Hindawi
Security and Communication Networks
Volume 2019, Article ID 6874592, 16 pages
https://doi.org/10.1155/2019/6874592

Research Article

FORTRESS: An Efficient and Distributed Firewall for Stateful Data Plane SDN

Maurantonio Caprolu, Simone Raponi, and Roberto Di Pietro

Hamad Bin Khalifa University (HBKU), College of Science and Engineering (CSE), Division of Information and Computing Technology (ICT), Doha, Qatar

Correspondence should be addressed to Maurantonio Caprolu; mcaprolu@hbku.edu.qa

Received 25 November 2018; Revised 12 January 2019; Accepted 29 January 2019; Published 19 February 2019

Academic Editor: Angelos Antonopoulos

The Software Defined Networking (SDN) paradigm decouples the logic module from the forwarding module on traditional network devices, bringing a wave of innovation to computer networks. Firewalls, as well as other security appliances, can largely benefit from this novel paradigm. Firewalls can be easily implemented by using the default OpenFlow rules, but the logic must reside in the control plane due to the dynamic nature of their rules that cannot be handled by data plane devices. This leads to a nonnegligible overhead in the communication channel between layers, as well as introducing an additional computational load on the control plane. To address the above limitations, we propose the architectural design of FORTRESS: a stateful firewall for SDN networks that leverages the stateful data plane architecture to move the logic of the firewall from the control plane to the data plane. FORTRESS can be implemented according to two different architectural designs: Stand-Alone and Cooperative, each one with its own peculiar advantages. We compare FORTRESS against FlowTracker, the state-of-the-art solution for SDN firewalling, and show how our solution outperforms the competitor in terms of the number of packets exchanged between the control plane and the data plane—we require 0 packets for the Stand-Alone architecture and just 4 for the Cooperative one. Moreover, we discuss how the adaptability, elegant and modular design, and portability of FORTRESS contribute to make it the ideal candidate for SDN firewalling. Finally, we also provide further research directions.

1. Introduction

Distributed networking protocols running on routers and switches are the most important key technologies enabling digital information transmission. Despite their large diffusion, traditional networks are complex and difficult to manage [1]. In fact, to implement high-level policies, network administrators need to configure each single network device separately, using low-level commands with sometimes different syntax—depending on the hardware manufacturer. In addition to configuration difficulties, traditional networks struggle to meet the dynamic nature of modern networks. Indeed, enforcing policies in response to dynamic events such as failures or network load is a real challenge in traditional networks. The mismatch between network management needs and the state-of-play of classical networks management has led to the rising of a new network paradigm, called Software-Defined Networking (SDN) [2].

The main advantages of SDN networks include the following:

(i) *Intelligence and Speed.* Having both a global vision of network status and the total control on the whole network, the controller is able to optimize and distribute the workload—the payoff being an optimized use of resources and an increase in transmission speed.

(ii) *Easy Network Management.* The administrator remotely controls the whole network and is able to change its characteristics (e.g., services and connectivity) instantly and efficiently.

(iii) *Cloud Multi-Tenancy Enabler.* In a cloud environment, multiple users develop their own applications on different Virtual Machines (VMs). The Cloud Provider needs to guarantee every user the highest level of isolation. The SDN features are well suited to support this type of management.

(iv) *Virtual Application Networks.* Using SDN, it is possible to hide the physical details of the lower levels of networking, allowing users to focus on network tasks—i.e., addressing the *what* rather than the *how* [2].

The innovation introduced by the SDN architecture to computer networks also affects firewalls. Indeed, with the separation between the control plane and the data plane, infrastructure level devices become just "executors"; they simply forward incoming packets according to their flow table defined by the controller, without applying any advanced decision logic. This model is particularly suitable for stateless firewall implementation. In fact, the concept of "flow" allows forwarding devices to treat similar packets in the same way, rather than making individual decisions for each packet in an uncoordinated way. Every forwarding device, having a flow table appropriately configured with a particular set of static security policies, behaves as a distributed stateless firewall, as proposed by a few research contributions [3–7]. In this way, the firewall functions are implemented at the data plane, relieving the controller from the filtering burden, and leaving to it just the task to deploy static rules on each switch.

On the contrary, the implementation of firewalls with more advanced functions, like the stateful ones, is completely different: they need to store the status of every network flow, in order to forward only packets belonging to a legitimately established connection. This means that data plane devices are no longer able to filter out network packets using only the static flow tables. In fact, every switch in the network must forward a copy of every incoming packet to the controller before forwarding it to the destination. The controller (or other dedicated hardware in the control layer) analyzes the packet header to decide if it is legitimate or not, according to its information on the flows state. Then, the controller replies to the switch, communicating whether to forward the packet, based on its previous decision.

The advantage given by the SDN architecture to the development of stateful firewalls consists of the centralized logic. Indeed, using the standard OpenFlow APIs, it is possible to implement a single software-based stateful firewall that serves the entire network. By installing this type of firewall in the controller, it is possible to filter out packets of the entire network from a single device, rather than using many coordinated devices like in traditional IP networks. Despite the implementation advantages that the SDN architecture offers to the firewalls implementation, there is still one main pitfall; since both the stateful firewall logic and the filtering functions lie entirely in the control plane, this introduces a significant overhead in the communication channel. This is exactly the problem we address in this manuscript, providing the contributions detailed in the following.

Contributions. In this paper, we propose the architectural design of FORTRESS, a distributed stateful firewall for SDN networks that, in the Stand-Alone architecture, resides entirely in the data plane. Leveraging the stateful data plane properties, FORTRESS minimizes the network packet exchange between SDN layers. Our solution is presented in two configurations: the Stand-Alone Architecture and the Cooperative Architecture. The Stand-Alone Architecture is particularly suitable for data center network architectures and brings to zero the packets exchanged between SDN layers, as well as the computational power required on the controller side. The Cooperative Architecture, instead, is more flexible: it can be used in any network configuration and reduces to just 4 the packets exchanged per network flow.

We also compare our proposal against the state-of-the-art competing solution; the comparison showing the superior performance in terms of overhead and flexibility of FORTRESS. Finally, further research directions on SDN stateful firewalls are also discussed.

Roadmap. The paper is organized as follows. Section 2 provides some background concepts for SDN such as architecture, technologies, data center network topologies, and firewalls. Section 3 resumes the most important contributions in the area. The architecture of our solution is detailed in Section 4, while a discussion is provided in Section 5, where we compare our solution against the competing one, and we also introduce some future work. Finally, Section 6 draws some concluding remarks.

2. Technology Background

In this section, we provide a brief introduction of the main technologies touched throughout the paper. The section opens with an overview of Software Defined Networking, describing its architecture and the most important enabling technology (i.e., OpenFlow). Then, the concept of stateful data plane is introduced, being an essential part of our solution. A summary of the data center network topologies is provided thereafter, useful to support the architectural assumptions we made. Finally, we illustrate the different kinds of firewalls, together with their capabilities.

2.1. SDN Architecture. Software-Defined Networking (SDN) is a new networking paradigm where the control plane, responsible for deciding whether and how to forward a packet, is separated from the forwarding hardware. Designed to dramatically reduce the complexity of network management by encouraging innovation and evolution, SDN has immediately attracted the attention of both Academia and Industry. The basic idea is to allow developers to relate to network resources in the same simple way they interact with storage and computational resources. SDN centralizes network "intelligence" into one or more software controllers, making network devices simple forwarding hardware (i.e., data plane) that can be programmed via an open source interface [9].

Figure 1 shows the three different network architectures according to the control and data plane distribution. The traditional hardware-centric architecture, depicted in Figure 1(a), presents a vertical integration on each switch, which is a closed system that supports specific different control interfaces, depending on the manufacturer. In this type of architecture, distributing a new protocol or service (or simply

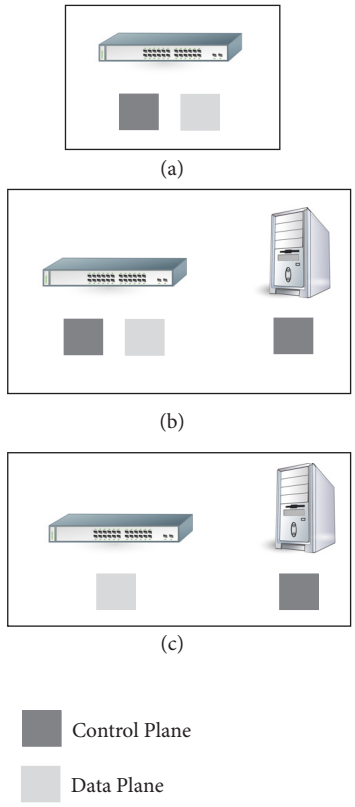

Figure 1: Networking architectures. (a) Traditional networking, (b) Hybrid networking and (c) Software defined networking.

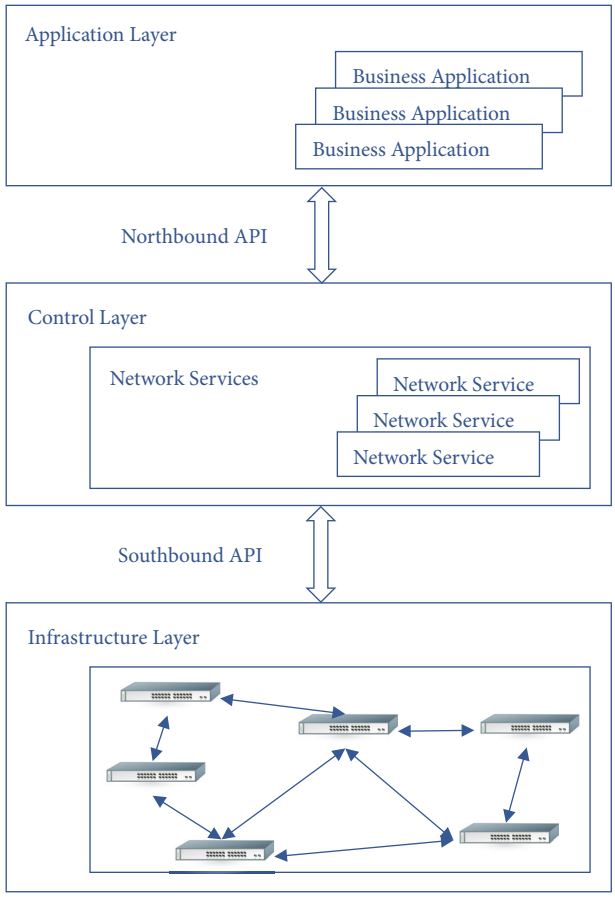

Figure 2: SDN architecture.

updating an already existing one) is a real challenge, since every single device has to be either updated or replaced. In the SDN architecture (Figure 1(c)) instead, with a clear separation between the control plane and the data plane, the switches become simple "executors" of rules established by the controller that controls the entire network. Finally, the hybrid mode, depicted in Figure 1(b), supports both the architectures [10].

Figure 2 shows the architecture of a standard SDN network that is vertically divided into three planes: *application plane*, *control plane*, and *data plane*, respectively.

The *application plane* level consists of one or more end-users applications that interact with the controller via open A-CPI (i.e., interfaces to allow communication between controllers and applications), using an abstract view of the network to program it, implementing different applications (e.g., Security, Traffic Monitoring). The *control plane* includes one or more software controllers that supervise the entire network, deciding whether and how to forward each individual packet in transit. This layer includes interfaces to allow communication between controllers and applications (A-CPI), between controllers and forwarding devices (C-CPI), and between different controllers (I-CPI). The *data plane* is the lowest level of architecture and consists of a set of forwarding devices, both physical and virtual, such as routers, switches, and access points [11].

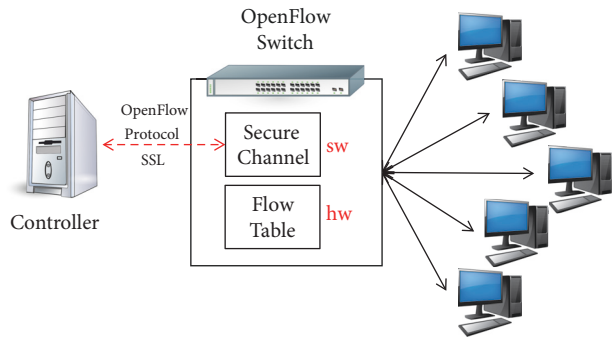

Figure 3: OpenFlow switch.

2.2. Enabling Technology for SDN. OpenFlow [12], considered the enabling technology of SDN networks, is the C-CPI type prevailing standard for the communication between the *control plane* and the *data plane*. Proposed by Stanford University, OpenFlow is a flow-oriented protocol that makes use of an abstraction of both switches and ports to manage network flows. Figure 3 shows an OpenFlow switch architecture. One or more flow tables are populated by the controller via the OpenFlow channel. Each entry of the table represents a routing rule that identifies a flow through one or more parameters (e.g., IP source, IP destination) and indicates the

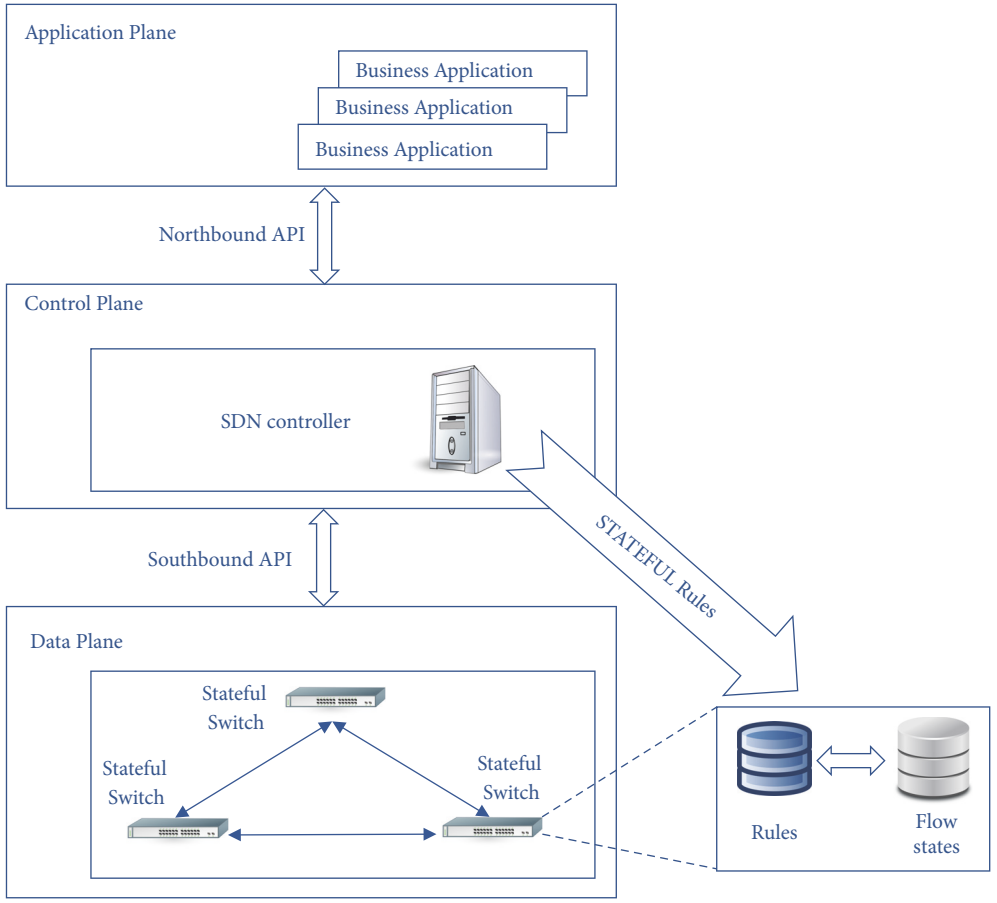

Figure 4: Stateful SDN architecture.

action to be performed (e.g., forward port x, drop). If a switch receives a packet for which none of the rules can be applied, the packet will be forwarded to the controller, which will make a decision about it and will also deploy a new switch rule to handle the same flow in the future.

2.3. Stateful Data Plane. The division between the control plane and the data plane introduced by the SDN network paradigm requires that one or more controllers send and update a set of forwarding rules on each switch, to handle data flows. This type of architecture greatly improves many aspects of traditional networks such as scalability, traffic monitoring, fault tolerance, and the possibility of applying stronger security policies. However, it introduces significant overhead on the communication channel between the data plane and the control plane, due to the packet exchange between SDN layers. Moreover, a not negligible increase in packet forwarding latency is introduced, due to the waiting times for the flow table updating. These limitations brought to a new research trend, born few years ago, allowing to move some control and stateful traffic processing from the controller to the switches.

The idea is to allow the programmer to both define and apply within the switches not only the stateless flow tables provided by the OpenFlow specifications, but also the stateful rules, i.e., forwarding rules that change dynamically in response to the flow transition from one state to another [13]. The term "state" in networking is defined as historical information that must be memorized to be used in processing future packets belonging to the same flow [14].

This new ability to configure stateful operations within the switches provides the data plane with a greater level of abstraction and programming expressivity. Figure 4 contains an overview of the SDN architecture with data plane stateful.

The concept underlying the stateful data plane architecture can be divided into two basic principles:

(1) Keep information on the state of each flow within the switch and allow to program state transactions according to the characteristics of the received packets.

(2) Allow the switch to independently make forwarding decisions based on the local state of the flow to which the incoming packet belongs, without the need to contact the controller.

The above principles imply that, as with traditional SDN architecture, routing decisions for a given flow are taken by the switches either on the basis of predefined rules imposed by the controller or independently on the basis of historical information previously stored for that flow [13].

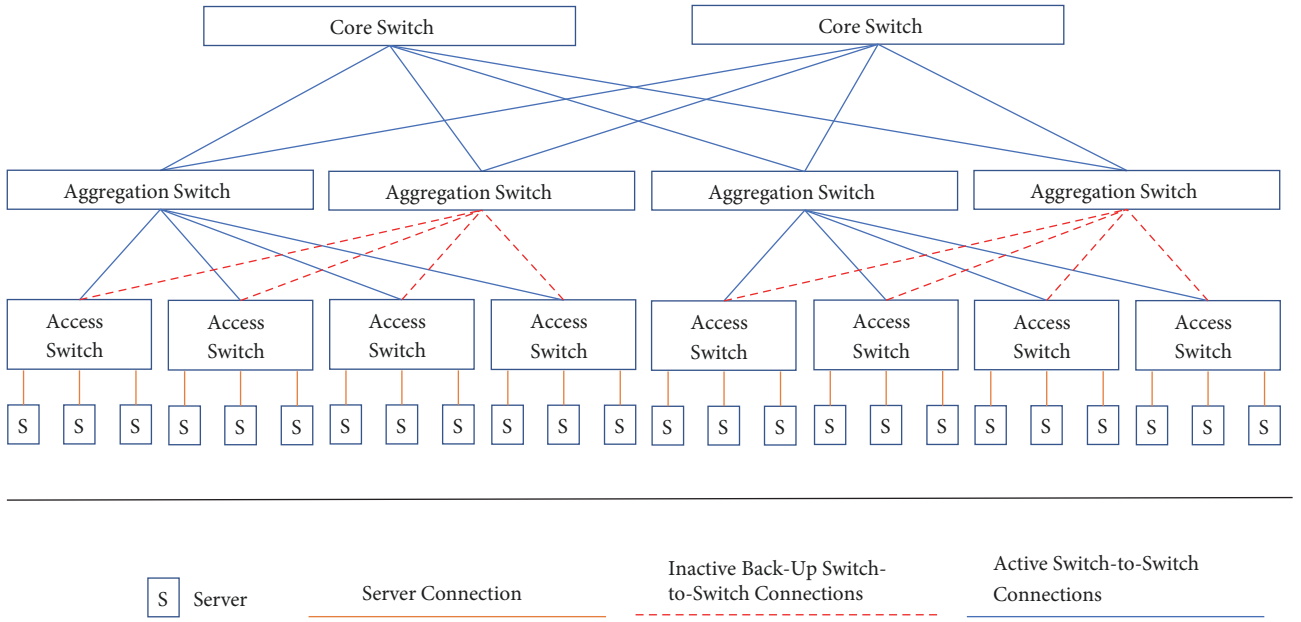

Figure 5: Data center design: Core-Aggregation-Access network topology [8].

2.4. Data Center Network Topologies. The evolutionary process of the hardware, which is becoming smaller, faster, and more powerful, is not enough to satisfy the ever increasing demand for computational power and storage space. Both companies and organizations, striving to comply with these requirements, are forced to outsource services to a Cloud Service Provider (CSP). The outsourcing procedure allows allocating responsibilities of managing services to a specialist third-party CSP that, in turn, aims at minimizing structural complexity by centralizing hardware and software components, instrumental to the management of the offered services, in facilities called Data Centers (DC)—consisting of servers, storage, and network devices, firewalls, as well as supporting components (e.g., backup equipment, and fire suppression equipment), power distribution systems, and cooling systems.

A data center network is the communication infrastructure used in a data center, directly identifiable by the network topology, the routing/switching equipment, and the protocols used (e.g., Ethernet and IP). The conventional data center network topology currently used in the data centers is the Core-Aggregation topology (Figure 5), also known as three-tier data center network.

In this topology, the Top-of-Rack (ToR) switch in the access layer provides connectivity to the servers mounted on every rack. Each aggregation switch (AS) in the aggregation layer (sometimes referred to as distribution layer) forwards traffic from multiple access layer (ToR) switches to the core layer. Every ToR switch is connected to multiple aggregation switches for redundancy. The core layer provides secure connectivity between aggregation switches and core routers (CR) connected to the Internet. A particular case of the conventional topology is the flat layer 2 topology, which uses only layer 2 switches [15].

2.5. Firewall. A firewall is a network security device that allows inbound and outbound traffic monitoring, using a predefined set of security rules to consent or block events. From an implementation point of view, a firewall can be hardware, software, or a combination of both. Depending on the monitoring functions it performs, it is divided into different types: packet filtering, circuit gateways, and application gateways [16]. Packet filtering firewalls (also known as stateless firewalls) work exclusively on the basis of the contents of the individual packets that pass through the firewall. The decisions are made based on the value of some flags (e.g., the source or destination IP address, the TCP port number of the source or the destinations, and so on) according to the rules set by the network administrator, without taking into account the "history" of the transited packets. Circuit gateway firewalls (also known as stateful firewalls), in addition to the same type of filtering performed by stateless firewalls, keep track of the connections established between the client and the server, blocking every packet that is not part of the connection. Application gateway firewalls instead, working at the application level, are able to provide more advanced functions as the packet is considered in its entirety, including payload. This requires the knowledge of the protocol of every single application that is necessary to monitor, as well as a specific implementation for each protocol. The innovation brought by the SDN architecture to computer networks closely involves firewalls. The concept of "Flow" allows infrastructure-level devices to treat similar packets in the same way rather than making individual decisions for each packet in an uncoordinated way. In fact, every forwarding device with the flow table appropriately configured with a set of static security policies behaves as a distributed stateless firewall [7]. The controller comes into play only to configure and (in case) update the policies; all

filtering work resides in the data plane. A different approach is required if we consider the implementation of a stateful firewall. With the separation between the control plane and the data plane, infrastructure-level devices become mere "executors" that handle (without making any decisions) the incoming packets according to the flows defined by the controller in their flow table. This model, which is well suited to stateless firewalls, complicates things in the case of a stateful firewall. In fact, the controller is the only entity able to modify the flow table of network devices in response to events such as an established or terminated connection, but to do so it must be able to analyze all the traffic. This implies that infrastructure-level devices must forward a copy of each packet received to the controller (or equivalent network devices that implement the stateful firewall), hence creating significant network overhead, a computational overload to the controller, and a delay in packet forwarding.

The introduction of the stateful data plane gives more autonomy to network devices. The ability to assign a state to each flow allows keeping the memory of past events, which will affect the forwarding of future packets. This technology makes it possible to implement a stateful firewall directly in the data plane. Whenever an established connection is recognized, the state of that flow changes. Consequently, even the future decisions about the packet forwarding for the same flow will change. This mechanism resides entirely within the infrastructure-level devices and does not involve the controller. As a result, the network overhead due to the forwarding of all traffic from the data plane to the control plane is eliminated, and packet forwarding time decreases because every time a connection is created, the network devices will manage the change of flow tables independently, without having to wait for the instructions from the controller.

3. Related Work

In this section, we provide a thorough analysis of the most representative related work in the area. The section opens with the introduction of the concept of firewalls in SDN technology. Then, stateful firewalls on SDN will be considered, focusing on FlowTracker, the state-of-the-art solution in the literature. The concept of stateful data plane is discussed later, together with a description of the state-of-the-art projects that have made the SDN architecture possible.

3.1. Firewalls on SDN. Firewall features definitely benefit in terms of both security and dynamicity from the network architecture introduced by SDN technology. In this new context, a static physical hardware is no longer required and can be replaced by a set of filtering rules that allow devices placed at the infrastructure level to block undesired network flows. Currently, there is a limited number of firewall technologies in the literature for SDN networks. In detail, all the proposed solutions fall into two categories: centralized firewalls and distributed firewalls, respectively.

In centralized firewalls, such as [3, 7, 17], all packets are forwarded to the controller, which acts as a centralized firewall. Among its burdens, the controller checks each packet according to the rules that are maintained in its tables and decides which packet to allow and which packet to block. Since the controller is the only entity involved in the operation of the firewall, this approach is very simple and it requires neither configuration nor maintenance of other devices' policies.

Other important challenges in SDN are policy-based detection [18] and security policy enforcement. Indeed, multiple OpenFlow custom applications running concurrently in the application layer may insert different control policies dynamically. These policies may contradict or override existing ones, introducing a challenge for the controller that must guarantee the absence of conflicts among rules [19]. Moreover, when a new flow rule is added to a flow table, it may generate intra-table dependencies [20]. Adversaries could use these dependencies to bypass firewalls, leveraging OpenFlow features to dynamically change the packet header.

In the context of centralized firewalls, some of these problems are addressed by FlowGuard [21], a framework designed to build more robust firewalls for SDN. FlowGuard takes into consideration different challenges of SDN-based firewall, like the ability to dynamically evaluate policy changes and conflict issues in flaw tables rules. FlowGuard's main goal is to detect and mitigate possible conflicts between firewall policies and flow rules. The proposed solution makes use of a violation discovery technique that, for every flow in the network, calculates its path space clearly identifying both its original source and its final destination, even if they have been modified during the crossing of the network. Then, FlowGuard calculates also the authorization space for that flow, if the path space and the authorization space have an overlap, a violation is detected and solved by applying one of the following resolution strategies: flow rerouting, flow tagging, flow removing, update rejecting, and packet blocking.

Despite the benefits of the advanced features provided by FlowGuard, centralized firewalls raise many concerns about both the computational overhead and performance scalability on the controller side. Moreover, adversaries can easily launch saturation attacks against the controller, as highlighted in [22].

Distributed firewalls, such as [23–27], use a different approach. Every network device that participates in forwarding packets is involved to "compose" the distributed firewall. Once the rules of filtering have been established, the controller installs the rules in the flow tables of each of these devices. In this way, the network devices will take care of the entire filtering work, lifting the controller from this computational burden. However, the firewall configuration will take more time and will be crucial, due to the migration of the filtering rules from the control plane to the data plane. Furthermore, the maintenance costs of the rules over time must also be taken into account [7]. The solutions proposed in [28, 29] also belong to this category, but differ from the different types of filtering rules installed on different types of devices. The work proposed in [30] addressed the firewall performance problem, but considering only the controller response time in the evaluation phase. The solution proposed

in [31] focuses on the replacement of physical switches by the virtual networking domain and also development of firewall and load balancing application on the control plane.

3.2. Stateful Firewall on SDN. The main function of a stateful firewall is to monitor the state of each connection on the network. Both incoming and outgoing packets are filtered based on the correspondence between the information contained within the packet's header and the active connections known by the firewall. A packet must be forwarded only if it belongs to an active connection. If it belongs to a nonexistent or terminated connection, it must be blocked. Several works, like [32], propose a stateful firewall for SDN network, but the logic resides on the control plane, causing nonnegligible overhead to the controller. In this context, two solutions deserve a detailed exposition.

3.2.1. SDFW. A recent contribution in this field is [33] that proposes a Stateful Distributed FireWall (SDFW) for the SDN architecture. The main goal is to solve the problems derived by the centralized nature of existing firewalls and the absence of flows state information in the data plane devices—this latter point making difficult to discover attacks originating in the data plane. The SDFW architecture is composed of one or more modules for every SDN layer. For the application plane, SDFW implements a user interface for the firewall configuration, defining high-level security policies and visualizing the state of the firewall. For the control plane, SDFW proposes some modules responsible for the translation of the user's security policies into OpenFlow rules, identifying possible conflicts between OpenFlow rules and security policies. These modules are as follows:

(i) *SDFW Manager*. It checks the status of every single firewall in the system and accepts the security policies written by users through the user interface.

(ii) *Network Information Base (NIB)*. It is a middleware between the application plane and the data plane, collecting data (for every switch) from the local event listeners.

(iii) *Policy-Graph Creator*. It is used by the control plane to check dependencies between different policies, creating a conflict-free Policy Graph.

(iv) *Traffic Statistic.* This module detects, for each switch, the internal events that could modify the network topology. When one of these events is detected, the module notifies the control plane.

Experimental results show that the proposed distributed model, compared with the traditional centralized one, is able to scale well on large networks. However, the core of the proposed solution entirely resides in the control plane. The distributed nature of the firewall is due to the use of data plane devices as a firewall component, but they still have to communicate with the middleware component, located in the control plane.

3.2.2. FlowTracker. FlowTracker [17] is a stateful firewall for SDN network that reduces the workload of the controller and the communication overhead between the control plane and the data plane without renouncing the accuracy and agility of the solution. FlowTracker, implemented as software installed on the controller, is able to manage both TCP and UDP connections. By analyzing the packets received from the controller, Flowtracker is able to deduce information about the topology of the network and to install rules in the switches accordingly. When a switch receives a new packet from one of the connected hosts, this packet is forwarded to the controller to know how to handle it. FlowTracker binds the source Mac address of the packet to the switch that forwarded it to the controller, deducing their direct connection. At this point, if the packet contains a flag that signals the start of a TCP connection, it will install on the switches the rules for the flow forwarding as per normal procedure. In addition to what is described, it will also install a rule for forwarding all traffic to the controller only on the switch directly connected to the source host. Conversely, if the active flag in a TCP frame is FIN, one party is requesting to terminate the connection, it will remove the rules from the switches. This solution mitigates the problems related to the network traffic congestion but, running on the control plane, still represents a computational overhead for controllers.

The separation between the control logic of the network (i.e., the control layer) and the forwarding hardware (i.e., the data plane) implies that the switches cannot detect and memorize the state of the network connections autonomously but must forward the packets to the control layer, potentially causing network congestion problems and computational overhead. To the best of our knowledge, the only contribution that attempted to mitigate this problem is [34] that implements a stateful firewall on the data plane. The implementation is made possible by leveraging a new Open vSwitch action proposed by [35]. Using this action, authors emulate the behavior of the stateful data plane. However, it has to be noticed that the proposed solution suffers from two main limitations: it handles traffic solely in one direction, and it is implementable only by using Open vSwitch.

3.3. Stateful Data Plane. In recent years, several implementations of data plane stateful architecture have been proposed that, although sharing the same basic principles, sometimes differ in other aspects, such as the implementation level (e.g., core platform programming languages and compilers, network-level framework, and so on) [13].

3.3.1. OpenState. OpenState [36] was proposed in 2014 as the first attempt to make the data plane programmable within proprietary platforms. The approach restricts the concept of stateful to OpenFlow's match/action rules. This allows obtaining an abstraction immediately applicable to any device, not necessarily open source. OpenState offers the ability to manage the states of a network flow within the switches using a simplified extended Finite State Machine (XFSM), known as the Mealy Machine. Formally it is a 5-tuple (S, S_0, I, O, T), where S is the set of states, S_0 is the

initial state (also known as Default), I is the finite set of input symbols (events), O is the finite set of output symbols (actions), and T: S x I ⟶ S x O is the transition function that maps a pair ⟨State, Event⟩ to a pair ⟨State, Action⟩. Since this is an abstraction of the OpenFlow rules in order to be multiplatform, the set O is restricted to the actions currently available on OpenFlow devices. However, the set I is restricted to the events that OpenFlow can identify by looking at headers and packet metadata.

3.3.2. Open Packet Processor. Open Packet Processor (OPP) [37] is another attempt to find a balance between configurable multiple hardware and flexible programming of the data plane. Although it is very different from the point of view of the implementation, Open Packet Processor is considered an evolution of OpenState, because it provides a stateful data plane to SDN networks using a full XFSM instead of the simplified version adopted by OpenState. The main objective is to provide a packet processing stage with the following properties: packet processing speed (in the order of a few nanoseconds), implementing a specific computing architecture for networking from scratch; efficient storage and management of stateful flow information, using a storage-oriented hash table called Flow Context Table, that allows obtaining stateful information of a flow in O(1); ability to specify and compute a wide class of stateful information, in addition to the primitive match/action of the OpenFlow packets processing pipeline, a series of stateful features, useful in traffic control applications; multiplatform, using an XFSM that allows hiding the hardware details related to the implementation of the individual primitives—hence, the programmer can simply combine them to achieve the desired behavior. The packet processing pipeline of the stateless data plane of a classic SDN architecture is limited to the original OpenFlow primitives. The XFSM used is a 7-tuple (I, O, S, D, F, U, T) where I is the set of input symbols (i.e., events), O is the set of output symbols (i.e., actions), S is the set of states, D is an n-dimensional linear space representing all the possible configurations of a memory register that includes registers dedicated to a flow or global, F is a set of functions of the type f_i: D ⟶ (0, 1) that define the conditions of the registers (i.e., Boolean predicates), U is a set of functions that define the possible operations of updating the contents of the registers, and T is the transition function.

3.3.3. Flow-Level State Transition. Flow-level State Transition (FAST) [38] is a primitive for supporting stateful applications on SDN networks data plane. The controller proactively installs a state machine on the switch that will allow it to memorize the state transitions of each flow autonomously. FAST consists of three main parts: an abstraction that allows programmers to create their own state machine for a wide variety of stateful applications; the FAST controller that translates the state machines into APIs for the data plane and the FAST data plane that includes a pipeline of tables for state machine support. In turn, the FAST controller consists of two components: the FAST compiler and the Switch Agent [14]. The FAST compiler translates the state machine defined by the programmer in a language that is understandable by the switch agent, which is charged with installing and monitoring the functioning of the state machine inside the switch. The switch agent knows the characteristics of the physical switch it is installed on, and how it supports FAST, and uses these pieces of information to translate the state machine using the switch API.

3.3.4. Stateful Data Plane Abstraction. Stateful Data Plane Abstraction (SDPA) [14] provides stateful support to the data plane on SDN networks by proposing a new "match-state-action" paradigm that contrasts with OpenFlow's "match-action". Unlike the other solutions, the SDN controller actively participates in the operation of the stateful applications implemented with this technology, effectively initializing state tables for each flow and communicating with the switch via the Forwarding Processor (FP). When a packet with no active flow arrives, it is sent to the responding controller that, in turn, indicates which state table should store the flow state. The subsequent packets belonging to the same flow will instead be managed by the switch itself. However, it is worth noticing that the controller maintains a total control over the management of changes to the state table, receiving periodic updates from FP.

3.3.5. Stateful Network-Wide Abstractions for Packet Processing. Stateful Network-wide Abstractions for Packet processing (SNAP) [39] is a framework for supporting stateful data plane over SDN networks that includes a programming language and a compiler. The main goal of SNAP is to provide network programmers with a tool that combines primitives of stateful operations with pure packet processing. Normally, to develop his/her own stateful application on SDN networks, a network programmer should manually distribute his/her own primitives on a large collection of independent hardware. The main strength of SNAP is to provide an abstraction that makes the entire data plane appear as a single large switch (OBS). In this way, network programmers can allocate global arrays within the OBS, without having to worry about how they are implemented or where they are placed in the physical data plane. These arrays can be indexed by one or more flags of the header of the incoming packet and modified over time as the network conditions change. Furthermore, if multiple arrays have to be modified simultaneously, the framework provides a sort of "network transaction" that atomically performs such updates. For these reasons, developing SNAP applications that react to network changes is very simple.

4. FORTRESS Architecture

Similarly to the concepts underlying the implementations proposed in the literature ([40, 41]) for monitoring traffic on SDN network with stateful data plane, our intuition is to transform each stateless network level device into a stateful one, able to analyze incoming packets and to take some decisions independently from the control layer. In this way, our solution could leverage every switch in the network to

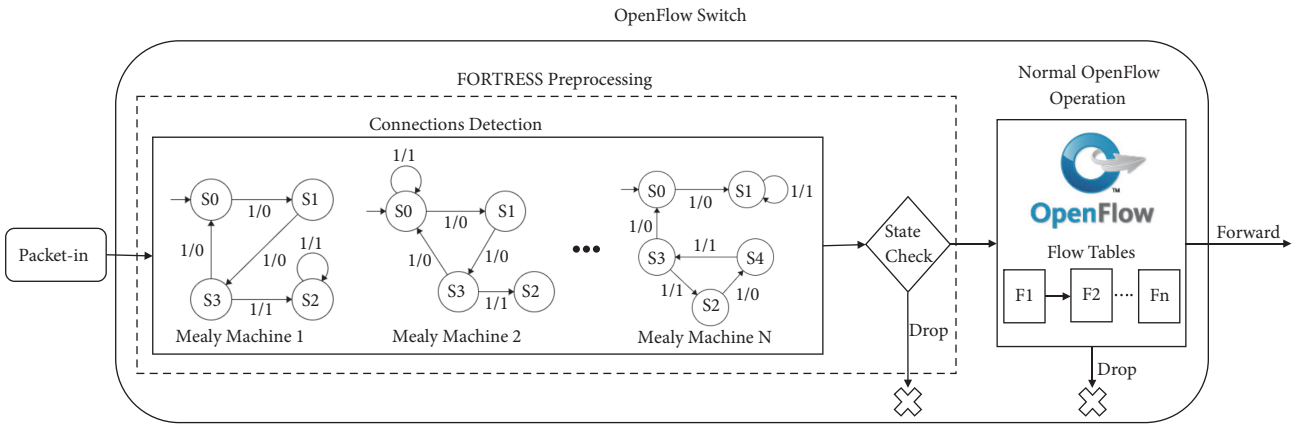

Figure 6: Detailed overview of the general FORTRESS architecture.

recognize and drop illicit network packets, allowing legitimate network flows only. The result is a distributed stateful firewall whose operations are independent of the control layer.

By definition, a stateful firewall keeps records of sessions and uses them to either allow or deny the forwarding of the packets within the network. This means that a stateful firewall has to implement the concept of sessions. In order to make data plane devices able to detect the establishment of a network session between two hosts, we use finite state machines to inspect incoming network packets. As depicted in Figure 6, the FORTRESS components (inside the dashed line) are installed within a classic OpenFlow switch. The "Connection Detection" component, checking the information contained in the packet header, eventually modifies the flow state according to its Mealy machines (one for each handled protocol). Then, the flow status is checked: if the incoming packet belongs to an established connection (or it belongs to the set of packets that are establishing a new one), it will be forwarded to the classic components of the switch. Otherwise, the packet will be directly dropped. If the packet is allowed by FORTRESS, the switch will decide whether to forward or drop the packet, according to the Flow Tables managed by the controller. In fact, FORTRESS works as a packets preprocessing stage, without interfering with the normal OpenFlow operations.

The TCP protocol is the most used and the most representative of the class of connection-oriented protocols. For this reason, we designed our stateful firewall to recognize TCP sessions, as depicted in Figure 7 and explained in Section 4.1, just to provide a real use case. Any other protocol belonging to the same class (connection-oriented protocols) can be managed by FORTRESS in the same way, by creating a specific Mealy machine for the recognition of the negotiation and closure phases of the connection and implementing it in the XFSM table. In this case, the FORTRESS Connections Detection component contains many Mealy Machines, one for every supported protocol, as shown in Figure 6. In such a scenario, the incoming packet preprocessing could happen in two different ways. The first option is to implement every Mealy Machine with a different XFSM table. This means that every incoming packet will be checked using different XFSM tables, sequentially. The second option is to generate a single Mealy Machine that represents the union of all the machines needed. This means that every incoming packet will be checked using only one XFSM table, but the overall complexity of the system increases. In this paper, we provide the design of the system with only one Mealy Machine (as a use case). The more convenient way to implement more than one Mealy Machine will be investigated in our future work.

4.1. Finite State Machine. The recognition of a new TCP session established between two hosts is made possible by the special mechanism provided by the protocol, called *three-way handshake*. It is sufficient to analyze the TCP headers of the incoming packet to identify the established or terminated connections. Let A be the host that would like to start the connection (also known as initiator) with the host B (also known as Listener). Host A sends a frame with the flag SYN equal to 1 and a random value j as a sequence number to B. Host B responds with a frame having both the flags SYN and ACK equal to 1, the sequence number equal to a random number y, and the acknowledgment number equal to j + 1. Finally, host A responds by sending to host B a frame with the ACK flag equal to 1 and the acknowledgment number equal to k + 1.

To recognize a TCP session, we make use of a finite state machine. Among the possible candidates, the most promising ones are the Mealy machine [42] and the Moore machine [43]. Our choice of a Mealy machine comes from the fact that it reacts faster to inputs. Indeed, while Mealy machines react in the same clock cycle, Moore machines require more logic, thus more circuit delays, which generally lengthens the reaction time. Moreover, with the introduction of the Mealy machine, we can open the doors to a future implementation on OpenState. Our implementation is therefore based on the insertion of a finite state machine inside the switches, similarly to what proposed in [44] for DDoS protection using OpenState [36] and in [45] for traffic classification using OPP [37].

Figure 8 shows the Mealy machine for the recognition of the *three-way Handshake*. The first state (Default) represents

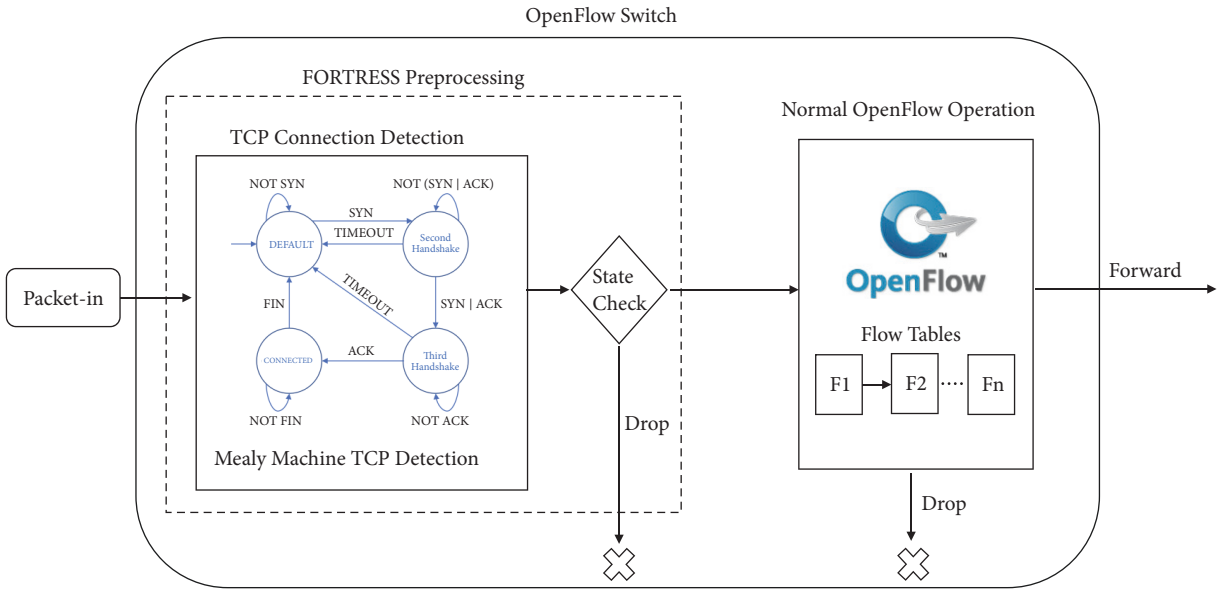

Figure 7: Detailed overview of the proposed FORTRESS architecture.

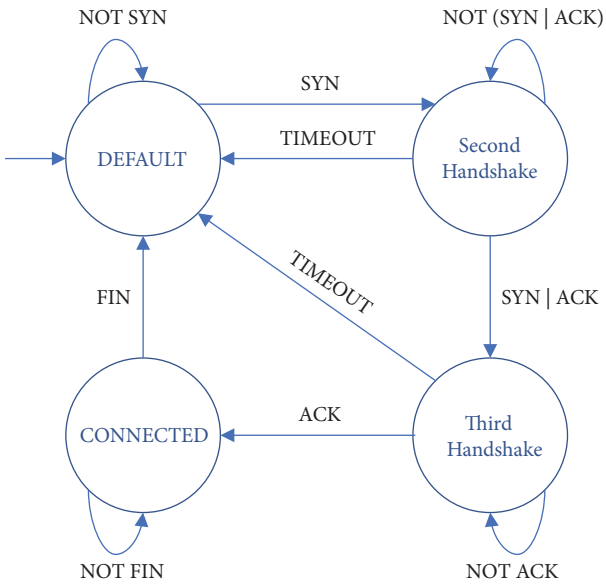

Figure 8: Mealy machine for three-way handshake recognition.

the state of the flow when a packet not belonging to a known established connection arrives. If the packet has the SYN flag equal to 1, a first connection attempt by a host is happening, and the packet is forwarded, making the flow state change to Second Handshake. This state indicates that the device is waiting for the second part of the connection procedure. In any other case, the packet is dropped. If a packet belonging to the same flow but in the reverse direction arrives, with both the SYN and the ACK flags equal to 1, the state becomes "Third Handshake". Finally, if a third packet arrives with the ACK flag equal to 1, the procedure is completed, the flow state becomes "Connected", and the two hosts are allowed to exchange any other kind of packet. This exchange of packets continues until a packet with the FIN flag equal to 1 arrives; from that moment onward the flow is considered close. During the session establishment, any other packet belonging to the flow but not corresponding to the illustrated requirements will be dropped.

4.2. System Design. The implementation is possible by using any of the stateful data plane enabling technologies described in Section 3.3. The design of the system will be presented below, modeled to be directly implemented with the simplest available technology: OpenState. In this way, our solution can be implemented also in more advanced frameworks, like OPP [37] or FAST [38], directly using the same design or with some modification to support additional features.

Our solution, like OpenState, employs a simplified extended finite state machine (XFSM), known as the Mealy Machine. Formally, an XFSM is an abstract model comprising an initial starting state S_0 (i.e., default) and a 4-tuple (S, I, O, T), where S is a finite set of states, I is a finite set of input symbols (i.e., events), O is a finite set of output symbols (i.e., actions), and T: S x I $\longrightarrow$ S x O is a transition function which maps the tuple ⟨state, events⟩ into the tuple ⟨state, action⟩ pairs [36].

The proposed architecture requires the following two tables:

(i) *State table*: a table that contains the state of each network flow.

(ii) *XFSM table*: a table that implements the XFSM for the three-way handshake recognition. In fact, for every set of state and event of the XFSM, this table contains the action to be performed, as well as any instruction on how to update the state table.

Table 1: Empty state table.

State Table	
Flow Key	State
*	Default

Table 2: State table after receiving ⟨x, y, a, b, (SYN)⟩.

State Table	
Flow Key	State
*	Default
⟨y, x, b, a⟩	Second Handshake

The Mealy Machine for a protocol connections detection, implemented by the XFSM table, detects all the incoming concurrent connections of the same protocol. The state of every flow with an established connection of that protocol is stored in the state table. The forwarding device does not need to instantiate other Mealy machines to handle connections for the same protocol.

When an incoming packet arrives to the switch, the following actions are performed:

(i) *State Lookup*. The state table is queried to identify the flow state to which the incoming packet belongs. This action is accomplished using either one of the packet header flags or a combination of them as a search key. In our case, the identifier of each flow is given by the combination of the following flags: ⟨IP src, IP dst, TCP src, TCP dst⟩. If the flow does not exist, a default state is returned.

(ii) *XFSM Transaction*. The XFSM table is queried by using, as a key, the flow state recovered in the previous step, and one or more flags of the packet header (e.g., SYN, ACK, FIN flags, also combined as appropriate). The action to be performed for the packet under examination is returned (e.g., forward, drop), with an optional indication on how to update the state table.

(iii) *State Update*. The state table is updated according to the indications obtained in the previous step, for example, updating the state of one or more flows or creating/deleting entries.

4.3. State Table. Let the tuple ⟨*IP_src*, *IP_dst*, *TCP_src*, *TCP_dst*, *(flags)*⟩ be a network packet representation, where *IP_src* represents the IP source address, *IP_dst* represents the IP destination address, *TCP_src* represents the TCP source port, *TCP_dst* represents the TCP destination port, and *flags* represents a set of TCP flags with value 1, respectively.

Tables 1, 2, 3, and 4 show the state tables for the recognition of the three-way handshake, while Table 5 represents the XFSM table. They illustrate how the state tables change throughout the connection negotiation process. Initially, the state table (Table 1) is populated only by an entry, representing a generic flow that does not belong to an active connection. Upon arrival of a generic packet ⟨x, y, a, b⟩, since there are no active flows to which it belongs, it is considered in the

Table 3: State table after receiving ⟨y, x, b, a, (SYN, ACK)⟩.

State Table	
Flow Key	State
*	Default
⟨y, x, b, a⟩	Second Handshake
⟨x, y, a, b⟩	Third Handshake

Table 4: State table after receiving ⟨x, y, a, b, (ACK)⟩.

State Table	
Flow Key	State
*	Default
⟨y, x, b, a⟩	Connected
⟨x, y, a, b⟩	Connected

"Default" state. For this state, the XFSM table requires the verification of the SYN flag within the packet header. If the SYN flag is equal to 1, the packet is forwarded, and an entry corresponding to the reverse flow ⟨y, x, b, a⟩ is added into the state table. The state of the entry will become "Second Handshake", in order to correctly handle the possible response (Table 2). Otherwise, the packet will be dropped. When the response arrives (i.e., the packet ⟨y, x, b, a⟩), the state table contains the state of the corresponding flow (i.e., "Second Handshake"), and the XFSM table requires the verification of both the SYN and the ACK flags. If they are both equal to one, an entry corresponding to the flow ⟨x, y, a, b⟩ is added to the state table, and the state of the flow will be promoted to "Third Handshake", in order to manage the remaining part of the negotiation procedure (Table 3). Finally, once the packet ⟨x, y, a, b⟩ arrives, and the ACK flag is equal to 1, the states of both the corresponding flows in the state table will be moved to "Connected" (Table 4).

As it can be seen from the XFSM table, hosts A and B can exchange packets of any type, provided that the FIN flag is different from 1. In case a packet with the FIN flag equal to 1 arrives, it means that one of the two hosts has requested the closure of the connection. As a result, both flows will be removed from the state table, and hosts A and B will no longer be allowed to communicate (with the exception of a novel three-way handshake exchange). Note that the rules contained within the XFSM table will drop all the traffic not conforming to the one scripted above, regardless of the flow state. The state table is also equipped with a time-out that eliminates inactive flows, a feature provided by the OpenFlow protocol, that cleans the table in different cases (e.g., the connection is open but unused, the connection closing packet is lost, one of the two communicating hosts has become unreachable, and so on). This architecture makes it possible to create the rules necessary for the recognition of the three-way handshake and to allow subsequent bidirectional communication only within the switches in the path between the two communicating hosts. If the path changes during the TCP connection, the packets would be passing through switches that do not contain their specific flow in the state table. In this case, the packets would be considered as malicious and

Table 5: XFSM table.

XFSM Table			
Match Flags		Actions	
State	Event	Actions	Next State
Default	SYN = 1	Forward	Second Handshake
Default	SYN ≠ 1	Drop	/
Second Handshake	SYN = 1 & ACK = 1	Forward	Third Handshake
Third Handshake	ACK = 1	Forward	Connected
Connected	FIN ≠ 1	Forward	/
Connected	FIN = 1	Forward	Default
.	.	.	.
.	.	.	.
.	.	.	.
First Handshake	SYN ≠ 1	Drop	/
Second Handshake	SYN ≠ 1 | ACK ≠ 1	Drop	/
Third Handshake	ACK ≠ 1	Drop	/

blocked. The fact that the path between two hosts remains unchanged throughout the duration of a TCP connection is a strong hypothesis, even for the SDN context. To overcome this limitation, we identify two distinct architectures: Stand-Alone Architecture and Cooperative Architecture. The Stand-Alone Architecture, particularly suitable for data centers network topologies, requires FORTRESS to be installed only in a small subset of switches (i.e., the peripheral ones), that will not change during the whole TCP connection. In the Cooperative Architecture, instead, switches rely on the controller to update their corresponding state tables. Every change in the state will be communicated to the controller which, in turn, has the burden to communicate this information to all the switches within the path.

4.4. Stand-Alone Architecture. In this architecture, FORTRESS works individually on each switch it is installed, without exchanging data either with the controller or with other switches—the emergent property is to have a single, distributed firewall. This architecture requires two assumptions on the network architecture.

Assumption I. The "peripheral switches" (i.e., those devices to which the hosts are/will be connected) are defined and known a priori.

Assumption II. The path between two hosts includes only two peripheral switches that remain unchanged throughout the communication: the switch directly connected to the source host and the one directly connected to the destination host.

These assumptions are directly met by any modern data center, as shown in the following example. Let us consider, for instance, the Core-Aggregation-Access network topology, shown in Figure 5, typical of the data centers discussed in Section 2.4. This particular design, like the Leaf-Spine model, reflects the first assumption because the hosts can only be connected to the access switches. Furthermore, the second assumption is easily achievable by carefully programming the control layer to include an access switch in the forwarding path, but only if either the source host or the destination host is directly connected to it. In such a network architecture, a packet can only enter the network across an access switch. For this reason, FORTRESS would be installed only on the access switches, which will filter the incoming and outgoing packets. Repeating the control on the higher level switches, which are therefore limited to simple forwarding, would be redundant. Negotiating a TCP connection between two hosts would cause the rules to be inserted only within the switches directly connected to the source and the destination. Since these switches are the only peripheral ones involved during the whole TCP session, the packets belonging to them can enter and exit the network without problems. If for any reason the path would change, only the upper-level switches will be affected. This is not a problem, considering that these switches only deal with forwarding the packets, not filtering. Figure 9 shows a FORTRESS deployment in a Core-Aggregation-Access network topology. Consider a network communication between the two servers A and B. All the possible links involved in this network session are represented with a solid line. A dotted line, instead, represents links that cannot be used to forward packets between host A and host B.

It can be seen that the only access layer switches between the two hosts are the ones directly connected to A and the ones directly connected to B, respectively. These two switches will update their state tables according to the XFSM during the three-way handshake between the hosts. Since all other possible switches in the communication path do not have FORTRESS installed, there is no need to communicate the flow state modification to the controller or to other switches. This system implements a stateful firewall that entirely resides in the data plane, bringing to zero the network packet exchanged between devices to install, modify, and delete filtering rules. Indeed, every switch autonomously handles its own state table, without sending/receiving any update.

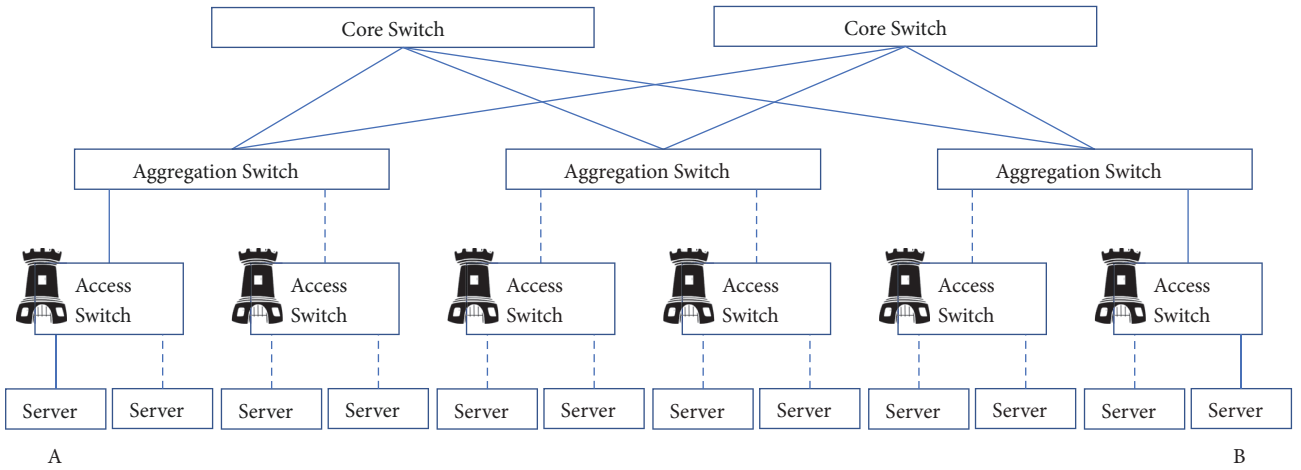

Figure 9: FORTRESS, Stand-Alone Architecture.

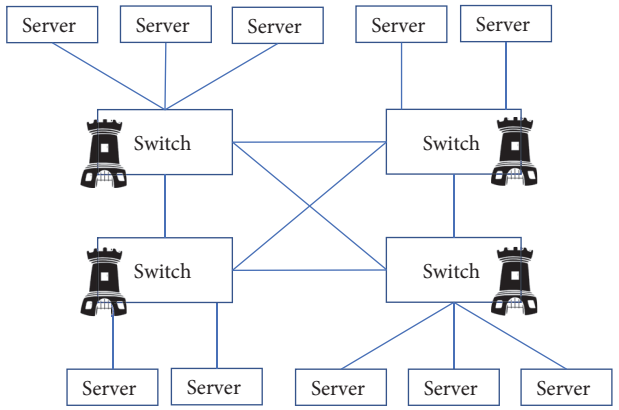

Figure 10: FORTRESS, Cooperative Architecture.

The controller is involved neither in the logic nor in the forwarding operations of the firewall, saving network and computational resources.

4.5. Cooperative Architecture. As discussed in Section 2.4, assumptions I and II fit the topology of the most common data center network architectures. For this reason, the Stand-Alone Architecture is particularly suitable for Data Center networks. In other topologies, especially in small networks, we cannot assume the use of "peripheral switches". In order to circumvent this problem, we present a second architecture, called "Cooperative Architecture". In fact, if the two assumptions mentioned above could not be guaranteed, FORTRESS functionalities could be still achieved, with some modifications in the deployment. In particular, it would be installed on all switches, as depicted in Figure 10. In the Cooperative Architecture, there is the need to establish a synchronization mechanism of the state tables among all the switches in the network. The FORTRESS switch, once a new flow has been identified (or a change happens in the state of an existing flow), updates accordingly its state table. Then, the switch sends the information to the controller that will forward them to all the other switches in the network. In this way, every switch has the knowledge about the status of every active flow in the network. In this architecture, every switch can handle the packets belonging to every legitimate active flow, even if it was not within the initialization path (i.e., the forwarding path used during the three-way handshake).

5. Discussion and Future Work

In this section, we first discuss the benefits of our solution, by highlighting the advantages to move the firewall logic from the control plane to the data plane. Later, we compare our solution against competing ones, in terms of packets exchanged between SDN layers—this parameter being of paramount importance for SDN firewalls. Finally, we highlight future research directions.

In the SDN architecture, data plane devices are no more than forwarders. They just handle incoming packets according to rules installed and managed by the controller, without any other advanced features. For this reason, firewalls, like any other complex application, must reside in the control plane in order to benefit from the global view of the network. Indeed, firewalls are typically installed directly on the controller, or in other devices (i.e., middle-ware) located on the control plane. As a result, the firewall must install a rule on data plane devices that, in turn, forward to the firewall every packet that needs to be analyzed. This creates huge traffic between SDN layers, especially in the case of stateful firewalls, that need to analyze every packet instead of switches statistics like other SDN applications. By moving the firewall logic from the control plane to the data plane, it is possible to avoid the packet forwarding between switches and firewall, keeping network links available for users network traffic and decreasing the congestion risk on the control plane.

Since the main goal of FORTRESS is to minimize the packets exchanged between the control plane and the data plane, our comparison with state-of-the-art solutions is focused on this aspect. To the best of our knowledge, the only stateful firewall implemented on the data plane is [34].

Table 6: Differences between FORTRESS and FlowTracker.

		FlowTracker	FORTRESS Cooperative Architecture	FORTRESS Stand-Alone Architecture
Layers involved	Data plane	✔	✔	✔
	Control plane	✔	✔	✘
# packets forwarded to the controller per session		All the traffic	4	0
Sessions detection	Control plane	✔	✘	✘
	Data plane	✘	✔	✔
Flow table modification	Control plane	✔	✘	✘
	Data plane	✘	✘	✘

As discussed in Section 3, it handles network traffic solely in one direction. This limitation makes the solution unusable in real world scenarios, so it has not been compared against our proposal. The firewall proposed by [33] instead is comparable with the proposed solution. Unfortunately, authors did not include any details about the number of messages exchanged between SDN layers, making impossible a comparison in these terms. Moreover, they do not provide any information enabling us to clearly infer these data—this because the focus of their solution is on the performance scalability in terms of network bandwidth.

Therefore, we compare FORTRESS with FlowTracker [17] that, to the best of our knowledge, is the stateful firewall implemented on traditional SDN architecture that minimizes the data exchange between data plane and control plane. FlowTracker integrates a firewall within the controller. The logic, therefore, lies in the control plane, where packets will be analyzed to recognize the established/terminated sessions. To do this, however, the controller needs first to analyze the whole traffic to find the three-way handshake and then the frame with the FIN flag equal to 1 that eventually closes the connection. For this reason, the first switch that receives a packet that does not belong to any flow is responsible for transmitting the whole traffic to the controller. Table 6 shows the differences between FORTRESS and FlowTracker. On the one hand, it can be observed how the implementation over stateful SDN networks of the FORTRESS firewall lifts the controller from the burden of both identifying the sessions and the corresponding modification of the flow tables for each switch, obtaining remarkable computational savings. In the Stand-Alone Architecture, FORTRESS completely eliminates the network overhead that is generated on classic SDN networks, due to the forwarding of all packets of each connection from the data plane to the control plane. On the other hand, in the Cooperative Architecture, channel usage is still very reduced, forwarding only four packets per session (i.e., three out of four for the three-way handshake initializing the communication, and the FIN packet that concludes it). Furthermore, FORTRESS is more reliable, indeed in case of unavailability of the controller, it would continue to work, an event that would not occur on classic SDN networks. Another difference involves the forwarding speed of packets. On classic SDN networks, the switch sends the first packet to the controller, waiting for a response before forwarding it. On the SDN network with stateful data plane this delay does not exist, because the switch automatically decides the course of action, forwarding or discarding the packet, according to its legitimacy.

As future work, we aim at implementing FORTRESS either using OpenState (for which it was designed) or by extending an existing OpenFlow switch (physical or virtualized). We will also provide other Mealy Machines to support different connection-oriented protocols, such as the Stream Control Transmission Protocol (SCTP) and the Quick UDP Internet Connections (QUIC) protocol, just to cite a few. Moreover, we will investigate the most performing way to implement these protocols inside the firewall. Another interesting direction involves the implementation of FORTRESS by using a technology for stateful data plane more expressive than OpenState (e.g., OPP or FAST). This would extend the functionality of FORTRESS, without changing its design. For instance, by using OPP we would have a complete XFSM available, instead of the simplified version used by OpenState. This latter situation is particularly interesting because the core of FORTRESS, i.e., the finite state machine for recognizing TCP sessions, would remain almost unchanged. In addition to this, by taking advantage of the ability to allocate additional memory for each flow, it is possible to enrich FORTRESS with several additional applications making it a next-generation firewall. In detail, statistics for each flow could be memorized, entropy measurements of some elements could be calculated, and, based on their variations, possible Denial of Service attacks or other relevant events could be recognized. Indeed, the adopted finite state machine approach can be easily modified to recognize any other protocol or extended to recognize multiple variants of the same one. Indeed, with the addition of few states, the Mealy Machine could be also made capable of detecting TCP Fast Open [46], a mechanism that enables the exchange of data during the TCP three-way handshake by reducing it from three to one round, thus decreasing network latency and data transfer delay.

6. Conclusion

In this paper, we have presented the architectural design of FORTRESS, a stateful firewall for SDN networks designed to run entirely within the data plane. FORTRESS supports two architectural designs: Stand-Alone and Cooperative. In the Stand-Alone Architecture, FORTRESS allows leveraging the features of a stateful firewall, hence completely eliminating

both the network overhead due to the forwarding of the packets to the controller, and the computational load on the controller.

Performance is evaluated in terms of packets exchanged between the control plane and data plane, comparing FORTRESS with FlowTracker, the state-of-the-art stateful firewall for SDN. Our proposal outperforms competition. In particular, the most efficient FORTRESS architecture, the Stand-Alone one, requires a striking 0-packets to be exchanged between the data and the control plane. Alternatively, the more general Cooperative Architecture can be used, requiring a network overhead of just 4 packets per TCP session. In comparison, FlowTracker requires to exchange *all* the session traffic. These performance results, combined with the elegant, adaptable, and portable design of FORTRESS, show the exceptional quality and viability of our proposed solution.

Finally, we have indicated future developments of this work, such as examining the portability of the proposed solution to more advanced technologies (e.g., OPP or FAST), with the goal of transforming FORTRESS into a next-generation firewall.

Data Availability

No data were used to support this study.

Disclosure

The information and views set out in this publication are those of the authors and do not necessarily reflect the official opinion of the QNRF.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This publication was partially supported by awards NPRP-S-11-0109-180242, UREP23-065-1-014, and NPRP X-063-1-014 from the QNRF-Qatar National Research Fund, a member of The Qatar Foundation.

References

[1] T. Benson, A. Akella, and D. Maltz, "Unraveling the complexity of network management," in *Proceedings of the 6th USENIX Symp. Networked Syst*, pp. 335–348, 2009.

[2] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: from concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[3] M. Suh, S. H. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in *Proceedings of the 16th International Conference on Advanced Communication Technology: Content Centric Network Innovation!, ICACT 2014*, pp. 744–748, Republic of Korea, February 2014.

[4] J. G. V. Pena and W. E. Yu, "Development of a distributed firewall using software defined networking technology," in *Proceedings of the 4th IEEE International Conference on Information Science and Technology, ICIST 2014*, pp. 449–452, China, April 2014.

[5] T. Javid, T. Riaz, and A. Rasheed, "A layer2 firewall for software defined network," in *Proceedings of the Conference on Information Assurance and Cyber Security, CIACS 2014*, pp. 39–42, Pakistan, June 2014.

[6] K. Kaur, K. Kumar, J. Singh, and N. S. Ghumman, "Programmable firewall using Software Defined Networking," in *Proceedings of the 2nd International Conference on Computing for Sustainable Global Development, INDIACom 2015*, pp. 2125–2129, India, March 2015.

[7] T. V. Tran and H. Ahn, "A network topology-aware selectively distributed firewall control in sdn," in *Proceedings of the International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 89–94, IEEE, Oct 2015.

[8] New switch architectures and the impact to 40/100g migration in the data center. http://blog.leviton.com/new-switch-architectures-and-impact-40100g-migration-data-center.

[9] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: past, present, and future of programmable networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[10] H. Farhady, H. Lee, and A. Nakao, "Software-defined networking: A survey," *Computer Networks*, vol. 81, pp. 79–95, 2015.

[11] M. Karakus and A. Durresi, "A survey: Control plane scalability issues and approaches in Software-Defined Networking (SDN)," *Computer Networks*, vol. 112, pp. 279–293, 2017.

[12] N. McKeown, T. Anderson, H. Balakrishnan et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[13] T. Dargahi, A. Caponi, M. Ambrosin, G. Bianchi, and M. Conti, "A Survey on the security of stateful SDN data planes," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1701–1725, 2017.

[14] S. Zhu, J. Bi, C. Sun, C. Wu, and H. Hu, "SDPA: Enhancing stateful forwarding for software-defined networking," in *Proceedings of the 23rd IEEE International Conference on Network Protocols, ICNP 2015*, pp. 323–333, November 2015.

[15] N. L. da Fonseca and R. Boutaba, *Cloud Services, Networking, and Management*, John Wiley & Sons, 2015.

[16] S. M. Bellovin and W. R. Cheswick, "Network firewalls," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 50–57, 1994.

[17] T. V. Tran and H. Ahn, "Flowtracker: A SDN stateful firewall solution with adaptive connection tracking and minimized controller processing," in *Proceedings of the 1st International Conference on Software Networking, ICSN 2016*, pp. 1–5, Republic of Korea, May 2016.

[18] N. Dayal, P. Maity, S. Srivastava, and R. Khondoker, "Research trends in security and DDoS in SDN," *Security and Communication Networks*, vol. 9, no. 18, pp. 6386–6411, 2016.

[19] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks, HotSDN 2012*, pp. 121–126, Association for Computing Machinery, Helsinki, Finland, August 2012.

[20] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header

space analysis," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 99–111, USENIX, Lombard, IL, USA, 2013.

[21] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: building robust firewalls for software-defined networks," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 97–102, ACM, 2014.

[22] S. Shin, V. Yegneswaran, and P. Porras, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks," in *Proceedings of the ACM SIGSAC conference on Computer & communications security*, pp. 413–424, ACM, Berlin, Germany, 2013.

[23] J. G. V. Pena and W. E. Yu, "Development of a distributed firewall using software defined networking technology," in *Proceedings of the 4th IEEE International Conference on Information Science and Technology, ICIST 2014*, pp. 449–452, IEEE, China, April 2014.

[24] T. Javid, T. Riaz, and A. Rasheed, "A layer2 firewall for software defined network," in *Proceedings of the Conference on Information Assurance and Cyber Security, CIACS 2014*, pp. 39–42, IEEE, Pakistan, June 2014.

[25] K. Kaur, J. Singh, K. Kumar, and N. S. Ghumman, "Programmable firewall using Software Defined Networking," in *Proceedings of the 2nd International Conference on Computing for Sustainable Global Development, INDIACom 2015*, pp. 2125–2129, IEEE, India, March 2015.

[26] S. Morzhov, I. Alekseev, and M. Nikitinskiy, "Firewall application for Floodlight SDN controller," in *Proceedings of the International Siberian Conference on Control and Communications, SIBCON 2016*, Russia, May 2016.

[27] A. Kumar and N. K. Srinath, "Implementing a firewall functionality for mesh networks using SDN controller," in *Proceedings of the 1st IEEE International Conference on Computational Systems and Information Technology for Sustainable Solutions, CSITSS 2016*, pp. 168–173, IEEE, India, October 2016.

[28] D. Satasiya, R. Raviya, and H. Kumar, "Enhanced SDN security using firewall in a distributed scenario," in *Proceedings of the International Conference on Advanced Communication Control and Computing Technologies, ICACCCT 2016*, pp. 588–592, IEEE, India, May 2016.

[29] A. Arins, "Firewall as a service in SDN OpenFlow network," in *Proceedings of the 3rd IEEE Workshop on Advances in Information, Electronic and Electrical Engineering, AIEEE 2015*, pp. 1–5, IEEE, Latvia, November 2015.

[30] M. S. Waheed, M. Al Mufarrej, M. Sobhieh, A. Al Barrak, A. Baig, and A. Al Mazyad, "Implementation of virtual firewall function in SDN (software defined networks)," in *Proceedings of the 9th IEEE-GCC Conference and Exhibition, GCCCE 2017*, Bahrain, May 2017.

[31] N. Zope, S. Pawar, and Z. Saquib, "Firewall and load balancing as an application of SDN," in *Proceedings of the Conference on Advances in Signal Processing, CASP 2016*, pp. 354–359, India, June 2016.

[32] V. Gupta, S. Kaur, and K. Kaur, "Implementation of stateful firewall using POX controller," in *Proceedings of the 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, pp. 1093–1096, India, March 2016.

[33] A. Chowdhary, D. Huang, A. Alshamrani et al., "SDFW: sdn-based stateful distributed firewall," https://arxiv.org/abs/1811.00634.

[34] P. Krongbaramee and Y. Somchit, "Implementation of SDN stateful firewall on data plane using open vswitch," in *Proceedings of the 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pp. 1–5, Nakhonpathom, July 2018.

[35] T. A. Hoff, "Extending open vswitch to faciliate creation of stateful sdn applications," https://cs.brown.edu/research/pubs/theses/ugrad/2016/hoff.timothy.pdf.

[36] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "Openstate: Programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 44–51, 2014.

[37] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, and C. Cascone, "Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing," https://arxiv.org/abs/1605.01977.

[38] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan, "Flow-level state transition as a new switch primitive for SDN," in *Proceedings of the 3rd ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking, HotSDN 2014*, pp. 61–66, USA, August 2014.

[39] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2016*, pp. 29–43, Brazil, August 2016.

[40] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *Proceedings of the IEEE Conference on Network Softwarization, NetSoft 2017*, pp. 1–4, IEEE, Italy, July 2017.

[41] C. Cascone, L. Pollini, D. Sanvito, and A. Capone, "Traffic management applications for stateful SDN data plane," in *Proceedings of the 4th European Workshop on Software Defined Networks, EWSDN 2015*, pp. 85–90, IEEE, Spain, October 2015.

[42] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Labs Technical Journal*, vol. 34, pp. 1045–1079, 1955.

[43] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata studies*, vol. 34 of *Annals of mathematics studies*, pp. 129–153, Princeton University Press, Princeton, NJ, USA, 1956.

[44] J. Boite, P.-A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," in *Proceedings of the IEEE Conference on Network Softwarization, NetSoft 2017*, pp. 1–9, Italy, July 2017.

[45] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *Proceedings of the IEEE Conference on Network Softwarization, NetSoft 2017*, pp. 1–4, Italy, July 2017.

[46] S. Radhakrishnan, Y. Cheng, J. Chu, A. Jain, and B. Raghavan, "TCP fast open," in *Proceedings of the 7th ACM International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '11)*, p. 21, ACM, December 2011.