Building An SDN Firewall Application In ONOS

Chaoran Huang, Lexuan Sun, Chunpeng Shao, Yang Cao

***Abstract*—Cyber-attacks lead to massive loss of people's wealth as the rapid development of Internet. In this proposal, we proposed a firewall application with ACL(Access Control List) rules based on software-defined network platform ONOS(Open Network Operating System) which is more flexible, extendable and accurate. Also, we discussed some advanced features (e.g. examining dynamic network policy updates, checking indirect security violation and stateful monitoring) that may be implemented to our firewall application.**

***Index Terms*—ONOS, Firewall, Software-Defined Networking, Network, Protocols, Software, Proposal, Network Security**

I. Introduction

A. SDN

SDN technology, whose core idea is taking advantage of open and vendor-neutral interfaces with the separation of control and data planes achieves programmable control and the independence between network hardware and software. By managing network resources and providing high-level abstraction and APIs, SDN provides an open platform to simplify the creation of new applications and serve a variety of hardware networks. According to the definition of Open Network Forum (ONF), SDN architecture can be divided into three main levels from bottom to top: Application layer, control layer and Infrastructure layer.

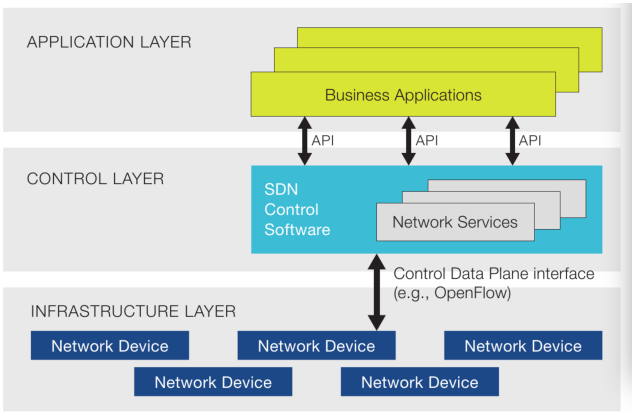


Fig. 1. SDN Architecture

In fig. 1, The control layer is the control plane of the SDN network logic set, which is composed of the cluster server, namely the SDN controller. The main functions of the controller include topology management, path calculation, traffic statistics, flow table management, northbound and southbound interface, log and alarm, security and reliability. The controller is responsible for the conversion of the network abstraction function to the actual forwarding. The controller receives the resource request and capability call from the application layer through the northbound interface, and converts it into a forwarding policy that can be executed by the physical network. Meanwhile, it is sent to the Infrastructure layer device through OpenFlow or other southbound interface protocol to perform the corresponding operation. Take the OpenFlow protocol as an example, the protocol specifies the multiple matching rules that the device forwards according to the flow table to meet the functional requirements of all aspects of the network. Below the control layer, it is the Infrastructure layer. The Infrastructure layer mainly provides flexible programmable forwarding capability. This layer needs to pay attention to the hardware performance of the device and achieve the high-speed forwarding of the data. These specific forms include physical forwarding devices and virtualized forwarding devices. Above the control layer is the application layer. The application layer is the key to SDN promotion. Based on SDN open architecture, a variety of innovative network applications can achieve business fast opening, resource assurance and improve network resource utilization by analyzing, arranging, and scheduling the basic network resources. Application layer communicates with the control layer through the north interface. It is not necessary for Application layer to understand the specific structure of the network, but it needs to define abstract information model.

C. Huang and Y. Cao were with the Department of The School of Computing. C. Shao and L. Sun with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634 USA e-mail: chaorah@clemson.edu chunpes@clemson.edu lexuans@clemson.edu cao9@clemson.edu.

B. OPENFLOW

OpenFlow is a communications protocol which gives access to the forwarding table of a network switch or router over the network. It is the most used open-source protocol which meets the SDN concept in the communities. The core of OpenFlow technology is making use of OpenFlow switches and control servers complete independently the forwarding of packets, which has replaced the forwarding process of traditional switch or router control packets. In fact, this conversion happens due to the change of control. For the traditional network, the traffic flow is directed artificially. Whats more, the switch and the router has no concept about data flow, which only has exchanges between the packets. However, in the OpenFlow network, the traditional routers is replaced by an unified control server which determines how the packets are transmitted over the network. The OpenFlow defines the interaction protocols between the controller and the routers, as well as a set of router operations. This controller-router protocol runs on a secure transport layer protocol or an unprotected TCP connection. The controller sends instructions to the routers in order to control the forwarding of packets, as

well as configuring parameters. The router sends a message to the controller when the link is interrupted or a packet without forwarding instruction occurs.

A flow table consists of flow entries. The main components of a flow entry in a flow table is posted in table I, where each flow table entry contains: [1]

1) Match Fields: which matches against packets.
2) Priority: which matches precedence of the flow entry.
3) Counters: which is updated when packets are matched.
4) Instructions: which modifies the action set or pipeline processing.
5) Timeouts: which represents the maximum amount of time or idle time before flow is expired by the switch.
6) Cookie: which represents opaque data value chosen by the controller.
7) Flags: which alter the way flow entries are managed.

TABLE I
MAIN COMPONENTS OF A FLOW ENTRY IN A FLOW TABLE

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

The match fields and priority can identify a unique flow entry in a specific flow table. When all of the match fields of the flow entry can match its header fields and pipeline fields, a packet matches a flow entry.

C. SDN SECURITY

There are some issues for implementing full-scale SDN. And one of these issues, SDN security is the most important aspect that is beginning to receive attention.

1) Unauthorized Access: Centralized control is one of SDNs characteristics. More than one controllers can access the data plane of the network in the SDNs functional architecture. Applications from multiple source applications can be linked to the controller pool. The controller provides an abstraction for the application so that the application can read/write network status. If an attacker emulates a controller/application, it can gain access to network resources and manipulate network operations.
2) Data Leakage: An attacker can detect the action applied to a particular grouping type by analyse the packet processing timing. For example, the time is different between input port to output port and controller processing that the attacker could know the configuration of the switch with a set of designed fake packets. After known what the rules are, the attacker could then craft a volume of fake request which would lead to DDoS attack. [2]
3) Data Modification: The controller application has the privilege to program the network equipments to control the traffic flow in SDN environment and if an attacker could hijack the controller, then the whole system is under his control and in further change the rules running in the devices. And, the security mechanism should be used in every device if intermediate entities are introduced between data planes and control planes for provisioning of virtual networks.
4) DDoS: DDoS has been a serious problem today in the network environment. And as mentioned in the characteristics of SDN part, one core characteristic of SDN is that the data plane and the control plane are logically separated, which gives attacker a way to generate DDoS attack against the controller application. The attacker could craft a set of packets that requesting a flow rule decision and make it unavailable to legal users, and use this set of packets to flood the controller. And as the flow table would cost resource, another way to DoS attacks this system is to flooding the flow table, which could also down the system. [3]

D. FIREWALL

Firewalls are the most widely deployed security mechanism in most businesses and institutions. A conventional firewall sits on the border between a private network and the public Internet, and examines all incoming and outgoing packets to defend against attacks and unauthorized access. [4]

E. ONOS

Open Network Operating System project is an open source community hosted by The Linux Foundation. ONOS is written in Java and provides a distributed SDN applications platform atop Apache Karaf OSGi container. The platform provides applications with a number of high-level abstractions, through which the applications can learn about the state of the network and through which they can control the flow of traffic through the network.Applications (core extensions) can be loaded and unloaded dynamically, via REST API or GUI, and without the need to restart the cluster or its individual nodes. [5]

F. ACL

Access Control List (ACL) consists of access control entries which can specify the access rights allowed, denied, or audited for an identified trustee. ACL can control if a routed packet is forwarded or blocked at the switch's interfaces so that the network traffic will be filtered. Based on the criteria configured in the access lists, the switch determines each packet to be forwarded or dropped. In order to improve the security of our network, it is necessary to configure ACL so that we can constraint the contents of routing updates and control the traffic flow. However, if ACL is not configured on our switches, it is possible that all of the packets can pass through the switches and access every part of our network. One of the important features of ACL is it can allow one host to access some parts of our network, meanwhile, prevent another host to access the same fields. For instance, in fig. 2 on the following page, when ACL is configured, host A can be allowed to access destination network while host B is blocked from accessing the same destination network. Also, taking the advantages of ACL, types of traffic either to be forwarded or dropped can be decided. For instance, allow email traffic and meanwhile prevent all Telnet traffic. [6]

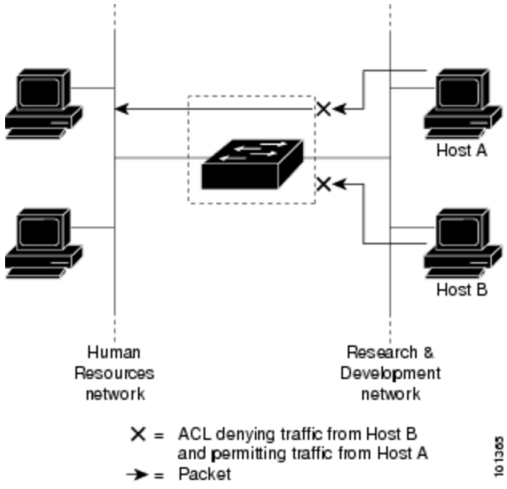


Fig. 2. Instance of Using Traffic Filters to Prevent Traffic from Accessing a Network

Generally, most protocols require at least two basic steps to configure ACL, which includes creating an access list definition and applying the access list to an interface. Two access lists can be applied to an interface, such as inbound access list and outbound access list for some protocols. The autonomous access control mechanism allows effective access control of information resources and provides independent protection of the confidentiality and integrity of the information. The owner-controlled access control mechanism is only valid according to the user's requirements. All users must know how access permissions are authorized and denied and how these are set up. For example, an ACL which is associated with a file system object (file or directory) can enforce access rights to relevant users. Such ACLs can enforce different levels of access (such as read or write) for different users. Typically, each object will have a defined owner and, in some cases, associated with the primary group. The owner of a particular object controls its freely determined access properties. The owner's property setting is used to create a valid user ID for the process. Many of the objects in the operating system (such as sockets and file system objects) have ACLs associated with different topics. The details of ACLs for these object types may differ from each other.

II. Developing and Testing Environment

A. Operating system

We use Ubuntu 16.04 as our developing and testing environment, and the ONOS development guide also recommends it. So, At first, we create a virtual machine using Virtualbox with Ubuntu 16.04 system. We allocate the virtual machine 6G memory and 4 processors. (See fig. 3.) For the network adapter, we select NAT (Network address translation).

B. Prerequisites

1) Mininet: Mininet is a well-used network emulator for creating switches and virtual hosts. The switches support OpenFlow, and the host can run Linux network software freely.

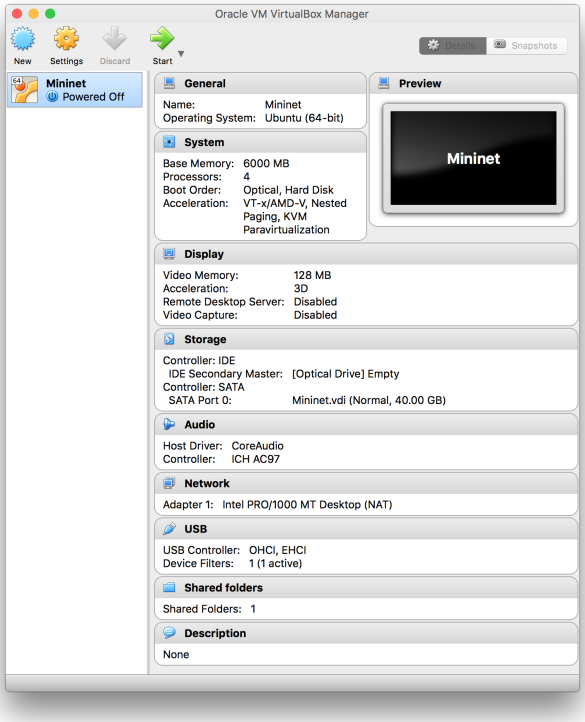


Fig. 3. Virtualbox Configure

So we can use Mininet to generate our network structure to test the firewall functions.

```
sudo apt-get update
sudo apt-get install mininet bridge-utils
```

2) JDK 8: The ONOS is written in Java, and the developing guide recommends developers using JDK 8 as their developing environment. So we need install it by running the following commands:

```
sudo apt-get install software-properties-
    common -y
sudo add-apt-repository ppa:webupd8team/java -
    y
sudo apt-get update
echo "oracle-java8-installer shared/accepted-
    oracle-license-v1-1 select true" | sudo
    debconf-set-selections
sudo apt-get install oracle-java8-installer
    oracle-java8-set-default -y
```

3) Git: Git is one of the most famous version control system softwares. The ONOS repository is hosted on Github, so we need git to clone the ONOS source code and checks out the particular version as our SDN controller. At the same time, we also need use Git and Github to collaborate and manage our firewall source code.

```
sudo apt-get install git
sudo apt-get install git-review
```

Maven: The ONOS is a Java Maven project, and it also provides many scripts to generate the ONOS Maven hello world project helping the developer to build their ONOS

application. So we need it to build the ONOS and our firewall application.

```
sudo apt-get install maven
```

IntelliJ IDEA Community: it is a well-used Java IDE (integrated development environment) for developing computer software. ONOS provided many supports on configuring and importing the ONOS source code to this IDE. It is really helped for us to understand the code and to write the firewall application for ONOS.

```
https://www.jetbrains.com/idea/download/
```

C. Build ONOS 1.8.4

We use ONOS 1.8.4, so at first, under the home directory, we cloned the entire ONOS project from Github and checked out a particular version marked with tag 1.8.4 as a new branch. Then we used the provided build tool to compile ONOS.

```
git clone https://github.com/opennetworkinglab
    /onos
cd onos
git checkout tags/1.8.4 -b 1.8.4
tools/build/onos-buck build onos
```

D. Running ONOS with Mininet and onos.py

ONOS provided a Mininet script (onos.py) to generate ONOS SDN testing environment quickly. The following commands will generate a Mininet network with a tree topology of depth 2 and fanout 2 (i.e. 4 hosts connected to 3 switches) using Open vSwitch switches under the control of one ONOS controller. (See fig. 4.)

```
cd ~/onos/tools/dev/mininet
sudo mn --custom onos.py --controller onos,1
    --topo tree,2,2
```

Then we used the following command to get controller IP:

```
onos1 ip address
```

The controller IP was 192.168.123.1. (See fig. 5.)

Next, we tried to login the ONOS WebUI by browsing http://192.168.123.1:8181/onos/ui with username ‘onos’ and password ‘rocks’. And we ran the following command on mininet to let the ONOS learn the entire network. (See fig. 6 on the following page.)

```
pingall
```

On the ONOS WebUI, we can see the network structure after we ran pingall and turned on the host visibility. (See fig. 7 on the next page.)

The hosts h1-h4 IP were from 10.0.0.1 to 10.0.0.4 respectively.

The ONOS default applications (config from onos.py) was shown on fig. 8 on the following page

Till now the developing and testing environment were fully configured and were ready to use.

```
vm@vm-mininet:~/onos/tools/dev/mininet$ sudo mn --custom o
nos.py --controller onos,1 --topo tree,2,2
[sudo] password for vm:
*** Creating network
*** Adding controller
*** Creating network
*** Adding hosts:
(unpacking /tmp/onos1)onos1
*** Adding switches:
cs1
*** Adding links:
(onos1, cs1)
*** Configuring hosts
onos1
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(s1, s2) (s1, s3) (s2, h1) (s2, h2) (s3, h3) (s3, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0 *** ONOS_APPS = drivers,openflow,fwd,proxyarp,mobility
*** Starting controller

*** Starting 1 switches
cs1
(starting onos1)
onos1(checking: karaf. ssh-port.(1 warnings).(2 warnings)
openflow-port.(3 warnings)..... client node-status)
*** Waited 18.74 seconds for ONOS startup
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet-onos>
```

Fig. 4. Running ONOS with Mininet and onos.py

```
mininet-onos> onos1 ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue stat
e UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: onos1-eth0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1
500 qdisc noqueue state UP group default qlen 1000
    link/ether 3e:e8:78:1b:3a:b1 brd ff:ff:ff:ff:ff:ff lin
k-netnsid 0
    inet 192.168.123.1/24 brd 192.168.123.255 scope global
 onos1-eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::3ce8:78ff:fe1b:3ab1/64 scope link
       valid_lft forever preferred_lft forever
```

Fig. 5. Controller IP

III. Design and Implementation

A. ONOS Structure

ONOS as the one of the most useful SDN system in this world it supports the responsibility of update and hold the flow table content, which means the ONOS system will hold a flow table content by themselves that contain all information about flow table and the relations between the device, which mostly means SDN switcher, and specific flow table rule. Because the effective of the specific rule of flow table is time sensitive, the ONOS will record all the run that is send by the API supported by ONOS, and check their effectiveness periodically. Based on the support of ONOS, the application which based on ONOS should just use the API supported by ONOS and do not need to care about the effectiveness of these rules of flow table they send.

```
mininet-onos> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Fig. 6. Pingall Test

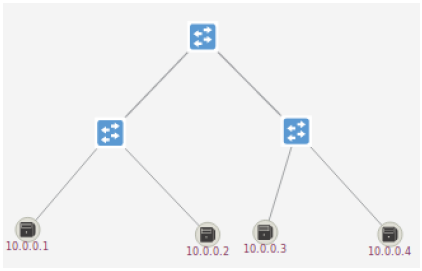


Fig. 7. ONOS WebUI Network Structure

For achieving this demand, ONOS system has separate several parts to solve this problem.

1) Application Web Resource: The REST API is the most important interactive method between user and ONOS system. This process is actually based on the Servlet of Java.

As the fig. 9 and following command shows, the API interfaces is actually in the AppWebResource file, which is actually the one catch the REST API request from user, and it generate the specific service of ONOS to deal with the data update, or other request.

```
curl -sSL --user karaf:karaf -X GET http
    ://192.168.123.1:8181/onos/onos-acl-
    firewall/sample
```

2) Service and Serialization: The Service is an important part of ONOS system. ONOS system has a lot of application, and these application may has same function or work. So Service of ONOS system equals to an specific module exported from one application to support specific function. And because based our demand, we need to search the ACL rules in specific ruleId, which means we have to hold a rule table by our own. This table should contains enough information to search the flow rules in ONOS flow table pool, and to update it.

Because this demand, we have to make new container to serialization our data of ruleId, this feature is only supported by Service of ONOS, which give a way to serialize the data.

3) Flow Table and Access Control List: The flow table control and daemon is another important part of ONOS system, which hold and update a pool of active flow table rules activated by user. Every flow rule activated by ONOS API will be inserted into the pool of flow table rules in ONOS system. Thus, we should use the API supported by ONOS system to implement a flow rule.

About the Access Control List, because the ONOS system actually has own ACL system, it just use a simply way to implement the ACL by change the specific detail of content in flow table rules. So we can use the same way to build our Access Control List.

Default device drivers	org.onosproject.drivers	1.8.4	Drivers	ON.Lab
Host Location Provider	org.onosproject.hostprovider	1.8.4	Provider	ON.Lab
Host Mobility App	org.onosproject.mobility	1.8.4	Utility	ON.Lab
LLDP Link Provider	org.onosproject.lldpprovider	1.8.4	Provider	ON.Lab
OpenFlow Meta App	org.onosproject.openflow	1.8.4	Provider	ON.Lab
OpenFlow Provider	org.onosproject.openflow-base	1.8.4	Provider	ON.Lab
Optical information model	org.onosproject.optical-model	1.8.4	Optical	ON.Lab
Proxy ARP/NDP App	org.onosproject.proxyarp	1.8.4	Traffic Steering	ON.Lab
Reactive Forwarding App	org.onosproject.fwd	1.8.4	Traffic Steering	ON.Lab

Fig. 8. ONOS Default Pre-enabled Applicaitons

```
/**
 * Get hello world greeting.
 *
 * @return 200 OK
 */
@GET
public Response test() {

    ObjectNode node = mapper().createObjectNode();
    AdvAclService advAclService = get(AdvAclService.class);
    List<AdvAclRule> rules = advAclService.getAdvAclRules();
    for(AdvAclRule rule : rules){
        node.put(rule.getId().toString(), rule.toString());
    }
    return ok(node).build();
}
```

Fig. 9. Example of REST API

4) Support IP Format: For the ONOS system, the API of control flow table rules is FlowService, which just accept the Classless Inter-Domain Routing (CIDR) IP format to search specific IP. Because the FlowService is required for implementation of our demand, but the input of CIDR format IP is easy for program process but hard for human being, so we need another format to input IP and another function to translate the input IP format to a list of CIDR IP format.

B. Implementation

Because the application develop is based on the structure of ONOS system, so our implementation has separated into several parts as below.

1) Design of REST API Interface: We have design four main function of our application.

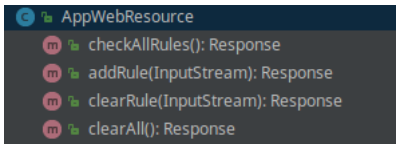


Fig. 10. Function List of REST API

As fig. 10:

- **checkAllRuels** shows the all inserted rule by user.
- **addRule** take a json input, which contain information of new ACL rule. And it will add into this system.
- **clearRule** take a json input, which contain a specific id of acl rule you want to delete, and it will delete it.
- **clearAll** will remove all the ACL rules inserted by user.

2) Service: Because Service is a necessary part in our application, and it just can generate in WebResource file. So

actually it is generated by factory method in ONOS system, which just generate a specific class and handle it by a interface. So we need to implement our class by implement some interface we design firstly.

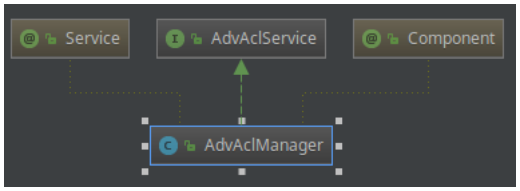


Fig. 11. Function Service implements relations

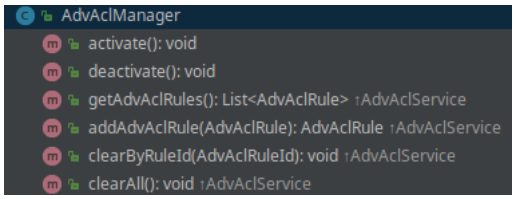


Fig. 12. Function offered by our Function Service

As the fig. 11 and fig. 12 shows, the AdvAclManager is our implement of AdvAclService. This is the functional service of our application, which contains the function we add our rule into our serialization service and ONOS flow table pools. Besides, this service contains another function to delete specific rule by id and clear it in ONOS system pool.

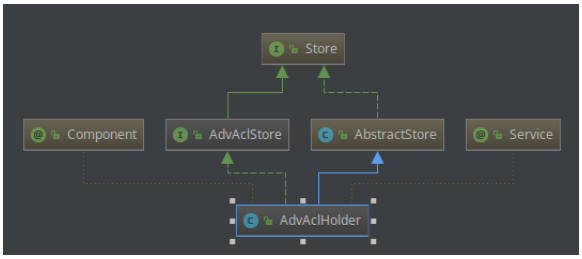


Fig. 13. Serialization Service implements relations

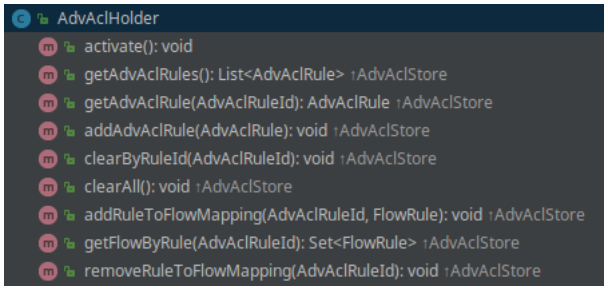


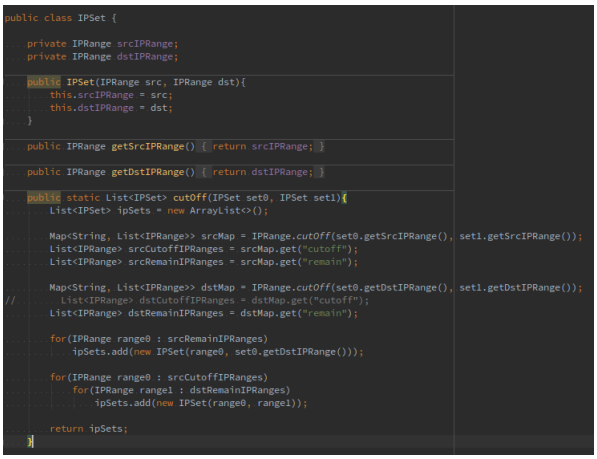
Fig. 14. Function offered by our Serialization Service

As fig. 13 and fig. 14 shows, the class AdvAclHolder which implements the AdvAclStore and AbstractStore is our serialization service. This service support us function and containers to serialize data made by ourselves. This means we can create an ACL rule list by ourselves to maintain the information of rules we inserted.

3) IP Format and Segmentation: Because in our application we have deal with the basic ACL function and advanced feature Examining Dynamic Network Policy Updates, which means we should check our existed rule when every time we insert a new rule. And if the new rule have overlapped with these old rules, what we should to do is cut off the part of old rules and only insert the non overlapped part.

And because the CIDR format of IP is really hard for human being to configure, we use a IPRange to express a continuous IP range, which contains a start IP and a end IP. And we use another function to translate a IPRange into a list of CIDR format IP.

Because every rule is a deny or allow list of maps from a continuous IP range to a destination continuous IP range, so only use IPRange is not enough. So we implement IPSet.



```
public class IPSet {

    private IPRange srcIPRange;
    private IPRange dstIPRange;

    public IPSet(IPRange src, IPRange dst){
        this.srcIPRange = src;
        this.dstIPRange = dst;
    }

    public IPRange getSrcIPRange() { return srcIPRange; }

    public IPRange getDstIPRange() { return dstIPRange; }

    public static List<IPSet> cutOff(IPSet set0, IPSet set1){
        List<IPSet> ipSets = new ArrayList<>();

        Map<String, List<IPRange>> srcMap = IPRange.cutOff(set0.getSrcIPRange(), set1.getSrcIPRange());
        List<IPRange> srcCutoffIPRanges = srcMap.get("cutoff");
        List<IPRange> srcRemainIPRanges = srcMap.get("remain");

        Map<String, List<IPRange>> dstMap = IPRange.cutOff(set0.getDstIPRange(), set1.getDstIPRange());
//        List<IPRange> dstCutoffIPRanges = dstMap.get("cutoff");
        List<IPRange> dstRemainIPRanges = dstMap.get("remain");

        for(IPRange range0 : srcRemainIPRanges)
            ipSets.add(new IPSet(range0, set0.getDstIPRange()));

        for(IPRange range0 : srcCutoffIPRanges)
            for(IPRange range1 : dstRemainIPRanges)
                ipSets.add(new IPSet(range0, range1));

        return ipSets;
    }
```

Fig. 15. Part of Source code of IPSet

As the fig. 15, IPSet is bundle with a source IPRange and a destination IPRange, it could be describe as a rectangle in 2-dimension plane. When a IPSet have been Cutoff by another IPSet, the result should be a list of IPSet, because depend on specific condition, the counts of IPSet in result maybe 4.

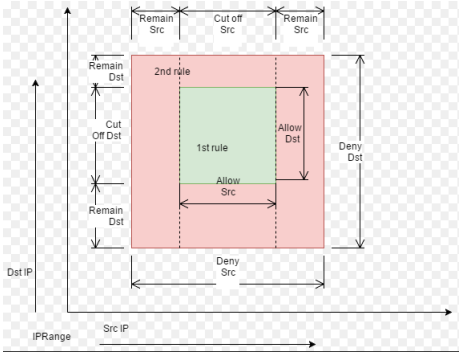


Fig. 16. Description of IP space and Rule cutoff

fig. 16 is an example of one IPSet is Cutoff by another IPSet. Because we define every rule inputted by user have just one IPSet, but what we actually insert into the flow table pool is depended on the result of Cutoff IPSet by the existed IPSet List.

In fig. 16 on the previous page, the green area is the 1st rule have inserted into the flow table pool that allow traffic, and the bigger rectangle is the 2nd rule we want to insert that deny traffic. But only the red area is the answer of the IPSet Cutoff result, and only this part we will insert into the flow table pool of ONOS system.

IV. Demonstration

1. Build our firewall application, ran following commands will download our firewall source code from Github and compile them to generate the oar file which is the compiled ONOS application and can be load and run in ONOS WebUI, CLI or using ONOS Application REST API.

```
git clone git@github.com:siriulx/onos-acl-
    firewall.git
cd onos-acl-firewall
mvn compile
mvn install
```

2. Load the oar file and enable the firewall in WebUI (See fig. 17.):

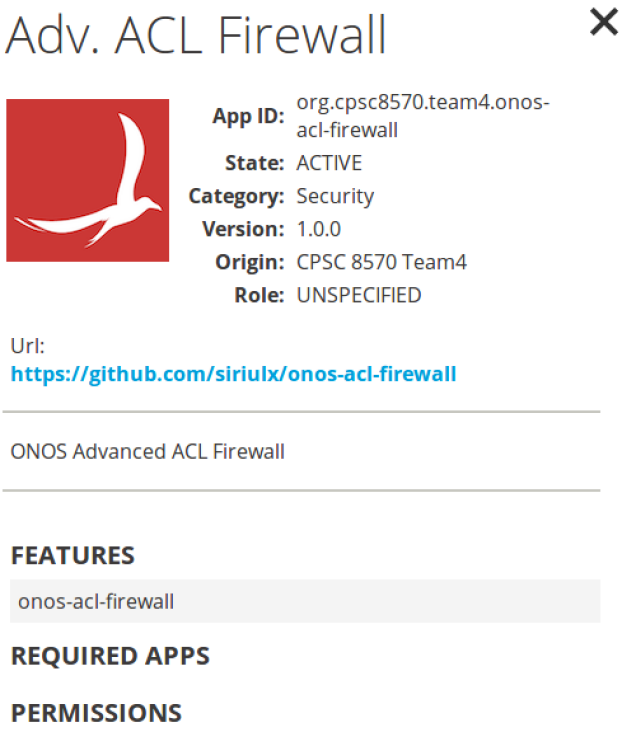


Fig. 17. Load oar file into ONOS

3. The firewall default setting is to allow all the traffic, so if we ran pingall on mininet CLI, all the hosts should be able to access any others. (See fig. 18.)

4. We can use our firewall restful GET method to check firewall rules by executing:

```
curl -sSL --user karaf:karaf -X GET http
    ://192.168.123.1:8181/onos/onos-acl-
    firewall/sample
```

We can see form fig. 19 that there is no rule in the firewall now.

```
mininet-onos> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Fig. 18. Pingall Test After Enable Firewall

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X GET http://192.168.123.1:8181/onos/onos-acl-firewall/sample
{}vm@vm-mininet:~/onos-acl-firewall$
```

Fig. 19. REST API Get Method Result

5. Then we added an ALLOW rule to allow the traffic from h2,h3 to h2,h3 by submitting an POST restful request to our firewall address.

```
curl -sSL --user karaf:karaf -X POST -H '
    Content-Type:application/json' http
    ://192.168.123.1:8181/onos/onos-acl-
    firewall/sample -d '{"action": "ALLOW", "
    srcIpStart":"10.0.0.2", "srcIpEnd
    ":"10.0.0.3", "dstIpStart":"10.0.0.2", "
    dstIpEnd":"10.0.0.3"}'
```

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X POST -H 'Content-Type:application/json' http://192.168.123.1:
8181/onos/onos-acl-firewall/sample -d '{"action": "ALLOW", "srcI
pStart":"10.0.0.2", "srcIpEnd":"10.0.0.3", "dstIpStart":"10.0.0.
2", "dstIpEnd":"10.0.0.3"}'
{"rule":"id:1048576,ipProto:,dstPort:0,action:ALLOW,ipSets:{srcI
PRange:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.2-10.0.0.3},},"}vm
@vm-mininet:~/onos-acl-firewall$
```

Fig. 20. REST API Post Method Result

We can see from fig. 20 the new rule was add into the firewall and id is 1048576. The pingall test result shows on fig. 21.

```
mininet-onos> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet-onos>
```

Fig. 21. Pingall Test After add new rules

6. Next we added a DENY rule to deny the traffic from h1-h4 to h1-h4.

```
curl -sSL --user karaf:karaf -X POST -H '
    Content-Type:application/json' http
    ://192.168.123.1:8181/onos/onos-acl-
    firewall/sample -d '{"action": "DENY", "
    srcIpStart":"10.0.0.1", "srcIpEnd
    ":"10.0.0.4", "dstIpStart":"10.0.0.1", "
    dstIpEnd":"10.0.0.4"}'
```

We can see that the new rule id is 1048577 and the IP range was dynamically separated. The result shows on fig. 22 on the next page. We also use Get Method to check if there is two rules in our firewall rules table, and the result shows on fig. 23 on the following page

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X POST -H 'Content-Type:application/json' http://192.168.123.1:
8181/onos/onos-acl-firewall/sample -d '{"action": "DENY", "srcIp
Start":"10.0.0.1", "srcIpEnd":"10.0.0.4", "dstIpStart":"10.0.0.1
", "dstIpEnd":"10.0.0.4"}'
{"rule":"id:1048577,ipProto:,dstPort:0,action:DENY,ipSets:{srcIP
Range:{10.0.0.1-10.0.0.1},dstIPRange:{10.0.0.1-10.0.0.4},srcIPRa
nge:{10.0.0.4-10.0.0.4},dstIPRange:{10.0.0.1-10.0.0.4},srcIPRang
e:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.1-10.0.0.1},srcIPRange:
{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.4-10.0.0.4},},"}vm@vm-min
inet:~/onos-acl-firewall$
```

Fig. 22. REST API Post Method Result

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X GET http://192.168.123.1:8181/onos/onos-acl-firewall/sample
{"1048576":"id:1048576,ipProto:,dstPort:0,action:ALLOW,ipSets:{s
rcIPRange:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.2-10.0.0.3},},"
,"1048577":"id:1048577,ipProto:,dstPort:0,action:DENY,ipSets:{sr
cIPRange:{10.0.0.1-10.0.0.1},dstIPRange:{10.0.0.1-10.0.0.4},srcI
PRange:{10.0.0.4-10.0.0.4},dstIPRange:{10.0.0.1-10.0.0.4},srcIPR
ange:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.1-10.0.0.1},srcIPRan
ge:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.4-10.0.0.4},},"}vm@vm-
mininet:~/onos-acl-firewall$
```

Fig. 23. REST API Get Method Result

Pingall test (See fig. 24). We can see the traffic only from h2-h3 to h2-h3 was allowed.

7. Next we tried to delete last rule (id: 1048577). Result shows on fig. 25.

```
curl -sSL --user karaf:karaf -X DELETE -H '
    Content-Type:application/json' http
    ://192.168.123.1:8181/onos/onos-acl-
    firewall/sample -d '{"ruleId": "1048577"}'
```

We used GET method to fetch the rules list, and the result shows on fig. 26

We can see now there was only one rule left. Form pingall test (See fig. 27), we can see all the traffic was allowed.

V. Future Work

1) The auto IP area cutoff function still can be optimized. For now the time cost of this function is O(n*n), and because the IP is a continuous number, we can arrange it by order, with this way, except the cost of arrange, the time cost of cutoff function can be reduced to O(n)
2) For now the IP cutoff function just support IPv4, we can implement it to support IPv6 in future.
3) We could do another study in Header Space Analysis Library, because if we use the HSA library, we can use the HSA library to implement the indirect detection.
4) About study OpenVSwitch, because the stateful monitor need some modify of how to deal with package on specific target, to implement this we need to use OpenVSwitch to modify the driver of SDN router.

VI. Contribution of Group Members

```
mininet-onos> pingall 1
*** Ping: testing ping reachability
h1 -> X X X
h2 -> X h3 X
h3 -> X h2 X
h4 -> X X X
*** Results: 83% dropped (2/12 received)
mininet-onos>
```

Fig. 24. Pingall Test After add new rules

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X DELETE -H 'Content-Type:application/json' http://192.168.123.
1:8181/onos/onos-acl-firewall/sample -d '{"ruleId": "1048577"}'
vm@vm-mininet:~/onos-acl-firewall$
```

Fig. 25. REST API Post Method Result

```
vm@vm-mininet:~/onos-acl-firewall$ curl -sSL --user karaf:karaf
-X GET http://192.168.123.1:8181/onos/onos-acl-firewall/sample
{"1048576":"id:1048576,ipProto:,dstPort:0,action:ALLOW,ipSets:{s
rcIPRange:{10.0.0.2-10.0.0.3},dstIPRange:{10.0.0.2-10.0.0.3},},"
}vm@vm-mininet:~/onos-acl-firewall$
```

Fig. 26. REST API Get Method Result

```
mininet-onos> pingall 1
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Fig. 27. Pingall Test After remove last rule

TABLE II
CONTRIBUTION OF GROUP MEMBERS

Yang Cao	Preliminary Design, Proposal, Experiment Environment Setup, Basic ACL supporting Advanced Feature Design & Implement, Final Report
Lexuan Sun	Preliminary Design, Proposal, Experiment Environment Setup, Basic/Advanced ACL supporting, Advanced Feature Design & Implement, Final Report
Chaoran Huang	Preliminary Design, Proposal, Basic/Advacned ACL supporting,Advanced Feature Design & Implement, Testing, Final Report
Chunpeng Shao	Preliminary Design, Proposal, Basic ACL supporting, Advanced Feature Design & Implement, Final Report

REFERENCES

[1] *Openflow switch specification, v1.5.0*, Open Networking Foundation, 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[2] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A survey of security in software defined networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, 2016.

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

[4] 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*, ACM, 2014, pp. 97–102.

[5] *Onos*. [Online]. Available: https://en.wikipedia.org/wiki/ONOS.

[6] I. Cisco, "Cisco ios security configuration guide, release 12.2, access control lists: Overview and guidelines," Tech. Rep., 2014. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/ios/12_2/security/configuration/guide/fsecur_c/scfacls.html.