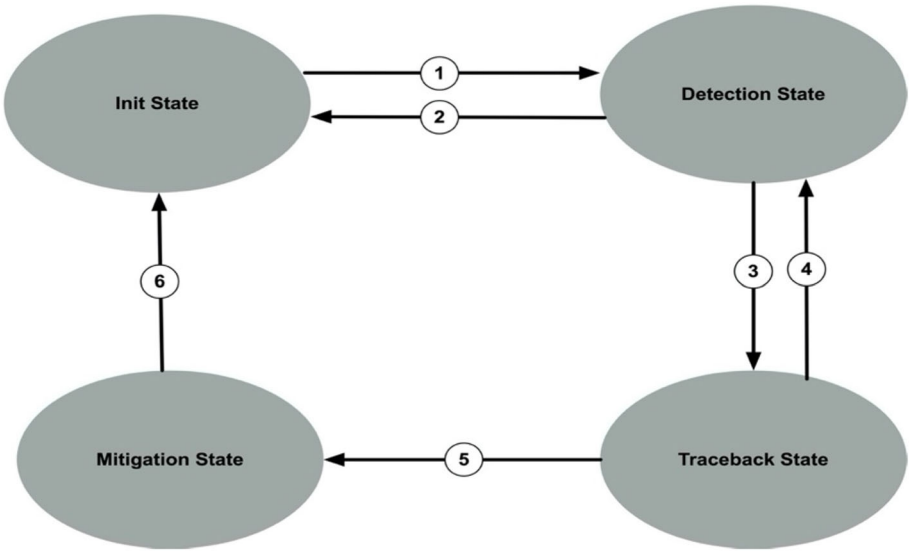


**Fig. 12** SD-Anti-DDoS four state diagram

In addition, a routing protocol based on the PSO (Particle Swarm Optimization) algorithm is used. PSO is a population-based optimization algorithm formed based on the social behavior of birds and fish searching for food. The algorithm considers node congestion, link congestion, and node delay for route definition. The work of Zhu et al. [163] addressed the DoS attack between different network domains. And for protection against this type of attack, it proposes an anomaly detection scheme formed by two servers, a server responsible for processing the data sent by the domains and another server responsible for detecting the attack. The scheme uses the adapted kNN (k Nearest Neighbor) algorithm to classify the data and detect the attack. Disruptive encryption is used to protect the privacy of the different network domains. Communications between these two servers and between the servers and the SDN Controllers occur via TLS. Each domain produces a 7-tuple from the flow table data. SN is a serial number generated sequentially,T is the flow timestamp; MPF is the average number of packets per flow. MBF is the average number of bytes per flow. PCF is the flow correlation percentage (considering two flows, where the source IP address of one flow is the same as the destination IP address of the other flow, and vice versa, using the same protocol. PFC is the sum of the flows that fit this situation, divided by the total flows); GOP describes the growth rate of the number of ports within a fixed interval; GSI represents the growth rate of the number of sources IP addresses within a specified interval. Besides, the domain generates a perturbation parameter and encrypts the 7-tuple with that parameter. The encryption result is sent to the processing server, and the perturbation parameter is sent to the detection server. Then, the processing server applies a part of the kNN algorithm and sends the processing result and the sevenfold to the detection server. The detection server decrypts the 7-tuple with the

perturbation parameter and uses the processing result with the 7-tuple to complete the kNN algorithm's execution. If the execution detects an attack, the scheme sends an alarm (formed by SN and T). Another work aims to protect a server against DoS through Moving Target Defense [28]. Where this work formulated a scoring system based on vulnerabilities and IDS alerts to trigger a countermeasure through the technique of continuously changing the port number of a particular service, making it difficult for the attacker to define where to attack. The work of Nagai et al. [91] aimed to combat DoS with authentication to protect a server against a TCP SYN message flooding attack through a TCP packet authenticator based on Openflow. When a switch receives a TCP packet with an SYN message, the switch forwards that packet to the Controller. Upon receiving this packet, the Controller generates an invalid SYN + ACK packet, forwards it to the switch, and inserts a flow into a table in the authenticator called CHECKING_TCP. The switch sends this packet to the client, which causes the client to send an RST message. The switch receives this message and forwards it to the Controller. The controller, in turn, checks whether the data in the packet matches any flow in the CHECKING_TCP table. If it matches, the authenticator registers the flow in a CHECKED_TCP table and logs the switch's flow, releasing access to the server. If the client sends another SYN message, the Controller checks that a flow already exists in the CHECKING_TCP table matches the packet data, thus discarding the packet and preventing a DoS attack. In the vulnerability involving tampering facility, Mohan et al. [87] aim to protect against the Interception category attack through a Safe Path category solution. Another solution to the vulnerability organized in the Tampering Facility category is proposed by Li et al. [77]. This work addresses an attacker's ability to alter packets in the network. It proposes a dynamic packet forwarding verification mechanism by sampling packets and collecting flow statistics at irregular intervals as a protection mechanism. These intervals are dynamically adjusted based on the results of previous verifications. The mechanism detects suspicious cases by identifying attack patterns, such as if a packet cannot be successfully verified or when flow statistics are inconsistent with expected packet forwarding behavior. When this occurs, the mechanism increases the rate of packet sampling and flow statistics collection to confirm an attack more quickly. An example of a detection technique of the mechanism occurs when a packet enters the network. The respective switch forwards that packet to the controller through a PACKET_IN message. The mechanism generates a MAC (Message Authentication Code) for the entire packet, and stores the MAC in a hash table, along with a TTL (Time To Live). Suppose the PACKET_IN message comes from a destination-connected switch, and the packet's MAC is stored in the hash table with an unexpired TTL. In that case, the record is removed from the table, ensuring that the packet has not suffered an attack that delays packet traffic. Another example of detection occurs if a packet is traversing the network.

Consequently, the MAC of that packet is being compared in the hash table after arriving at each switch in the route. At a certain point, the MAC does not match any record in the hash table. The mechanism concludes that the packet was tampered with during traffic. Another detection example occurs when the flow statistics do not show the same values on all switches of a route, and the mechanism concludes that the switch showing different statistics is a suspect switch. Phan and Park [103] propose an efficient solution to tackle DDoS attacks in the SDN-based cloud environment. The authors proposed a new hybrid machine learning model based on support vector machines and self-organizing map algorithms to improve traffic classification. The model introduces an enhanced history-based IP filtering scheme (eHIPF) to improve the attack detection rate and speed. Finally, the authors present a novel mechanism that combines the hybrid machine learning model and the eHIPF scheme to make a DDoS attack defender for the SDN-based cloud environment. Qasmaoui et al. [108] proposed an enhanced SOLID-FLOW to improve flow rules database integrity inside the SDN controller. According to SDN's specifications [133], the controller is the only entity responsible for the decision to route flux in the network. At the first phase, the flow rules are dispatched to the underlying infrastructure, containing network equipment such as routers and switches. The network administrator predefines those flow rules. The switches use their flow tables to forward packets to the destination in upcoming packets. On the other hand, when the switch receives a packet belonging to a new flux, it asks the controller to dispatch the flow rule matching the new destination. The controller responds with the corresponding recent rule. The switch forwards the packet to the next node. The controllers' role is crucial to good networking infrastructure. If the controller is compromised or disturbed, the totality of the network becomes paralyzed.

The controller's main components remain the flow rules management module responsible for adding, modifying, and deleting flow rules from the controller storage module. An unauthorized entity can alter the existing flow rules leading to a dispatch of packets to a man in the middle or even deleting flow rules, which paralyze the network resulting in a denial of service. Protecting the controller flow rules management component is essential to a more secure and stable SDN network. The authors tried to solve one of the most critical security issues of the SDN controller by reinforcing SDN Flow rule management's integrity inside the SDN controller. They propose an enhanced SOLID-FLOW, which improves flow rules database integrity inside the SDN controller, as shown in Fig. 13.

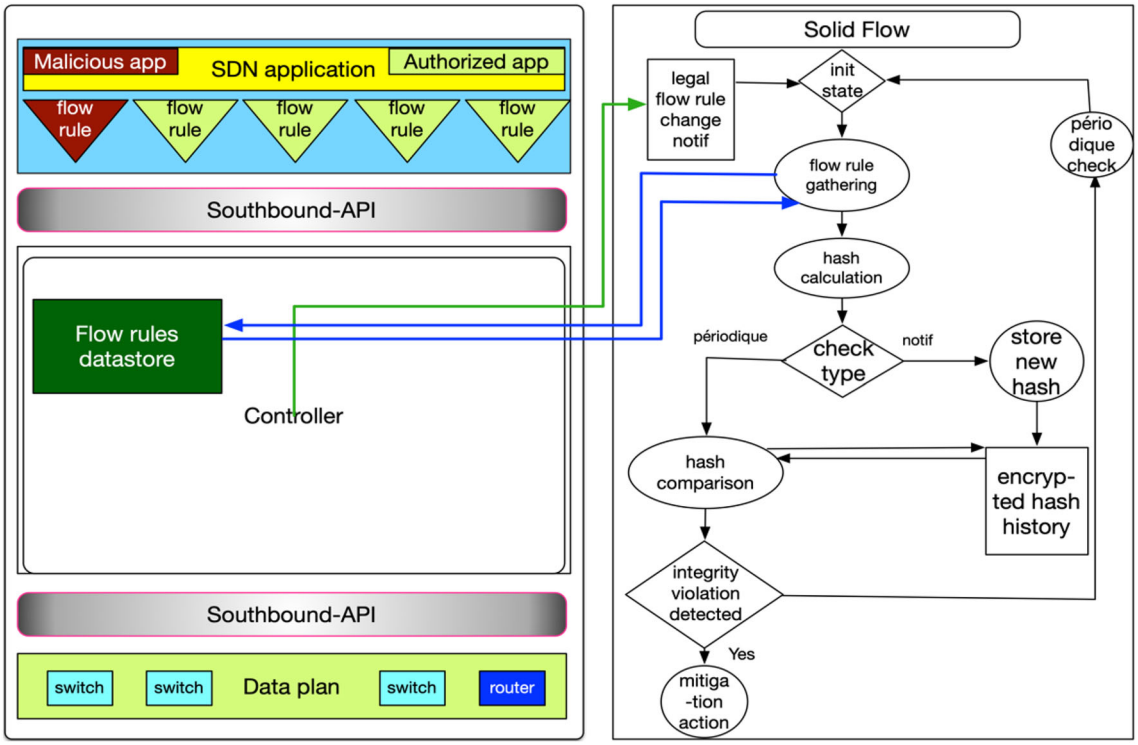


**Fig. 13** Enhanced solid-flow architecture

## 6 Discussion and comparison

As a result, centralized network management is available to the network controller and the applications integrated. The SDN architecture provides an opportunity to improve existing and create fundamentally new means of information protection. A wide range of research examines methods of protecting OpenFlow networks from specific, specific to Open-Flow threats to information security. In addition, means to counter traditional network threats, which can be prevented by using the features of the OpenFlow protocol, are being improved. An effective attack prevention method for OpenFlow networks is pre-detection of network configuration inconsistencies, which can be achieved by applying verification models. Al-Shaer et al. [7] described FlowChecker, which uses binary detection diagrams to divide network resources into layers. Khurshid et al. [67] propose to check traffic flow processing policies using modular theory. The VeriFlow project implements a real-time flow rule checking policy. A utility has been developed that intercepts flow rules sent by a network controller to an OpenFlow switch to analyze those flows further. Schlesinger et al. [21] propose a technology to verify the isolation of traffic of a particular application, dividing the network infrastructure into layers to achieve confidentiality and data integrity. It is important to note that SDN features can improve the traditional approaches to control traffic flows. Liu et al. [78] propose an original approach to network management in which traffic flows are allowed or denied based on source and destination security levels. Yao et al. [150] propose a network architecture with mandatory sender address verification for each data source. The network controller performs the verification in reactive mode for each packet transmitted to the controller due to non-compliance with any flow table rule. Jafarian et al. [61] present Moving Target Defense (MTD) implementation for networks based on the OpenFlow protocol. MTD is implemented by hiding real internal host addresses for external sources. The controller assigns virtual IP addresses to network hosts and performs translation while forwarding. Due to the unpredictable distribution of virtual addresses.

Several papers investigate authentication and authorization mechanisms at the application level. Mechanisms adapted for differentiation of access of different network services are proposed. The FortNox framework resolves conflicts related to conflicting rules from different sources. It is an extension for the NOX controller, which checks flow rules before applying them to network devices. Shin et al. [124, 125] have developed a framework for OpenFlow network security control (FRESCO) that allows information security-related applications to be integrated into OpenFlow networks. Wen et al. [144] present the PermOF control system, which aims to give OpenFlow applications the minimum rights necessary to prevent abuse. Since the most probable area of SDN implementation in RF is cloud technologies, the most serious

security threat from the commercial point of view is "denial of service" (DoS-, DDoS-attacks), this topic deserves special attention. It is fundamentally important to note that OpenFlow technologically allows you to use the SDN network as a sensor, because information about the flows can be obtained from all critical points. The AVANT-GUARD controller extension [132] optimizes the use of network resources by reducing the amount of control traffic sent between the controller and switches to mitigate DDoS attacks. One of the most effective protection methods against DoS attacks in networks of traditional architecture is intrusion prevention systems (IDS). Several researchers have suggested architectures or implemented experimental IDS based on the principles of OpenFlow networks. Hu et al. [58] developed an IDS that analyzes events in an OpenFlow network with multiple controllers; the architecture of the developed system allows the analysis of many traffic flows, based on intrusion detection rules configured by the network engineer. NICE [30] is a mechanism for detecting DoS attacks in OpenFlow networks, based on an analytical model of attack graphs. Moazenni et al. [85] propose a Reliable Distributed SDN (RDSDN) method to improve the fault tolerance of SDN controllers. An architecture of multiple distributed controllers is used, where each controller is responsible for one subnetwork as a master, and defined as a slave for the other subnets. A new formula is proposed to calculate the reliability rate of each subnet. Thus, the calculated reliability rates are shared among the controllers, to select the controller with the highest reliability value, chosen as the coordinator for the network. In RDSDN, the failure of the controllers is detected by the coordinator, who decides which other controller is the most appropriate to take over the subnetwork whose controller has failed.

Each of the above protection methods against attacks was evaluated in terms of its strengths and weaknesses. The solutions were analyzed to identify the gaps that had not yet been filled from the point of view of SDN security. To summarize the methods analyzed above, we created a table showing each of the solutions, the type of attack they are trying to protect against, the logical level of SDN at which they are working, and the security aspects they are trying to maintain. As SDN vulnerabilities are still in the discovery and solution phase, several works that answer RQ1 (main vulnerabilities) do not yet have a defense mechanism. Several works that answer RQ2 (main defense mechanisms) exploit SDN's benefits to solve security problems in traditional networks, thus not directly correlated with the vulnerabilities categorized in this paper. Table 6 examines possible attacks on all software and hardware network levels and analyzes protection systems. The structure of these systems, their advantages and limitations were analyzed, a comparative characteristic of protection methods was carried out. It was shown that the general problem with the proposed protection solutions is that they are designed to protect against only one particular type of attack and do not provide for general protection in network settings. To ensure a protected software configuration network, it would be worth considering the parallel implementation of combinations of the proposed schemes. This can lead to a large number of processing costs and the creation of conflicting configurations. These problems lead to the possible emergence of additional variability and obtaining compromising solutions when scanning the network for attacks. However, it is clear that by adding additional protection mechanisms to the controller, the productivity of the network in terms of its throughput capacity can be negatively changed. To improve safety in SDN networks, it would be important to balance security and productivity by making a critical, network-wide safety mode. Therefore, developing a comprehensive system that would simultaneously protect against several attacks is necessary.

## 7 Open research issues, challenges, and directions

Today, it is extremely important to protect any system against intrusions and unauthorized access, as uses have evolved with the arrival of connected objects. Security has become an essential and crucial component of any network solution. Fortunately, research opportunities are emerging with SDN technology. The functionality provided by SDN controllers can easily scale in terms of processing capacity, taking advantage of the benefits offered by cloud computing.

As discussed, the existing solutions suffer from limitations that hinder their real-world application. We expect future research to propose more advanced and practical protection mechanisms based on existing limitations. Specifically, none of the existing solutions consider the most recent proposals, including stateful data planes. Hence, one of the immediate future required extensions is to evaluate the applicability of existing solution for stateful forwarding devices and propose solutions to ensure if any such device is fully, or partially, compromised it can be detected. Another direction of research that we expect to be very active in the future is designing secure hardware for SDN-enabled forwarding devices based on the latest software advances and requirements.

SDN architecture significantly changes the structure of the network. Therefore, there are new security threats caused by vulnerabilities in individual infrastructure components. In addition, most of the threats associated with traditional data networks are critical to the same or greater extent in the context of SDN networks. On the other hand, SDN architecture offers opportunities for innovation in developing security

**Table 6** Summary of proposed security measures for SDN layers (planes)

| SDN layer (Plane) | Security projects | Targeted threat | Main contribution | Mitigation type | Limitations |
|---|---|---|---|---|---|
| Control Plane | HyperFlow [135] | Controller availability | Provides a scalable logically centralized but physically distributed control platform based on event propagation | Detection framework | The additional delay when a controller converges or reaches network synchronization increases the response time |
| | McNettle [139] | Controller scalability | Proposes a multiple processor core SDN controller developed for large networks supporting resource-intensive control algorithms | Control event processing throughput scales | Control modeling greatly impacts the network scalability |
| | Hybrid controller [42] | Controller scalability | Evaluates OpenFlow controller performance | Traffic behavior and set a path on-demand | Management of heterogeneous control plane is difficult |
| | AVANT-GUARD [125] | DDoS | Reduces communication overhead between data and control plane, fast response to changing flow dynamics at the data layer | Control plane saturation attack | The problem is that Avant-Guard can only handle TCP-based attacks and introduces latency for legal SYN packets |
| | DISCO [104] | Controller scalability | Provides security functions at the control layer for distributed heterogeneous networks | Distributed controller architecture | The main challenge is the overall consistency of views across all controllers |
| | Floodguard [141] | DoS | Provides a scalable, lightweight security solution for SDN networks using two modules: Proactive Flow Analyzer and Packet Migration | Proactive flow rule analyzer | The buffer saturation attack between the control-data plane |
| | FlowRanger [143] | DoS/DDoS | Proposes an intrusion detection and prevention system | window size and threshold | The use of the demand planning algorithm significantly increases the performance of FlowRanger |
| | LineSwitch [12] | DoS | Employs probabilistic proxying and blacklisting of network traffic | Control plane saturation attack | Lineswitch has performance, overhead and deployment challenges |
| | [86] | DDoS | Collaborative detection and containment mechanism of network attacks | DDoS Flooding Detection and Containment | Network intrusion detection that is challenged by large volumes of data and complex network topologies |

**Table 6** (continued)

| SDN layer (Plane) | Security projects | Targeted threat | Main contribution | Mitigation type | Limitations |
|---|---|---|---|---|---|
| | FLEXPROTECT [24] | DDoS | A flexible distributed DDoS protection architecture for multi-tenant data centers | Distributed controller architecture | There are two non-trivial issues for FLEXPROTECT, the delay in final attack mitigation actions, and the need to ensure sufficient resources to deploy a robust virtual anti-DDoS infrastructure |
| | RADAR [161] | DDoS | Suspicious traffic is throttled and attack traffic is dropped using port-based max–min fairness technique | Adaptive correlation analysis on COT SDN switches | New emerging sophisticated DDoS attacks (e.g., Crossfire) constructed by lowrate and short-lived "benign" traffic are even more challenging to capture |
| | DYNAPFV (Q. [77] | Packet forwarding | Adjusts dynamically the rates of packet sampling and flow statistics collection based on the prior detection results to preserve the verification accuracy | Forwarding verification mechanism | Can't address passive (or even subtle active) reconnaissance |
| | SoftGuard [147] | Low-rate TCP attack | Installs flow rules on switches to detect low-rate TCP attacks, and performs mitigation on ingress switches | Detection framework | Processing load on the controller |
| | Enhanced Solid-Flow [108] | Malicious Flow Rules | Proposes a new security mechanism based on hashing algorithms and authorization to handle flow rules insertion | Flow rules database integrity | Effective only against flow rules integrity |
| Data Plane | FlowChecker [7] | Faulty flow rules | Proposes a configuration analyzer of OpenFlow switches | Configuration verification tool | Bad effects on performance—Does not check entire network |
| | Monitoring Function Openflow [66] | Data plane fault recovery | Monitors function on OpenFlow switches | Place a general message generator and processing function | Processing load on the controller |

**Table 6** (continued)

| SDN layer (Plane) | Security projects | Targeted threat | Main contribution | Mitigation type | Limitations |
|---|---|---|---|---|---|
| | VeriFlow [67] | Faulty flow rules | Flow rules checker | Prefix tree (an ordered tree data structure) | Bad effects on performance |
| | FortNOX [98, 106] | Flow rules contradictions | Provides an authorization mechanism based on digital signatures and security constraint thanks to an extension on a NOX controller | Controller framework | FortNOX records rule relations in alias sets, which are unable to track network traffic flows accurately |
| | Packet-In Filtering [72] | Packet-In messages | Filtering the mechanism on OpenFlow switches | Extend the OpenFlow specification | CPU loads in the switches/ |
| | ISAVA (C. [153, 154] | DoS | Provides an SDN-based integrated IP source address validation architecture | Packet signature signing and verification protocol between AS alliance | It complicates system implementation as it has to modify the client's host-stack |
| | DPX [101] | Packet detouring | Supports security services as a set of abstract security actions that are then translated to OpenFlow rule sets | Frequent packet parsing | DPX cannot directly enable distributed solutions that span the whole network |
| | TMPTCP [145] | The multipath transmission control protocol (MPTCP) vulnerabilities | Combines the SAV technology and the conventional MPTCP | Source Address Validation (SAV) | Switch computing overhead |
| Application Plane | SE-Floodlight [98, 106] | Applications authorization | Conflict resolution, authorization, audit system | Secure controller architecture and secure AppCtrl API | The design presentation also does not discuss the administrative responsibilities in configuring and operating a secure control layer in an operational deployment scenario |

**Table 6** (continued)

| SDN layer (Plane) | Security projects | Targeted threat | Main contribution | Mitigation type | Limitations |
|---|---|---|---|---|---|
| | FRESCO [124, 125] | Threats within/from applications | Defines constraint-based actions for flows such as drop, forward, redirect, mirror, network separation, etc | Security applications development frfamework | Leads to many issues of reliability and scalability |
| | PERM-GUARD [142] | Hijacked/Rogue Controller | Cross-layer protection framework | Flow rules | Controllers might potentially become a bottleneck for the network operations |
| | PermOF [144] | Access control | Provides a resource isolation and access control | Minimizes SDN application privileges | Leads to many issues of reliability and scalability |
| | FLOVER [132] | Security policy violation | Proposes a flow rules checker | Utilizes the satisfiability modulo theories | VeriFlow boasts low-latency of the checking process, it cannot handle multiple controllers |
| | sFlow Security-centric [5] | DDoS, DNS Amplification Attack | Proposes a substitute solution via sFlow with security-centric SDN to timely detect and reasonably mitigate DNS amplification attack | Security-Centric SDN | The detection is susceptible to delays in the same way |
| | Assertion [48] | Flow rules contradiction | Proposes an SDN application debugger | VeriFlow verification algorithm | Do not support complex packet processing pipelines |
| | SDNSOC [29] | Flow rules Conflict Detection | Proposes an object oriented programming framework—SDN Security Operation Center (SDNSOC), which handles policy composition | Policy Conflict Detection | Does not consider indirect security policy violation detection |
| | SAIDE [57] | Application Interference | Combines the action and refactor fields of policies to detect the application interferences | SDN Application Interference Detection and Elimination | Cannot handle multiple controllers |

tools. The combination of centralized management and programmability of the network makes it possible to improve the efficiency of network security tools.

The presence of a centralized network controller, which is critical for the normal functioning of the network, entails vulnerability to DoS attacks directed at the controller, which can lead to the failure of the entire network infrastructure. In addition, scanning attacks are often the precursors of targeted DoS attacks. They can also involve specific vulnerabilities in the controller software, making them critical within SDN networks' framework. One of the most effective countermeasures against DoS and scan attacks in traditional data networks is intrusion detection systems. They can protect against threats specific to SDN, but this will require adapting analysis methods and developing new rules and threat patterns for the most effective detection. However, most threats in SDN networks are inherited from traditional networks, which implies that traditional IDS can be used without significant modifications.

Between the application layer and user interactions, the SDN control flows deal with personal information and privacy. Privacy-preserving procedures can be executed on SDN plans, and some of the privacy-preserving methods have been assumed by challenge, which may be validated using model checking tools. Li and Jin and Ahmad et al. It is vital to evaluate safety standards in control flow and data flow procedures.

A fault-tolerant network approach might be capable of returning the network to a stable operating state in the occurrence of an incident that leaves the network disabled or unable to work properly. Remarkable network incidents include controller crash or shutdown connectivity disruption and energy outages [38]. SDN deployments are prone to suffer any of those incidents, particularly when the network is under a certain specialized attack vector. As perceived in SDN security state-of-art, current research in SDN security focuses on predicting, preventing, protecting, detecting and reacting to security issues, but very few proposals include mechanisms to recover the network state after a successful attack, only in [114] path monitoring detects congested links in DoS/DDoS attack situations and a path computation scheme redistributes the congestion flows in alternative network paths. Despite the single point of failure problem inherent to the SDN paradigm, insufficient works approach this problem or propose recovery schemes for controller outage. A complete security system might be proposed as a compound architecture that includes both proactive mechanisms that comprise the front line to act against the situations that compromise the security. And reactive mechanisms as well, to enforce actions that focus on recovering the network state once it has been compromised.

It was previously pointed out that network controllers can neither distinguish when network applications demand the installation of malicious flow rules in the infrastructure nor when certain applications interfere in the packet handling process exploiting vulnerabilities in the northbound APIs, preventing other applications from handling the packets they are intended to process. Besides, it was mentioned that although certain applications might not be designed to execute deviant behavior, in certain cases such applications innocently, when integrated into the system, release or enable vulnerabilities. From the perspective that the SDN is prone to both mentioned situations, it is necessary to implement a security mechanism whose principal function should be to audit the security of network applications' security in two phases. In one phase, the application might be audited in isolation, as an individual component that will be later integrated to the SDN system, at this stage, a vulnerability assessment and code inspection (coverage tests, unitary tests, anomalous code snippets, etc.) should be executed. In the other phase, a functional assessment might detect if the integration and interaction of the application with some other SDN instances induces any undesired behavior, for instance, if the application conflicts with network policies when instructing the installation of flow rules. Suggested measures should be complemented with a trust authentication and authorization mechanism, therefore, restricting the execution and access to the network to ill-intentioned third-party applications or shellcodes injected using another vulnerable services.

All the solutions presented in this survey can be used as a basis for programming and automating the security of SDN networks. All these solutions for securing exchanges in a traditional SDN network. But they can be extended to the security of cyber-physical systems and IoT objects in the future.

## 8 Conclusion

Since the beginning of computer network development, several new protocols, specifications, methodologies, techniques, approaches, and technologies have emerged for the network to solve society's growing demands. However, the increased complexity of these solutions and the accelerated growth of new solutions made networks ossified, increasing the challenge of proposing innovative solutions due to the need to keep compatibility with what has already been implemented in the past, which impacts the resistance to disruption. As an alternative to the status quo, the SDN paradigm was proposed, enabling the development of solutions more decoupled through the separation of the network plans (management, control, and data). The break of the vertical integration and applying the separation of responsibilities principle make some new functionalities available, such as the network logical centralization and the network programmability. This allows the development of new solutions and functionalities for the network to emerge without

compromising its operation. However, like every disruptive innovation, the SDN paradigm has entered the Gartner curve's adoption phase, attracting the attention of academia and industry. As SDN is a paradigm that has not yet reached maturity, society's focus was for a long time directed towards developing the paradigm's main features to be functionally applied in real environments. This impacted that requirements such as performance were prioritized to the detriment of others, such as security, for instance. As the paradigm evolved, security started to be a critical point. Several vulnerabilities were discovered and exploited to provide several types of attacks, thus requiring new defense mechanisms.

This paper categorized the main vulnerabilities, attack types, and mitigations mechanisms involving SDN described in the literature.

## References


1. Abdullaziz OI, Wang L (2019) Mitigating DoS Attacks against SDN controller using information hiding. In: 2019 IEEE Wireless Communications and Networking Conference (WCNC). pp 1–6.https://doi.org/10.1109/WCNC.2019.8885764
2. Agborubere B, Sanchez-Velazquez E (2017) OpenFlow communications and TLS security in software-defined networks. In: 2017 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp 560–566. https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2017.88
3. Ahmad I, Namal S, Ylianttila M, Gurtov A (2015) Security in software defined networks: a survey. IEEE Commun Surv Tutor 17(4):2317–2346. https://doi.org/10.1109/COMST.2015.2474118
4. Ahmed ME, Kim H (2017) DDoS attack mitigation in internet of things using software defined networking. In: 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService). pp 271–276.https://doi.org/10.1109/BigDataService.2017.41
5. Aizuddin AA, Atan M, Norulazmi M, Noor MM, Akimi S and Abidin Z (2017) DNS Amplification attack detection and mitigation via sflow with security-centric SDN. In: Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication. https://doi.org/10.1145/3022227.3022230
6. Al-Haj S, Tolone WJ (2017) FlowTable pipeline misconfigurations in Software Defined Networks. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp 247–252.https://doi.org/10.1109/INFCOMW.2017.8116384
7. Al-Shaer E, Al-Haj S (2010) FlowChecker: configuration analysis and verification of federated openflow infrastructures. In: Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration, pp 37–44. https://doi.org/10.1145/1866898.1866905
8. Alasadi E, Al-Raweshidy HS (2018) SSED: servers under software-defined network architectures to eliminate discovery messages. IEEE/ACM Trans Netw 26(1):104–117. https://doi.org/10.1109/TNET.2017.2763131
9. Alcorn JA, Chow CE (2014) A framework for large-scale modeling and simulation of attacks on an OpenFlow network. In: 2014 23rd International Conference on Computer Communication and Networks (ICCCN). pp 1–6. https://doi.org/10.1109/ICCCN.2014.6911848
10. Allouzi M, Khan J (2018) SafeFlow: authentication protocol for software defined networks. In: 2018 IEEE 12th International Conference on Semantic Computing (ICSC). pp 374–376. https://doi.org/10.1109/ICSC.2018.00076
11. Alparslan O, Gunes O, Hanay YS, Arakawa S, Murata M (2017) Improving resiliency against DDoS attacks by SDN and multipath orchestration of VNF services. In: 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). pp 1–3.https://doi.org/10.1109/LANMAN.2017.7972158
12. Ambrosin M, Conti M, Gaspari FD, Poovendran R (2017) LineSwitch: tackling control plane saturation attacks in software-defined networking. IEEE/ACM Trans Netw 25(2):1206–1219. https://doi.org/10.1109/TNET.2016.2626287
13. Aseeri A, Netjinda N, Hewett R (2017) Alleviating eavesdropping attacks in software-defined networking data plane. In: Proceedings of the 12th Annual Conference on Cyber and Information Security Research. https://doi.org/10.1145/3064814.3064832
14. De Assis MVO, Hamamoto AH, Abrão T, Proença ML (2017) A game theoretical based system using holt-winters and genetic algorithm with fuzzy logic for DoS/DDoS mitigation on SDN networks. IEEE Access 5:9485–9496. https://doi.org/10.1109/ACCESS.2017.2702341
15. Bailey J, Budgen D, Turner M, Kitchenham B, Brereton P, Linkman S (2007) Evidence relating to object-oriented software design: a survey. In: First international symposium on empirical software engineering and measurement (ESEM 2007). pp 482–484. https://doi.org/10.1109/ESEM.2007.58
16. Banse C, Schuette J (2017) A taxonomy-based approach for security in software-defined networking. In: 2017 IEEE International Conference on Communications (ICC). pp 1–6. https://doi.org/10.1109/ICC.2017.7997245
17. Bauer R, Dittebrandt A, Zitterbart M (2019) GCMI: a generic approach for SDN control message interception. In: 2019 IEEE Conference on Network Softwarization (NetSoft). pp 360–368. https://doi.org/10.1109/NETSOFT.2019.8806661
18. Bera S, Misra S, Vasilakos AV (2017) Software-defined networking for internet of things: a survey. IEEE Internet Things J 4(6):1994–2008. https://doi.org/10.1109/JIOT.2017.2746186
19. Braga R, Mota E, Passito A (2010) Lightweight DDoS flooding attack detection using NOX/OpenFlow. IEEE Local Comput Netw Conf. https://doi.org/10.1109/LCN.2010.5735752
20. Brooks M, Yang B (2015) A man-in-the-middle attack against opendaylight SDN controller. In: Proceedings of the 4th Annual ACM Conference on Research in Information Technology. pp 45–49. https://doi.org/10.1145/2808062.2808073
21. Schlesinger C, Story A, Gutz S, Foster N and W D (2012). Splendid isolation: Language-based security for softwaredefined networks. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks. ACM pp 79–84
22. Carvalho RN, Bordim JL, Alchieri EAP (2019) Entropy-based DoS attack identification in SDN. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp 627–634.https://doi.org/10.1109/IPDPSW.2019.00108
23. Chang S, Park Y, Babu BBA (2019) Fast IP hopping randomization to secure hop-by-hop access in SDN. IEEE Trans Netw Serv Manage 16(1):308–320. https://doi.org/10.1109/TNSM.2018.2889842
24. Chen M-H, Ciou J-Y, Chung I-H, Chou C-F (2018) FlexProtect: a SDN-based DDoS attack protection architecture for multi-tenant data centers. Proc Int Conf High Perform Comput Asia-Pacific Region. https://doi.org/10.1145/3149457.3149476

25. Chica JCC, Imbachi JC, Vega JFB (2020) Security in SDN: a comprehensive survey. J Netw Comput Appl 159:102595
26. Chi P-W, Kuo C-T, Guo J-W, Lei C-L (2015) How to detect a compromised SDN switch. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). pp 1–6. https://doi.org/10.1109/NETSOFT.2015.7116184
27. Chin T, Mountrouidou X, Li X, Xiong K (2015). Selective packet inspection to detect DoS flooding using software defined networking (SDN). In: 2015 IEEE 35th International Conference on Distributed Computing Systems Workshops. pp 95–99. https://doi.org/10.1109/ICDCSW.2015.27
28. Chowdhary A, Alshamrani A, Huang D, Liang H (2018). MTD analysis and evaluation framework in software defined network (MASON). In: Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization. pp 43–48. https://doi.org/10.1145/3180465.3180473
29. Chowdhary A, Huang D, Ahn G-J, Kang M, Kim A, Velazquez A (2019) SDNSOC: object oriented SDN framework. In: Proceedings of the ACM International Workshop on Security in Software Defined Networks and Network Function Virtualization. pp 7–12. https://doi.org/10.1145/3309194.3309196
30. Chung C, Member S, Khatkar P, Xing T (2013) NICE : network intrusion detection and countermeasure. IEEE Trans Depend Secure Comput 10(4):1–14. http://dblp.uni-trier.de/db/journals/tdsc/tdsc10.html#ChungKXLH13
31. Conti M, Gaspari FD, Mancini LV (2020) A novel stealthy attack to gather SDN configuration-information. IEEE Trans Emerg Top Comput 8(2):328–340. https://doi.org/10.1109/TETC.2018.2806977
32. Controller T (2013) Trema controller. Full-Stack OpenFlow Framework in Ruby and C. Retrieved September 12, 2020, from https://trema.github.io/trema/
33. Cui H, Chen Z, Yu L, Xie K, Xia Z (2017) Authentication mechanism for network applications in SDN environments. In: 2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC). pp 1–5. https://doi.org/10.1109/WPMC.2017.8301788
34. Cui Y, Yan L, Li S, Xing H, Pan W, Zhu J, Zheng X (2016) SD-Anti-DDoS: fast and efficient DDoS defense in software-defined networks. J Netw Comput Appl 68:65–79. https://doi.org/10.1016/j.jnca.2016.04.005
35. Cziva R, Jouët S, Stapleton D, Tso FP, Pezaros DP (2016) SDN-based virtual machine management for cloud data centers. IEEE Trans Netw Serv Manage 13(2):212–225
36. D'Orsaneo J, Tummala M, McEachen J, Martin B (2018) Analysis of traffic signals on an SDN for detection and classification of a man-in-the-middle attack. In: 2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS). pp 1–9. https://doi.org/10.1109/ICSPCS.2018.8631762
37. Dargahi T, Caponi A, Ambrosin M, Bianchi G, Conti M (2017) A Survey on the Security of Stateful SDN Data Planes. IEEE Commun Surv Tutor. https://doi.org/10.1109/COMST.2017.2689819
38. da Silva AS, Smith P, Mauthe A, Schaeffer-Filho A (2015) Resilience support in software-defined networking: a survey. Comput Netw 92:189–207
39. Dridi L, Zhani MF (2016) SDN-Guard: DoS attacks mitigation in SDN networks. In: 2016 5th IEEE International Conference on Cloud Networking (Cloudnet). pp 212–217. https://doi.org/10.1109/CloudNet.2016.9
40. Erickson D (2013) The beacon openflow controller. In: Proceedings of the second ACM SIGCOMM workshop on hot topics in software defined networking, August 2013, pp 13–18
41. Feghali A, Kilany R, Chamoun M (2015) SDN security problems and solutions analysis. In: 2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS). pp 1–5. https://doi.org/10.1109/NOTERE.2015.7293514
42. Fernandez MP (2013) Comparing OpenFlow controller paradigms scalability: reactive and proactive. In: 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). pp 1009–1016. https://doi.org/10.1109/AINA.2013.113
43. Fichera S, Galluccio L, Grancagnolo SC, Morabito G, Palazzo S (2015) OPERETTA: an openflow-based remedy to mitigate TCP SYNFLOOD attacks against web servers. Comput Netw 92:89–100. https://doi.org/10.1016/j.comnet.2015.08.038
44. Fielding RT, Taylor RN (2000) Architectural styles and the design of network-based software architectures, vol 7. University of California, Irvine
45. Floodlight (2013) Floodlight OpenFlow controller. Available from http://www.projectfloodlight.org/floodlight
46. Foerster K, Ludwig A, Marcinkowski J, Schmid S (2018) Loop-free route updates for software-defined networks. IEEE/ACM Trans Netw 26(1):328–341. https://doi.org/10.1109/TNET.2017.2778426
47. François J, Dolberg L, Festor O, Engel T (2014) Network security through software defined networking: a survey. Proc Conf Principles Syst Appl IP Telecommun. https://doi.org/10.1145/2670386.2670390
48. Freire L, Neves M, Leal L, Levchenko K, Schaeffer-Filho A, Barcellos M (2018) Uncovering bugs in P4 programs with assertion-based verification. Proc Sympos SDN Res. https://doi.org/10.1145/3185467.3185499
49. Gao S, Li Z, Xiao B, Wei G (2018) Security threats in the data plane of software-defined networks. IEEE Network 32(4):108–113. https://doi.org/10.1109/MNET.2018.1700283
50. Gao S, Li Z, Yao Y, Xiao B, Guo S, Yang Y (2018) Software-defined firewall: enabling malware traffic detection and programmable security control. In: Proceedings of the 2018 on Asia Conference on Computer and Communications Security. pp 413–424. https://doi.org/10.1145/3196494.3196519
51. Giotis K, Argyropoulos C, Androulidakis G, Kalogeras D, Maglaris V (2014) Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Comput Netw 62:122–136
52. Goksel N, Demirci M (2019) DoS attack detection using packet statistics in SDN. In: 2019 International Symposium on Networks, Computers and Communications (ISNCC). pp 1–6https://doi.org/10.1109/ISNCC.2019.8909114
53. Gude N, Koponen T, Pettit J, Pfaff B, Casado M, McKeown N, Shenker S (2008) NOX: towards an operating system for networks. Comput Commun Rev. https://doi.org/10.1145/1384609.1384625
54. Hall RS, Cervantes H (2004) An OSGi implementation and experience report. In: First IEEE Consumer Communications and Networking Conference, 2004. CCNC 2004. pp 394–399. https://doi.org/10.1109/CCNC.2004.1286894
55. Hamdan M, Hassan E, Abdelaziz A, Elhigazi A, Mohammed B, Khan S, Vasilakos AV, Marsono MN (2021) A comprehensive survey of load balancing techniques in software-defined network. J Netw Comput Appl 174:102856. https://doi.org/10.1016/j.jnca.2020.102856
56. de la Hoz E, Cochrane G, Moreira-Lemus JM, Paez-Reyes R, Marsa-Maestre I, Alarcos B (2014) Detecting and defeating advanced man-in-the-middle attacks against TLS. In: 2014 6th International Conference On Cyber Conflict (CyCon 2014). pp 209–221. https://doi.org/10.1109/CYCON.2014.6916404
57. Hu T, Yi P, Hu Y, Lan J, Zhang Z, Li Z (2020) SAIDE: Efficient application interference detection and elimination in SDN. Comput Netw 183:107619. https://doi.org/10.1016/j.comnet.2020.107619

58. Hu Y, Su W, Wu L, Huang Y, Kuo S (2013) Design of event-based intrusion detection system on openflow network. In: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp 1–2. https://doi.org/10.1109/DSN.2013.6575335
59. Ishii S, Kawai E, Takata T, Kanaumi Y, Saito S, Kobayashi K, Shimojo S (2012) Extending the RISE controller for the interconnection of RISE and OS3E/NDDI. In: 2012 18th IEEE International Conference on Networks (ICON). pp 243–248. https://doi.org/10.1109/ICON.2012.6506564
60. Isong B, Molose RRS, Abu-Mahfouz AM, Dladlu N (2020) Comprehensive review of SDN controller placement strategies. IEEE Access 8:170070–170092. https://doi.org/10.1109/ACCESS.2020.3023974
61. Jafarian JH, Al-Shaer E, Duan Q (2013) Formal approach for route agility against persistent attackers. In: Crampton J, Jajodia S, Mayes K (eds) In european symposium on research in computer security. Springer, Berlin, pp 237–254
62. Jäger B, Röpke C, Adam I, Holz T (2015) Multi-layer access control for SDN-based Telco clouds. In: Buchegger S, Dam M (eds) In Nordic conference on secure IT systems. Springer International Publishing, pp 197–204
63. Jain R (2012) OpenADN: mobile apps on global clouds using software defined networking. In: Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services. pp 1–2.https://doi.org/10.1145/2307849.2307851
64. Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, Venkata S, Wanderer J, Zhou J, Zhu M, Zolla J, Hölzle U, Stuart S, Vahdat A (2013) B4: Experience with a globally-deployed software defined wan. SIGCOMM Comput Commun Rev 43(4):3–14. https://doi.org/10.1145/2534169.2486019
65. Jeong K, Kim J, Kim Y (2012) QoS-aware Network Operating System for software defined networking with Generalized OpenFlows. In: 2012 IEEE Network Operations and Management Symposium. pp 1167–1174.https://doi.org/10.1109/NOMS.2012.6212044
66. Kempf J, Bellagamba E, Kern A, Jocha D, Takacs A, Sköldström P (2012) Scalable fault management for OpenFlow. In: 2012 IEEE International Conference on Communications (ICC). pp 6606–6610.https://doi.org/10.1109/ICC.2012.6364688
67. Khurshid A, Zou X, Zhou W, Caesar M, Godfrey PB (2013) VeriFlow: verifying network-wide invariants in real time. In: 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13). pp 15–27. https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/khurshid
68. Kim E, Kim K, Lee S, Jeong JP, Kim H (2018) A Framework for managing user-defined security policies to support network security functions. In: Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication. https://doi.org/10.1145/3164541.3164569
69. Klaedtke F, Karame GO, Bifulco R, Cui H (2015) Towards an access control scheme for accessing flows in SDN. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft). pp 1–6. https://doi.org/10.1109/NETSOFT.2015.7116185
70. Klaedtke F, Karame GO, Bifulco R, Cui H (2014) Access control for SDN controllers. Proc Third Workshop Hot Top Softw Defined Netw. https://doi.org/10.1145/2620728.2620773
71. Koponen T, Casado M, Gude N, Stribling J, Poutievski L, Zhu M, Ramanathan R, Iwata Y, Inoue H, Hama T, Shenker S (2010) Onix: a distributed control platform for large-scale production networks. In OSDI. In OSDI, 10
72. Kotani D, Okabe Y (2016) A packet-in message filtering mechanism for protection of control plane in OpenFlow switches. IEICE Trans Inf Syst 99(3):695–707
73. Kreutz D, Ramos FMV, Verissimo P (2013) Towards secure and dependable software-defined networks. In: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. pp 55–60.https://doi.org/10.1145/2491185.2491199
74. Kuerban M, Tian Y, Yang Q, Jia Y, Huebert B, Poss D (2016) FlowSec: DOS attack mitigation strategy on SDN controller. In: 2016 IEEE International Conference on Networking, Architecture and Storage (NAS). pp 1–2. https://doi.org/10.1109/NAS.2016.7549402
75. Lévai T, Pelle I, Németh F, Gulyás A (2015) EPOXIDE: a modular prototype for SDN troubleshooting. SIGCOMM Comput Commun Rev 45(4):359–360. https://doi.org/10.1145/2829988.2790027
76. Li H, Li P, Guo S, Yu S (2014) Byzantine-resilient secure software-defined networks with multiple controllers. In: 2014 IEEE International Conference on Communications (ICC). pp 695–700.https://doi.org/10.1109/ICC.2014.6883400
77. Li Q, Zou X, Huang Q, Zheng J, Lee PPC (2019) Dynamic packet forwarding verification in SDN. IEEE Trans Dependable Secure Comput 16(6):915–929. https://doi.org/10.1109/TDSC.2018.2810880
78. Liu B, Bi J, Zhou Y (2016) Source address validation in software defined networks. In: Proceedings of the 2016 ACM SIGCOMM conference. pp 595–596. https://doi.org/10.1145/2934872.2960425
79. Maestro. (2009). Maestro. Maestro homepage: http://zhengcai.github.io/maestro-platform/
80. Masoud MZ, Jaradat Y, Jannoud I (2015) On preventing ARP poisoning attack utilizing Software Defined Network (SDN) paradigm. In: 2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT). pp. 1–5.https://doi.org/10.1109/AEECT.2015.7360549
81. Matsumoto S, Hitz S, Perrig A (2014) Fleet: defending SDNs from malicious administrators. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. pp 103–108.https://doi.org/10.1145/2620728.2620750
82. Mekky H, Hao F, Mukherjee S, Zhang Z-L, Lakshman TV (2014) Application-aware data plane processing in SDN. In: Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. pp 13–18. https://doi.org/10.1145/2620728.2620735
83. Midha S, Triptahi K (2019) Extended TLS security and Defensive Algorithm in OpenFlow SDN. In: 2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence). pp 141–146. https://doi.org/10.1109/CONFLUENCE.2019.8776607
84. Mihai-Gabriel I, Victor-Valeriu P (2014) Achieving DDoS resiliency in a software defined network by intelligent risk assessment based on neural networks and danger theory. In: 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI). pp 319–324. https://doi.org/10.1109/CINTI.2014.7028696
85. Moazzeni S, Khayyambashi MR, Movahhedinia N, Callegati F (2018) On reliability improvement of Software-Defined Networks. Comput Netw 133:195–211. https://doi.org/10.1016/j.comnet.2018.01.023
86. Mohammadi R, Javidan R, Conti M (2017) SLICOTS: an SDN-based lightweight countermeasure for TCP SYN flooding attacks. IEEE Trans Netw Serv Manag 14(2):487–497
87. Mohan PM, Truong-Huu T, Gurusamy M (2018) Towards resilient in-band control path routing with malicious switch detection in SDN. In: 2018 10th International Conference on Communication Systems & Networks (COMSNETS). pp 9–16. https://doi.org/10.1109/COMSNETS.2018.8328174
88. Monsanto C, Foster N, Harrison R, Walker D (2012) A compiler and run-time system for network programming languages. SIGPLAN Not 47(1):217–230. https://doi.org/10.1145/2103621.2103685

89. Monsanto C, Foster N, Harrison R, Walker D (2012) A complier and run-time system for network programming languages. Sigplan Not. https://doi.org/10.1145/2103621.2103685
90. Morzhov SV, Nikitinskiy MA (2018) Development and research of the PreFirewall network application for floodlight SDN controller. In: 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT). pp 1–4.https://doi.org/10.1109/MWENT.2018.8337255
91. Nagai R, Kurihara W, Higuchi S, Hirotsu T (2018) Design and implementation of an OpenFlow-based TCP SYN flood mitigation. In: 2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud). pp 37–42. https://doi.org/10.1109/MobileCloud.2018.00014
92. Namal S, Ahmad I, Gurtov A, Ylianttila M (2013) Enabling secure mobility with OpenFlow. In: 2013 IEEE SDN for Future Networks and Services (SDN4FNS). pp 1–5. https://doi.org/10.1109/SDN4FNS.2013.6702540
93. Nguyen T, Yoo M (2016) Attacks on host tracker in SDN controller: Investigation and prevention. In: 2016 International Conference on Information and Communication Technology Convergence (ICTC). pp 610–612.https://doi.org/10.1109/ICTC.2016.7763545
94. Nife F, Kotulski Z (2018). In: Gaj P, Sawicki M, Suchacka G, Kwiecień A (eds) New SDN-oriented authentication and access control mechanism BT-computer networks. Springer International Publishing, Berlin, pp 74–88
95. Oktian YE, Lee S, Lee H, Lam J (2015) Secure your Northbound SDN API. In: 2015 Seventh International Conference on Ubiquitous and Future Networks. pp 919–920.https://doi.org/10.1109/ICUFN.2015.7182679
96. Oktian YE, Lee SG, Lee HJ, Lam JH (2017) Distributed SDN controller system: a survey on design choice. Comput Netw 121:100–111. https://doi.org/10.1016/j.comnet.2017.04.038
97. OpenDaylight (2014) OpenDaylight: a linux foundation collaborative project. http://www.opendaylight.org/
98. Porras P, Cheung S, Fong M, Skinner K and Y V (2015) Securing the software-defined network control layer
99. Padekar H, Park Y, Hu H, Chang S-Y (2016) Enabling dynamic access control for controller applications in software-defined networks. In: Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies. pp 51–61. https://doi.org/10.1145/2914642.2914647
100. Pan H, Li Z, Zhang P, Salamatian K, Xie G (2020) Misconfiguration checking for SDN: data structure, theory and algorithms. In: 2020 IEEE 28th International Conference on Network Protocols (ICNP). pp 1–11. https://doi.org/10.1109/ICNP49622.2020.9259353
101. Park T, Kim Y, Yegneswaran V, Porras P, Xu Z, Park K, Shin S (2019) DPX: data-plane extensions for SDN security service instantiation. In: Perdisci R, Maurice C, Giacinto G, Almgren M (eds) International conference on detection of intrusions and malware, and vulnerability assessment. Springer International Publishing, pp 415–437
102. Petersen K, Feldt R, Mujtaba S, Mattsson M (2008) Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), vol. 12. pp 1–10
103. Phan TV, Park M (2019) Efficient distributed denial-of-service attack defense in SDN-based cloud. IEEE Access 7:18701–18714. https://doi.org/10.1109/ACCESS.2019.2896783
104. Phemius K, Bouet M, Leguay J (2014) DISCO: Distributed multi-domain SDN controllers. In: 2014 IEEE Network Operations and Management Symposium (NOMS). pp 1–4.https://doi.org/10.1109/NOMS.2014.6838330
105. Porras P, Shin S, Yegneswaran V, Fong M, Tyson M, Gu G (2012) A security enforcement kernel for OpenFlow networks. https://doi.org/10.1145/2342441.2342466
106. Porras P, Cheung S, Fong M, Skinner K, Yegneswaran V (2015) Securing the software defined network control layer. In: Proceedings of the 2015 Network and Distributed System Security Symposium (NDSS), February, 8–11. https://doi.org/10.14722/ndss.2015.23222
107. Prete LR, Shinoda AA, Schweitzer CM, Oliveira RLS (2014) Simulation in an SDN network scenario using the POX Controller. In: 2014 IEEE Colombian Conference on Communications and Computing (COLCOM). pp 1–6. https://doi.org/10.1109/ColComCon.2014.6860403
108. Qasmaoui Y, Haqiq A (2020) Enhanced solid-flow: an enhanced flow rules security mechanism for SDN. IAENG Int J Comput Sci 47(3):522–532
109. Qasmaoui Y, Haqiq A (2017) Solid-flow: a flow rules security mechanism for SDN. In: 2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech). pp 1–7. https://doi.org/10.1109/CloudTech.2017.8284734
110. Qi C, Wu J, Hu H, Cheng G, Liu W, Ai J, Yang C (2016) An intensive security architecture with multi-controller for SDN. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). pp 401–402.https://doi.org/10.1109/INFCOMW.2016.7562109
111. Ranjbar A, Komu M, Salmela P, Aura T (2016) An SDN-based approach to enhance the end-to-end security: SSL/TLS case study. In: NOMS 2016—2016 IEEE/IFIP Network Operations and Management Symposium. pp 281–288. https://doi.org/10.1109/NOMS.2016.7502823
112. Ryu (2017) Ryu SDN framework. Ryu Homepage: http://osrg.github.io/ryu/.
113. Saâdaoui A, Souayeh NBYB, Bouhoula A (2019) Automated and optimized formal approach to verify SDN access-control misconfigurations. In: Gao H, Yin Y, Yang X, Miao H (eds) International conference on testbeds and research infrastructure. Springer International Publishing, pp 96–112
114. Sahay R, Blanc G, Zhang Z, Debar H (2017) ArOMA: an SDN based autonomic DDoS mitigation framework. Comput Secur 70:482–499
115. Sasaki T, Pappas C, Lee T, Hoefler T, Perrig A (2016) SDNsec: forwarding accountability for the SDN data plane. In: 2016 25th International Conference on Computer Communication and Networks (ICCCN). pp 1–10. https://doi.org/10.1109/ICCCN.2016.7568569
116. Sasaki T, Perrig A, Asoni DE (2016) Control-plane isolation and recovery for a secure SDN architecture. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft). pp 459–464.https://doi.org/10.1109/NETSOFT.2016.7502485
117. Schehlmann L, Abt S, Baier H (2014) Blessing or curse? Revisiting security aspects of Software-Defined Networking. In: 10th International Conference on Network and Service Management (CNSM) and Workshop. pp 382–387. https://doi.org/10.1109/CNSM.2014.7014199
118. Scott-Hayward S, Kane C, Sezer S (2014) OperationCheckpoint: SDN application control. In: 2014 IEEE 22nd International Conference on Network Protocols. pp 618–623. https://doi.org/10.1109/ICNP.2014.98
119. Scott-Hayward S, O'Callaghan G, Sezer S (2013) SDN security: a survey. Future networks and services (SDN4FNS), 2013 IEEE SDN for. pp 1–7
120. Sebbar A, Boulmalf M, Kettani MDE-CEl, Baddi Y (2018). Detection MITM attack in multi-SDN controller. In: 2018 IEEE 5th International Congress on Information Science and Technology (CiSt). pp 583–587. https://doi.org/10.1109/CIST.2018.8596479

121. Sezer S, Scott-Hayward S, Chouhan PK, Fraser B, Lake D, Finnegan J, Viljoen N, Miller M, Rao N (2013) Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Commun Mag 51(7):36–43. https://doi.org/10.1109/MCOM.2013.6553676
122. Shaghaghi A, Kaafar MA, Buyya R, Jha S (2018) Software-Defined Network (SDN) Data plane security: issues, solutions and future directions. ArXiv Preprint http://arxiv.org/abs/1804.00262.
123. Shin J, Kim T, Lee B, Yang S (2017) IRIS-HiSA: highly scalable and available carrier-grade SDN controller cluster. Mob Netw Appl. https://doi.org/10.1007/s11036-017-0853-6
124. Shin S, Porras P, Yegneswaran V, Gu G (2013) FRESCO: Modular composable security services for software-defined networks. Netw Distrib Syst Secur Sympos 1(1):1–16
125. Shin S, Yegneswaran V, Porras P, Gu G (2013) Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In: Proceedings of the 2013 ACM SIGSAC conference on computer and communications security. pp 413–424
126. Shu Z, Wan J, Li D, Lin J, Vasilakos AV, Imran M (2016) Security in software-defined networking: threats and countermeasures. Mob Netw Appl 21(5):764–776. https://doi.org/10.1007/s11036-016-0676-x
127. Shuangyu H, Jianwei L, Jian M, Jie C (2014) Hierarchical solution for access control and authentication in software defined networks. In: Au MH, Carminati B, Kuo C-CJ (eds) International conference on network and system security. Springer International Publishing, pp 70–81
128. Singh J, Behal S (2020) Detection and mitigation of DDoS attacks in SDN: a comprehensive review, research challenges and future directions. Comput Sci Rev 37:100279. https://doi.org/10.1016/j.cosrev.2020.100279
129. SNAC (2012) SNAC: simple network access control. https://github.com/
130. Son J, Buyya R (2018) A taxonomy of software-defined networking (SDN)-enabled cloud computing. ACM Comput Surv (CSUR) 51(3):59
131. Son J, Dastjerdi AV, Calheiros RN, Buyya R (2017) SLA-aware and energy-efficient dynamic overbooking in SDN-based cloud data centers. IEEE Trans Sustain Comput 2(2):76–89
132. Son S, Shin S, Yegneswaran V, Porras P, Gu G (2013) Model checking invariant security properties in OpenFlow. IEEE Int Conf Commun. https://doi.org/10.1109/ICC.2013.6654813
133. Specification OS (2013) Open networking foundation. Version ONF TS-015 1(3):1–164
134. Suh J, Choi H, Yoon W, You T, Kwon TT, Choi Y (2010) Implementation of content-oriented networking architecture (CONA): a focus on DDoS countermeasure. In: 1st European NetFPGA Developers Workshop. pp 1–5. https://mmlab.snu.ac.kr/publications/docs/2010_EU_netfpga_workshop_jhsuh.pdf
135. Tootoonchian A, Ganjali Y (2010) HyperFlow: a distributed control plane for OpenFlow
136. Tootoonchian A, Gorbunov S, Ganjali Y, Casado M, Sherwood R (2012) On controller performance in software-defined networks. In: 2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12). https://www.usenix.org/conference/hot-ice12/workshop-program/presentation/tootoonchian
137. Voellmy A, Hudak P (2011). In: Rocha R, Launchbury J (eds) Nettle: taking the sting out of programming network routers BT-practical aspects of declarative languages. Springer, Berlin, pp 235–249
138. Voellmy A, Kim H, Feamster N (2012). Procera: a language for high-level reactive network control. In: HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks. https://doi.org/10.1145/2342441.2342451
139. Voellmy A, Wang J (2012) Scalable software defined network controllers. In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. pp 289–290. https://doi.org/10.1145/2342356.2342414
140. Wang H (2014) Authentic and confidential policy distribution in software defined wireless network. In: 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). pp 1167–1171.https://doi.org/10.1109/IWCMC.2014.6906520
141. Wang H, Xu L, Gu G (2015) FloodGuard: a DoS attack prevention extension in software-defined networks. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp 239–250. https://doi.org/10.1109/DSN.2015.27
142. Wang M, Liu J, Chen J, Liu X, Mao J (2016) PERM-GUARD: authenticating the validity of flow rules in software defined networking. In: Proceedings—2nd IEEE International Conference on Cyber Security and Cloud Computing, CSCloud 2015—IEEE International Symposium of Smart Cloud, IEEE SSC 2015, 37. pp 127–132. https://doi.org/10.1109/CSCloud.2015.89
143. Wei L, Fung C (2015) FlowRanger: a request prioritizing algorithm for controller DoS attacks in Software Defined Networks. In: 2015 IEEE International Conference on Communications (ICC). pp 5254–5259.https://doi.org/10.1109/ICC.2015.7249158
144. Wen X, Chen Y, Hu C, Shi C, Wang Y (2013) Towards a secure controller platform for openflow applications.https://doi.org/10.1145/2491185.2491212
145. Wu B, Li H, Wu Q, Jiang Z, Liu J (2020) TMPTCP: a lightweight trust extension for multipath-TCP. In: 2020 International Conference on Networking and Network Applications (NaNA). pp 342–347.https://doi.org/10.1109/NaNA51271.2020.00065
146. Wu G, Wang J, Obaidat MS, Yao L, Hsiao K-F (2019) Dynamic switch migration with noncooperative game towards control plane scalability in SDN. Int J Commun Syst 32(7):e3927. https://doi.org/10.1002/dac.3927
147. Xie R, Xu M, Cao J, Li Q (2019) SoftGuard: defend against the low-rate TCP attack in SDN. In: ICC 2019—2019 IEEE International Conference on Communications (ICC). pp 1–6. https://doi.org/10.1109/ICC.2019.8761806
148. Yan Z, Zhang P, Vasilakos AV (2016) A security and trust framework for virtualized networks and software-defined networking. Secur Commun Netw 9(16):3059–3069. https://doi.org/10.1002/sec.1243
149. Yang M, Li Y, Jin D, Zeng L, Wu X, Vasilakos AV (2015) Software-defined and virtualized future mobile and wireless networks: a survey. Mob Netw Appl 20(1):4–18. https://doi.org/10.1007/s11036-014-0533-8
150. Yao G, Bi J, Xiao P (2011) Source address validation solution with OpenFlow/NOX architecture. In: 2011 19th IEEE International Conference on Network Protocols. pp 7–12. https://doi.org/10.1109/ICNP.2011.6089085
151. Ying Q, Wanqssing Y, Kai Q (2016) OpenFlow flow table overflow attacks and countermeasures. In: 2016 European Conference on Networks and Communications (EuCNC). pp 205–209.https://doi.org/10.1109/EuCNC.2016.7561033
152. Yue M, Wang H, Liu L, Wu Z (2020) Detecting DoS attacks based on multi-features in SDN. IEEE Access 8:104688–104700. https://doi.org/10.1109/ACCESS.2020.2999668
153. Zhang C, Hu G, Chen G, Sangaiah AK, Zhang P, Yan X, Jiang W (2018) Towards a SDN-based integrated architecture for mitigating IP spoofing attack. IEEE Access 6:22764–22777. https://doi.org/10.1109/ACCESS.2017.2785236
154. Zhang H, Cai Z, Liu Q, Xiao Q, Li Y, Cheang CF (2018) A survey on security-aware measurement in SDN. Secur Commun Netw
155. Zhang K, Qiu X (2018) CMD: a convincing mechanism for MITM detection in SDN. In: 2018 IEEE International Conference on

Consumer Electronics (ICCE). pp 1–6.https://doi.org/10.1109/ICCE.2018.8326334
156. Zhang L, Guo Y, Yuwen H, Wang Y (2016) A port hopping based DoS mitigation scheme in SDN network. In: 2016 12th International Conference on Computational Intelligence and Security (CIS). pp 314–317. https://doi.org/10.1109/CIS.2016.0077
157. Zhang L, Wang Z, Gu K, Miao F, Guo Y (2016) Transparent synchronization based port mutation scheme in SDN network. In: 2016 5th International Conference on Computer Science and Network Technology (ICCSNT). pp 581–585. https://doi.org/10.1109/ICCSNT.2016.8070225
158. Zhang L, Wei Q, Gu K, Yuwen H (2016) Path hopping based SDN network defense technology. In: 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD). pp 2058–2063. https://doi.org/10.1109/FSKD.2016.7603498
159. Zhang P, Wang H, Hu C, Lin C (2016) On denial of service attacks in software defined networks. IEEE Network 30(6):28–33. https://doi.org/10.1109/MNET.2016.1600109NM
160. Zhang Y, Beheshti N, Tatipamula M (2011) On resilience of split-architecture networks. In: 2011 IEEE Global Telecommunications Conference - GLOBECOM 2011. pp 1–6. https://doi.org/10.1109/GLOCOM.2011.6134496
161. Zheng J, Li Q, Gu G, Cao J, Yau DKY, Wu J (2018) Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis. IEEE Trans Inf Forensics Secur 13(7):1838–1853. https://doi.org/10.1109/TIFS.2018.2805600
162. Zhou H, Wu C, Yang C, Wang P, Yang Q, Lu Z, Cheng Q (2018) SDN-RDCD: a real-time and reliable method for detecting compromised SDN devices. IEEE/ACM Trans Netw 26(5):2048–2061. https://doi.org/10.1109/TNET.2018.2859483
163. Zhu L, Tang X, Shen M, Du X, Guizani M (2018) Privacy-Preserving DDoS attack detection using cross-domain traffic in software defined networks. IEEE J Sel Areas Commun 36(3):628–643. https://doi.org/10.1109/JSAC.2018.2815442
164. Zou D, Lu Y, Yuan B, Chen H, Jin H (2018) A fine-grained multi-tenant permission management framework for SDN and NFV. IEEE Access 6:25562–25572. https://doi.org/10.1109/ACCESS.2018.2828132

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.