Software Defined Networking Reactive Stateful Firewall

Salah Eddine S. E. Zerkane, Philippe Le Parc, Frederic Cuppens, David Espes

▶ **To cite this version:**

Salah Eddine S. E. Zerkane, Philippe Le Parc, Frederic Cuppens, David Espes. Software Defined Networking Reactive Stateful Firewall. 31st IFIP International Information Security and Privacy Conference (SEC), May 2016, Ghent, Belgium. pp.119-132, 10.1007/978-3-319-33630-5_9 . hal-01333445

HAL Id: hal-01333445
https://hal.univ-brest.fr/hal-01333445

Submitted on 21 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Software Defined Networking Reactive Stateful Firewall

Salaheddine Zerkane[1], David Espes[2], Philippe Le Parc[2], and Frederic Cuppens[3]

[1]IRT B<>COM, UBO, Télécom Bretagne,
35510 Cesson-Sévigné, France
Salaheddine.ZERKANE@b-com.com
[2] IRT B<>COM, UBO,
29200 Brest, France
{David.Espes, Philippe.Le-Parc}@univ-brest.fr
[3] IRT B<>COM, Télécom Bretagne,
35510 Cesson-Sévigné, France
Frederic.Cuppens@telecom-bretagne.eu

Abstract. Network security is a crucial issue of Software Defined Networking (SDN). It is probably, one of the key features for the success and the future pervasion of the SDN technology. In this perspective, we propose a SDN reactive stateful firewall. Our solution is integrated into the SDN architecture. The application filters TCP communications according to the network security policies. It records and processes the different states of connections and interprets their possible transitions into OpenFlow (OF) rules. The proposition uses a reactive behavior in order to reduce the number of OpenFlow rules in the data plane devices and to mitigate some Denial of Service (DoS) attacks like SYN Flooding. The firewall processes the Finite State Machine of network protocols so as to withdraw useless traffic not corresponding to their transitions' conditions.

In terms of cost efficiency, our proposal empowers the behavior of Openflow compatible devices to make them behaving like stateful firewalls. Therefore, organizations do not need to spend money and resources on buying and maintaining conventional firewalls. Furthermore, we propose an orchestrator in order to spread and to reinforce security policies in the whole network with a fine grained strategy. It is thereupon able to secure the network by filtering the traffic related to an application, a node, a subnetwork connected to a data plane device, a sub SDN network connected to a controller, traffic between different links, etc. The deployment of rules of the firewall becomes flexible according to a holistic network view provided by the management plane. In addition, the solution enlarges the security perimeter inside the network by securing accesses between its internal nodes.

Keywords: Software Defined Networking, Stateful Firewall, Security, Orchestration, TCP

1 Introduction

Classical networks are complex due to the lack of abstraction and due to the heterogeneity of the network infrastructure. They are costly in terms of deployment, maintenance and reconfiguration. Also, their structure is statically defined which makes tedious their provisioning and upgrading. In this context, Software Defined Networking (SDN) proposes new network architecture [1] to face the challenges of legacy networks. It is based on the physical separation of the data plane and the control plane.

The SDN architecture is organized in two layers. The data plane is responsible for forwarding the network traffic. It is organized into a set of SDN compatible devices connected to each other. The control plane embeds the network intelligence: the controller and the network applications. It is responsible for network configuration and for programming the data plane devices. It offers also an interface to the network applications, to enable them manipulating the data plane layer. They interact with the controller by a Northbound API which allows them also to collect network data and to transfer their commands to the controller, via a specific interface.

The controller interacts with the data plane via a standardized southbound API. OpenFlow [2] [3] is the most common interface. It enables the controller to install Openflow rules in the data plane layer and reprogram it through its flow tables. A flow table is a collection of flow entries. Each entry is a composition of matching fields, an instruction describing the way of executing a set of actions and many counters to keep traffic statistics. The data plane devices process the traffic according to their OpenFlow tables. The inward packets' headers are compared to the matching fields. If there is a correspondence between them then the instruction is executed. More, the controller can add, modify and delete flow entries. It collects the counters and may receive encapsulated packets in Openflow format (packet-in) from the data plane devices to process them.

Potentially, SDN will offer [4] advanced abstractions by adding visibility to network applications and services and by simplifying network administration. It will enable transparent levels of scalability while elevating user experience. It will save costs of network provisioning, deployment and maintenance. Additionally, it will enhance network agility by easing network function virtualization and automating network configuration.

SDN security is challenging and two sided [5]. On one side, SDN facilitates the development and the integration of flexible, efficient and controllable security solutions. It empowers security applications by providing them a network holistic view. However, on the other side, it introduces new vulnerabilities into the network. Some of them can have major impacts on the network. For example, breaching the controller will put the entire network beneath the attacker's control.

We propose a SDN stateful reactive firewall to protect the network from illicit access. SDN firewalls offer many advantages compared to traditional firewalls. They are cost effective because they enable to elevate the data plane with firewalling behavior. Thus, legacy firewall devices are no longer needed. They are also flexible since the controller can at any time reconfigure them and deploy them in any place. They offer a management interface for administrators to ease their tasks. Also, they

enable them to apply efficiently the network security policies in the data plane devices.

Many stateless SDN firewall had been proposed in the SDN realm. We are the first to propose an operational stateful SDN firewall. Moreover, our solution can also handle stateless communications.

The proposed firewall behaves in a reactive mode according to a generic algorithm. The later takes in entry the Finite State Machine (FSM) of any network protocol and produces the appropriate Firewall machine. In each transition, it incorporates as set of OF rules to express the corresponding action. We propose a first implementation to process TCP traffic. It receives connection synchronization packets, verifies their legitimacy against the security policies and validates them. The reactive behavior of the firewall saves flow table's space in the data plane devices by reducing the number of the installed flow rules. Besides, the firewall processes the traffic according to the states of the connection. For each connection's state, it receives only the traffic corresponding to the transitions from this state. This mechanism enables to restrict the traffic to useful communications and to mitigate some DoS attacks like Syn flooding.

Our solution is entirely integrated into the SDN architecture. We take full advantage of the SDN paradigm in terms of automatization, flexibility, abstraction and efficiency. In this regards, the firewall spreads dynamically the security policies according to its global view and adapts its behavior whenever the topology is updated. It installs its rules in any OpenFlow compatible device and enables the later to behave according to the access control decisions alike a firewall. Besides, it enables the user to express its policies without worrying about their installation and maintenance in the network. In addition, it enables to save costs related to repetitive firewall maintenance and provisioning tasks.

The architecture of the solution is as following. The application layer runs above the controller. It expresses the logic of the firewall. Below, in the data plane layer, we integrate a set of Openflow rules. These rules express the security policies according to OpenFlow. Besides the two conventional SDN layers, we propose a management level to orchestrate and reinforce the security policies. It enables the configuration of the firewall management and provides the administrator with a global view on the network.

The remainder of this paper is organized as follow. In section 2, we describe the state of the art of SDN firewalls. In section 3, we present the architecture and algorithm of our solution. We provide in section 4 the details of its implementation and the results of the performance tests. Finally, we conclude with some insights and related perspectives, in section 5.

2 Related Work

A firewall is a mechanism used to protect a network by filtering the traffic coming or going to an untrusted network [6]. It matches the packets' headers of the untrusted network with a set of security policies, and it filters them in order to allow only the accepted traffic to enter the network. A security policy is a set of filtering rules expressing the security policies of the organization [7]. Each filtering rule gathers 3 blocks. (1) A priority is used to determine the order of the rule's execution. (2) Many

matching fields enable the classification of a packet based on the values of its headers. (3) An action is applied to allow or deny the packet to its destination. There are mainly 3 types of firewalls [8] [9] [10]: stateless, stateful and application firewalls.

Stateless firewalls neither process nor keep in memory the different states of a connection. They do not take into consideration dynamic network information such as port source negotiation. Therefore, they are vulnerable and can be breached. Stateful firewalls have been introduced to resolve the shortcomings of the previous technology. They record in their memory the different states of a connection. They use, in addition, the attributes related to the states of a connection in their matching fields. Application firewalls are advanced stateful firewalls. They use application layer matching fields to classify packets and handle application level threats.

There are several works in the field of SDN stateless firewalls. Most of these solutions use Openflow rules to express the firewall security policies. The authors in [11] [12] [13] propose such SDN stateless firewalls. Their solutions forward to the controller the unknown traffic for processing. The controller then, parses the packet headers and it matches their values with the policy rules. The administrator can install the firewall policy rules in the data plane using the Openflow protocol. In this case the controller interprets these policies into Openflow rules and sends them to the data plane devices.

Besides, many SDN controllers propose their own version of stateless SDN firewall [14]. These firewalls lack of user graphical interface and are connectionless. FleXam [15] [16] [17] is an extension of Openflow which integrates a stateless firewall. It runs on the controller and provides a means to specify a set of flow filters on specified parts of a packet. Then, it applies the associated action to the packets.

Moreover, some SDN frameworks have been proposed to implement stateless firewall functions. FRESCO [18] [19] offers the possibility to instantiate predefined security modules and connect them together into a SDN stateless firewall. Flowguard [20] [21] is another SDN framework. It provides means to build Openflow stateless firewall rules into the data plane and to verify flow rule policy violations.

We have found in the literature one proposition [22] related to SDN stateful firewalls. The solution is based on Openflow and adds three new tables. This firewall keeps a table in the controller to save the connections' states and to synchronize the controller with the connection updates happened on the data plane tables. The other two tables are in the switch. One table manages the actual states of the connections and the other enables the data plane devices to process the next states. The limitation of this firewall relies on its excessive memory space consumption in the data plane and the volume of the generated traffic with the controller in order to keep it synchronized.

3 Firewall Design

Our solution is integrated into the SDN architecture and uses Openflow as a way to express the security policies. It is stateful since it records the states of the connections, and processes the information related to these states to protect the network. The behavior of the firewall is reactive. It reacts to the traffic by filtering packets and accordingly installs the appropriate Openflow rules to manage their connection.

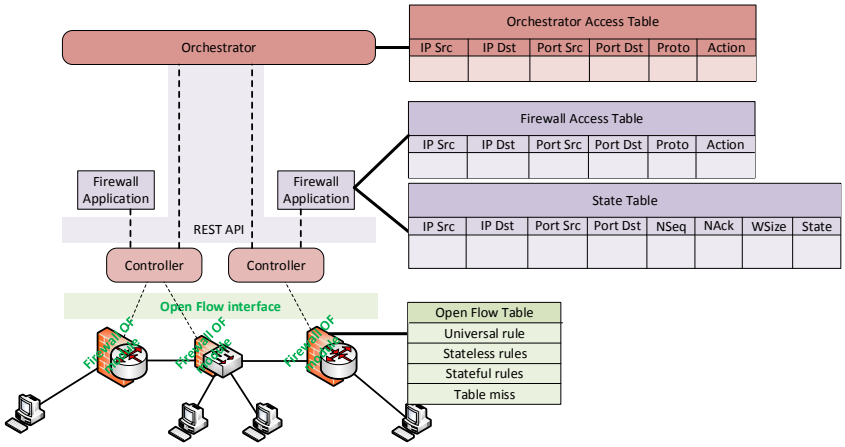

Fig. 1. SDN Stateful firewall general architecture

3.1 Firewall General Architecture

Our solution (see Figure 1) is distributed into 3 levels. (1) The higher level offers orchestration services including a security policies management. (2) A stateful firewall application is integrated on the top of the control layer. It is responsible for processing the states of the connections and installing the Openflow rules and filters connection requests according to the security rules. (3) An OpenFlow level expresses the security policies with OF rules which are installed in the data plane layer.

The orchestrator offers a management interface so that the administrator can express the security policies and access to the network global view. The orchestrator can be considered as a federation point because it collects the security policies and propagates them to the controllers. Also, it collects network data such as statistics and network logs and keeps them into its database. Based on these data, the orchestrator constructs a holistic network view including the network topology. In order to reinforce and propagate the security policies, it uses an Access Table. It contains all the stateful and stateless security policies specified by the administrator. The orchestrator manages this table to propagate the security policies into the network.

The orchestrator can also configure the behavior of the Firewall Applications dynamically. When the latter receives a new configuration, it loads the corresponding module and stops the old one. Such configurations options are the behavior mode: stateful or stateless, the event mode: periodic (according to a timer) or instantaneous (according to a sensor) and the topology discovering mode: static (topology data provided by the user) or dynamic (by learning dynamically the topology).

Each time a new controller joins the orchestrator, the latter sends to this controller the security policies that concern the part of the network it controls. These security policies are recorded in the Access table of the firewall application. Each controller is connected to the orchestrator by a Rest API. It enables any type of controller to interact with our orchestrator. It ensures also the interactions between each controller and its instance of the Firewall Application.

When a new data plane device joins the controller, the firewall application produces an OF universal rule and sends it to the new connected side. This rule matches with connection initialization packets and executes a forward to controller action. The firewall application also configures the data plane device by setting its table miss entry. In this case, all packets without a correspondence are dropped by default. Hence, each synchronization packet is sent to the controller which then transmits it to the firewall application. The latter then, verifies if the connection is legitimate or not using the Access table. In case the connection is rejected the packet is dropped.

Each instance of the firewall application uses a state table to record the connections' states and their attributes. This table enables the application to keep track of the connection, its state and its possible transitions to the next states. It is also used to create State OF rules in order to restrict the traffic only to the packets corresponding to the actual state and its possible transitions. This mechanism guaranties that the controller receives only the events triggering the transitions from the actual state of the Active connection. As an outcome, the Firewall Application reduces the load on the controller and mitigates some DoS attacks like Syn flooding. Because, we can restrict the number of synchronization requests for a connection, and clean the traffic from packets which are inconsistent with the connection state. For example, when a connection's synchronization succeeds, the firewall denies any further synchronization demand for it.

The data plane devices store in their Openflow tables (see Figure 1) the security policies in the Openflow rule structure. The universal rule matches with any synchronization packet and executes a forward to the controller action. The *Stateless rules* express the stateless policies of the firewall. The *Stateful rules* correspond to the stateful behavior of the firewall Application and the tables miss entry to process unmatched traffic. Except for synchronization packets, any other traffic is dropped in the data plane devices, if it is not corresponding to a legitimate connection state in the firewall application. Therefore it contributes to mitigate some DoS attacks since spoofed traffic will be directly dropped in the data plane devices.

The data plane devices perform firewalling behaviors by running the above OF entries. In the SDN architecture, each data plane device can be seen as a firewall from an external point of view. Thus, instead of using dedicated and specialized hardware, the SDN firewall elevates the behavior of the switch by reprogramming it according to the network security policies.

3.2 Firewall Generic Algorithm

The Class *Firewall_General_Behaviour* describes the generic algorithm of the firewall. The algorithm is thought in a way to process the Finite States Machine (FSM) of any communication protocol. It takes as entries the observed network events. Then, it verifies them with the preconditions of the actual FSM's state. If they fulfill the preconditions, it applies the corresponding actions. The firewall adapts the FSM's actions according to OF protocol by interpreting the original actions into OF rules. Then, it transits to the new state.

In our work we apply the generic algorithm to TCP communications. It has been instantiated with TCP states, transitions, their preconditions and actions. We also

encapsulate some transitions' actions with Openflow rules in order to comply with Openflow standard.

In the first step, the Orchestrator sends the Universal OF Rule and the settings of the table miss to all the Firewall Application instances. The following program code describes the structure of the universal OF Rule for TCP communications.

```
def Universal_OF_Rule():
1. action = OF_ActionOutput (FORWARD_CONTROLLER)
2. instruction = Instructions(OF_APPLY_ACTIONS, action)
3. matching_fields = OF_Match(eth_type = IPv4, IP_PROTO =
   TCP, TCP_FLAGS = (SYN) )
4. OF_RULE = OF_FlowMod (match = matching_fields, command
   = ADD_RULE, priority = 1, instructions = instruction)
5. send_msg (OF_RULE)
```

Through the controller, the Firewall Application is constantly listening to a potential connection of new data plane devices. It reacts to the network events by propagating the received Universal Rule and the settings of the table miss. It also, installs the Openflow Stateful Rules when the state transitions are triggered. The Firewall Application observes networks events according to two modes:

1. **Periodic Mode:** in this mode (see the *Firewall_Periodic_Mode Class*) the firewall sets a timer to observe periodically new network events coming from the controller. When the timer reaches its threshold, it sends a request to the controller to check if any new data plane device has been connected or if any update happened in any known data plane device. Once an alteration is observed, the firewall generates the corresponding Openflow Universal Rule and the configuration of the table miss. Then, it installs them on the new data plane device.

```
Class Firewall_Periodic_Mode :
1. Read (threshold)
2. While (true):
3.    Sleep(start_time=0, end_time=threshold)
4.    If (new_data-plane-device is connected):
5.      Send(Universal_OF_Rule,new_data-plane-device)
6.      Set(Table-miss(Drop),new_data-plane-device)
```

2. **Instantaneous Mode:** The firewall (see the *Firewall_Instantaneous_Mode Class*) puts in place a sensor into the controller to collect new network events coming into it. When the sensor detects a new data plane device, it prompts the firewall immediately. Then, the firewall generates the corresponding Openflow Universal Rule with the settings of the table miss and installs them on the new data plane device.

```
Class Firewall_Instantaneous_Mode :
1. While (true):
2.   Sleep(Waking_Event=Sensor_Notification)
3.   If Sensor_Event == New_Data_Plane_Device_connected :
4.     Send(Universal_OF_Rule,new_data-plane-device)
5.     Set(Table-miss(Drop),new_data-plane-device)
```

The firewall application uses one of the previous modes to update the data plane devices. When it receives a synchronization packet, it verifies its legitimacy. It checks in the access table if the connection is accepted or denied. If there is no specified policy for the connection in the table, it is denied by default. If the connection is accepted, an entry is created in the connection table. Then, the firewall verifies if the preconditions in the packet activate any of the transitions from the actual connection's state. If a transition is found, the firewall applies the corresponding actions associated with it, and then, it sends delete requests to the data plane device to remove the previous State Openflow rules. In the opposite case, the packet is dropped. Finally it updates the state in the state table and installs the new corresponding State OF rules. This mechanism lessens the load of the traffic into the controller, because the data plane devices send only the packets that can prompt the available transitions. Any traffic outside this zone is automatically dropped in the data plane device.

```
Class Firewall_General_Behaviour :
1. If (FW_Mode==Periodic):
2.  Firewall_Periodic_Mode();
3. Else:
4.  Firewall_Instantaneous_Mode();
5. While (true):
6.  C_Event=Collect(Controller_Events);
7.  If (C_Event.Type==Packet-in)
8.   Connection_Status =Check_Connection(C_Event)
9.   If (Not Connection_Status)
10.    Protocol_Data = C_Event.Protocol.Data
10.    Legitimacy = Check_Legimacy(Protocol_Data)
11.    If (Legitimacy):
12.     Create_Connection(Protocol_Data,State_Table)
13.    Else:
14.     Return;
16.  Else: #If Connection_Status == True
17.     Continue;
18.  Preconditions=Check_Preconditions(Protocol_Data)
19.  If (Not C_Preconditions.Status):
20.    Return;
21.  Else: #If the preconditions allow any transition
22.   C_Transition=Select_Transitions(C_Preconditions);
23.   Apply(C_Transition.Actions);
24.   Update(Connection_OF_Rules.Previous);
25.   Install(Connection_OF_Rules.New);
26.   Set(C_Transition.State,State_Table)
27. Else: #If C_Event.Type is not Packet-in
28.   Return;
```

4 SDN Firewall Proof of concept

We have implemented the firewall on the RYU [23] controller using the Python language. RYU is a software component SDN controller. It provides means to use multi-threading, to parse and serialize packets and to communicate with different data plane devices. We based our implementation on the instantiation of the generic algorithm for TCP. Then, we deploy a testbed to measure the performance of the SDN firewall.

4.1 Implementation

The firewall API (see Figure 2) comprises two packages. The orchestrator package runs in the management layer. It offers a General user Interface to manage the security policies and the OF rules. It allows adding, modifying and displaying the security policies and manages the static topology information. It offers to the administrator the possibility to configure many parameters of the firewall such as event modes, behavior modes, etc. It keeps open sockets with all the Firewall Application instances to communicate with them. Through these canals, it sends management commands and collects network events.

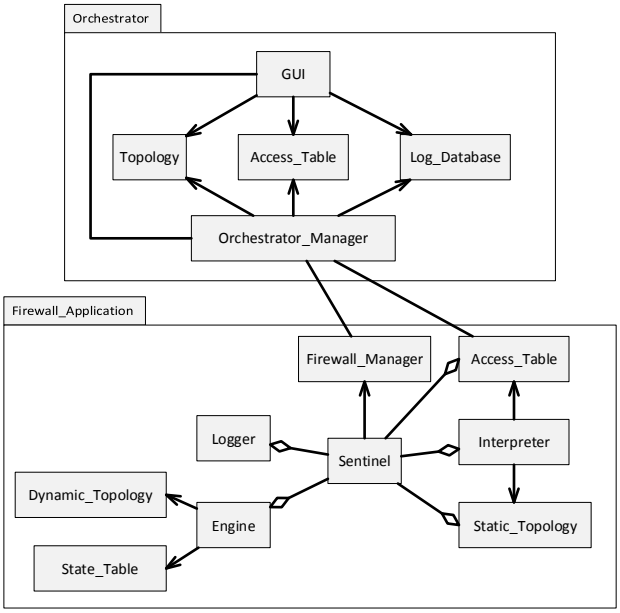

Fig. 2. Firewall API Class Diagram

The core package (*Firewall_Application*) is running on the Ryu Controller. It is mainly formed of the following components. (1) The *firewall Manager* Module keeps an open socket with the orchestrator. It sends the data coming from the other modules to the orchestrator and vice versa. (2) The *Interpreter* module translates the Adminis-

trator rules into Openflow rules according to the specification of the Openflow protocol. (3) The *State_Table* keeps the connection states, their properties and the firewall policies. (4) The *Logger* class collects information on the firewall components, connections data and traffic statistics. (5) The *Static_Topology* class provides data on the network topology. (6) The *Sentinel* singleton is responsible for the interaction with the controller. It configures also the firewall components and instantiates them. (7)The *Engine* class expresses the behavior of the firewall in handling all the phases of a stateful connection and in processing the communication between the client and the server.

4.2 Test & Results

We deploy and configure our test environment (see Figure 3) in mininet [24]. The latter is a Python application to emulate virtual networks. Our mininet environment comprises 3 clients and a HTTP server. All are connected by their virtual network interfaces to a virtual switch (OVS [25]). Each client runs a Python script which generates a number of simultaneous queries to request data from the HTTP server. The Ryu controller is remotely connected to the virtual switch via the virtual channel offered by mininet. It offers an execution environment to the Firewall Application and ensures all its interactions with the virtual environment. Besides we run the orchestrator and we connect it with a socket to the firewall Application. Our environment is running under Ubuntu 14, 64 bits, 2 GB of RAM and 2 processors at 2.8 GHz. The effective average latency between the controller and the switch is 0.25 ms, while the chosen bandwidth is 1 GB/s.

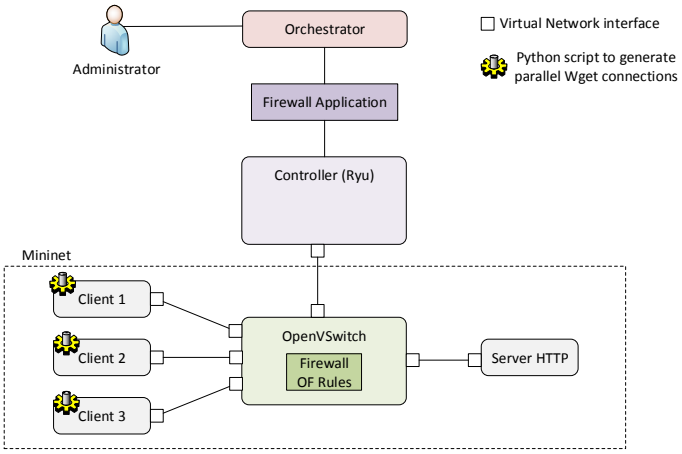

Fig. 3. SDN Firewall experiment Environment

We perform two different experiments in order to show the impact of the firewall on the connections' processing times and its effects on the user quality of service. In the first experiment (see Figure 3) we remove from the test bed the orchestrator and the firewall Application. Then, we activate the learning switch module of the Controller so that it behaves like a learning switch. In this case, the virtual switch sends the unmatched traffic to the controller. The latter maintains dynamically a table associating each IP and Mac host addresses with an OVS port number. When the route is

found in the table, the controller installs the corresponding OF Rules to enable the switch to forward directly the traffic to its destination. In case it does not find a port for the unmatched packet, it broadcasts the packet to all the switch ports and it waits for an answer to add the new correspondence.

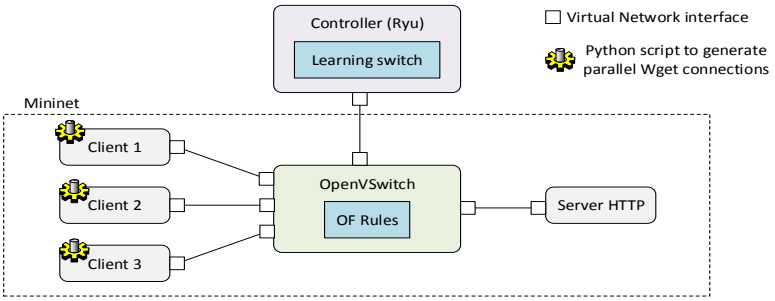

Fig. 4. SDN Learning Switch experiment Environment

In the second experiment (see Figure 3), we run the Firewall Application on the Controller. We connect to them the Orchestrator and we disable the learning switch module in order to let our firewall handling all the traffic.

In the two cases, the clients generate the same number of simultaneous TCP connections. We started the tests with 10 simultaneous connections and ended at 1000 simultaneous connections per second in a continuous and a constant interval of time. We perform in every experiment the following measurements: the average processing time of a packet-in and the average TCP connection time (the average time needed to process a complete TCP session). Furthermore, in the case of the experiment 2, we measured the maximum and minimum time of packet-in processing.

We analyze the data with two objectives. The first one is the performance of the firewall compared with a controller without a firewall (the learning switch controller). In this case, we are interested in observing how much extra time the Firewall needs to process the packet-in and the connections. Performance results are presented in Figures 5 and 6. In the second case, we focus on the scalability of the firewall by observing the evolution of the packet-in processing time zones with the number of simultaneous connections. The results are presented in Figure 7.

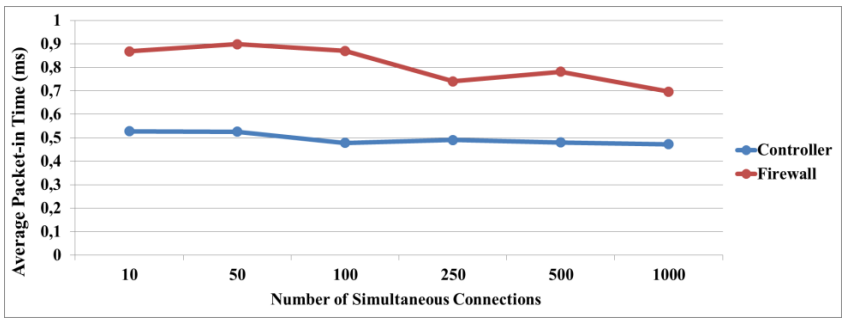

Fig. 5. Average processing times of packet-in

Figure 5 displays the Average packet-in processing time. We observe in both experiments a rather constant average time. The average packet-in processing time of the Firewall stays between 0.9 ms and 0.7 ms. For the learning switch controller it is almost constant around 0.5 ms. The firewall takes 0.3 ms more than the learning switch from 10 to 250 simultaneous connections then this extra time decreases to 0.2 ms till 1000 simultaneous connections. The time added by the firewall can be considered as inconsequential. It does not also inflate with the surge of the number of parallel connections.

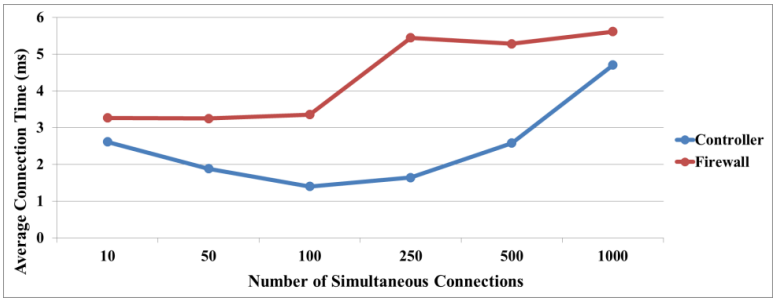

Fig. 6. Average TCP connections processing times

The results regarding the average TCP connections time are shown in Figure 6. In the case of the firewall the Average time is almost steady (around 3.3 ms) from 10 to 100 simultaneous connections while for the learning switch the average time decreases from 2.6 ms to 1.4 ms. From 100 simultaneous connections, the average time of the firewall increases till 250 simultaneous connections (5.4 ms) and then stays almost steady. While for the learning switch it continues in each step to increase reaching at the end a value of 4.7 ms. The processing time added by the firewall increases from 0.6 ms to 3.8 ms then decreases to 0.9 ms. The extra time added by the firewall in this case is also insignificant and scale very well with the increase of the number of simultaneous connections. In terms of Quality of Experience (QoE) this time is indiscernible for the user and does not reach the TCP timeouts values.

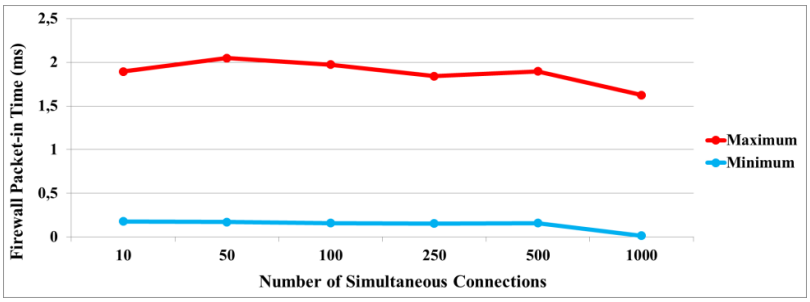

Fig. 7. Firewall Packet-in time values domain

In Figure 7, we observe the amplitude of the Packet-in processing time values for the firewall. It is almost steady (around 1.8 ms) all along the growth of the simultane-

ous connections. The interval of the packet-in processing time values is as following. The maximum values are between 2 ms and 1.6 ms while the minimum values are between 0.2 ms and 0.01 ms. The maximum and minimum values are related to RYU multithreading processing. The Engine threads are created by the Sentinel and then are put in a queue. If the queue reaches an important size, the packet-in processing time increases to the maximum values.

5 Conclusion

We introduce in this paper the first SDN reactive firewall. We speak about the advantages of our solution in terms of flexibility, performance, security enforcement and effectiveness. We discuss about its conceptual foundations based on a general algorithm specified for the TCP protocol. Finally, we show the details of its implementation, its deployment in a virtual environment and the results of the tests.

We add to the SDN architecture an orchestrator to manage the network according to a holistic view. We also integrate a Firewall Application which enforces the security policy.

In terms of performance, our solution adds a negligible delay to process packet-in or TCP connections. Regarding scalability, we show that the time processing does not increase with the number of simultaneous connections. These results are encouraging and confirm the effectiveness of our proposition.

We plan to consider the following enhancements in order to improve our solution. The first improvement will focus on the evaluation part. We will deploy the SDN test bed in a real environment. All the SDN elements will be hosted in dedicated powerful machines. We will push the firewall capabilities to their limits in order to measure the maximum number of connection that it can handle and the impacts on the network performances. In the second enhancement, we propose to develop a meta-firewall on the management plane. The orchestrator will instantiate it dynamically into different firewall applications and will place them in different SDN locations. This specialization will take into account the global view of the orchestrator, the spatial, historical and temporal context of the SDN network and the type of the network communication protocol.

6 References

1. D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky et S. Uhlig, Software-Defined Networking: A Comprehensive Survey, Proceedings of the IEEE, pp. 14-76, 2014.
2. The Open Networking Foundation, OpenFlow Switch Specification, 2014.
3. A. Lara, A. Kolasani and B. Ramamurthy, Network Innovation using OpenFlow: A Survey, IEEE communications surveys & tutorials, vol. 16, no. 1, pp. 493-512, 2014.
4. D. M. Jammal, T. Singh, A. Shami, R. Asal and Y. Li, Software-Defined Networking State of the Art and Research Challenges, Journal of Computer Networks, pp. 1-24, 2014.
5. L. Schehlmann, S. Abt and H. Baier, Blessing or curse? Revisiting security aspects of Software-Defined Networking, 10th International Conference on Network and Service Management, pp. 382-387, 2014.

6. R. K. Sharma, H. K. Kalita and B. Issac, Different Firewall Techniques: A Survey, 5th ICCCNT, 2014.
7. S. Zeidan and Z. Trabelsi, A Survey on Firewall's Early Packet Rejection, International Conference on Innovation and Information Technology, pp. 203-208, 2011.
8. H. Bidgoli, Packet filtering and stateful firewalls, Handbook of Information Security, Threats, Vulnerabilities, Prevention, Detection, and Management, New Jersey, John Wiley & Sons, 2006, pp. 526-536.
9. Z. Trabelsi, Teaching Stateless and Stateful Firewall Packet Filtering : A Hands-on Approach,16th Colloquium for Information Systems Security Education, pp. 95-102, 2012.
10. F. Guo and T.-c. Chiueh, Traffic Analysis: From Stateful Firewall to Network Intrusion Detection System, RPE report, New York, 2004.
11. J. Collings and J. Liu, An OpenFlow-based Prototype of SDN-Oriented Stateful Hardware Firewalls, in IEEE 22nd International Conference on Network Protocols, Chapel Hill, 2014.
12. J. G. Pena and W. E. Yu, Development of a Distributed Firewall Using Software Defined Networking Technology, 4th IEEE International Conference on Information Science and Technology , pp. 449-452, 2014.
13. C. Yoon, T. Park, S. Lee, H. Kang, S. Shin and Z. Zhang, Enabling security functions with SDN: A feasibility study, Computer Networks, vol. 85, no. 1389-1286, pp. 19-35, 2015.
14. M. Suh, S. H. Park, B. Lee and S. Yang, Building Firewall over the Software-Defined Network Controller, The 16th International Conference on Advanced Communications Technology, pp. 744-748, 2014.
15. S. Shirali-Shahreza and Y. Ganjali, Efficient Implementation of Security Applications in OpenFlow Controller with FleXam, 21st Annual Symposium on High-Performance Interconnects, pp. 49-54, 2013.
16. S. Shirali-Shahreza and Y. Ganjali, FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow, Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, pp. 167-168, 2013.
17. S. Shirali-Shahreza and Y. Ganjali, Empowering Software Defined Network Controller,IEEE International Conference on Communication, pp. 1335-1339, 2013.
18. S. Shin, P. Porras, V. Yegneswaran and G. Gu, A Framework For Integrating Security Services into Software-Defined Networks,2013 Open Networking Summit, 2013.
19. S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu and M. Tyson, FRESCO: Modular Composable Security Services for Software-Defined Networks, Network and Distributed System Security Symposium, pp. 1-16, 2013.
20. H. Hu, W. Han, G.-J. Ahn and Z. Zhao, Towards a Reliable SDN Firewall, Open Networking Summit, 2014.
21. H. Hu, W. Han, G.-J. Ahn and Z. Zhao, FLOWGUARD: Building Robust Firewalls for Software-Defined Networks, HotSDN'14, 2014.
22. W. Juan, W. Jiang, C. Shiya, J. Hongyang and K. Qianglong, SDN (self-defending network) firewall state detecting method and system based on OpenFlow protocol. China Patent CN 104104561 A, 11 August 2014.
23. RYU Team, component-based software defined networking framework, [Online]. Available: http://osrg.github.io/ryu/. [Accessed 27 August 2015].
24. Heller, B. Reproducible network research with high-fidelity emulation. Doctoral Thesis. Stanford University. 2013.
25. OpenVSwitch, [Online]. Available: http://openvswitch.org/. [Accessed 02 September 2015].