THE FINAL REPORT OF BUILDING AN SDN FIREWALL APPLICATION IN OPENDAYLIGHT

TEAM NUMBER:	1
TEAM MEMBERS:	Jinrui Wang, Yao Yao, Tian Lian, Ying Zhang
INSTRUCTOR:	Dr. Hongxin Hu
DATE OF SUBMISSION:	01 May 2017

CONTENT

ABSTRACT

For this project, we have accomplished a web-based SDN firewall application in OpenDaylight. We implemented an interface based on web, so that users can use it to configure the firewall easily. In this report we are going to introduce the design of our system, the implementation of the application, the measurement of the firewall. In addition, we also talked about the advantages of our application and the future work in our application.

keywords: OpenDaylight; SDN; firewall; SDN firewall;

INTRODUCTION

In this section, we give some important concepts which we will use in the next section. Thoes concepts include SDN, Firewall, OpenDaylight, OpenFlow, and ACL.

1. SDN

Software Defined Network (SDN) [2] is a new network innovation architecture, it is also an implementation of network virtualization. Its core technology, OpenFlow, separates network control plane from data plane, achieving the flexibility of network traffic control, which makes the network as a pipeline become more intelligent.

The traditional IT architecture in the network, according to the business needs of the deployment of the line, if the business needs to be changed, modify the configuration of the corresponding network equipment is very complicated. In the rapidly changing Internet/mobile Internet business environment, high stability and high performance of the network are not enough to meet the business needs, flexibility and agility are more critical. What SDN does is to separate the control of the network device, managed by the centralized controller, without relying on the underlying network equipment, shielding the differences from the underlying network equipment. And the control is completely open, the user can customize any network routing and transmission rules to achieve the strategy, which is more flexible and intelligent.

2. FIREWALL

In the network, the firewall is a way to separate the intranet and public access network, it is actually an isolation technology. A firewall is an access control scale that is executed at the time of two network communications, which allows you to "agree" people and data into your network while at the same time shutting out your "disagree" people and data, to prevent the network hackers to access your network. In other words, if you do not pass the firewall, the company's internal people can not access Internet, Internet people can not communicate with people inside the company.

It has some basic features: All network traffic between the internal network and the external network must pass through the firewall; Only data flows that meet the security policy can be passed through the firewall; The firewall itself should have a very strong anti-attack immunity; Application layer firewall should have more detailed protection; Database firewall should have blocking ability for database malicious attacks. Firewall has some advantages: It can can enhance security policy; It can effectively record the activities on the Internet; Firewall restrictions expose user points. Firewalls can be used to separate a network from another network segment. In this way, it is possible to prevent the problem that affects a network segment from being transmitted through the entire network; The firewall is a security policy checkpoint. All access to information must pass through the firewall, the firewall will become a security checkpoint, so that suspicious access was denied at the door.

3. OPENDAYLIGHT

OpenDaylight [3] is the largest open source of SDN controller, which is helping to lead this transformation. By working with a common SDN platform, the OpenDaylight community which contains solution providers, personal developers, and users who work together, providing interoperable programmable networks for service providers, businesses, universities and organizations across the globe internet. ODL has a modular, pluggable and flexible control platform as the core, the control platform is based on Java development, theoretically, can run on any platform that supports Java. ODL controller using OSGI framework, SGI framework is a dynamic model system for Java, which implements an elegant, complete and dynamic component model, the application without rebooting can be remotely installed, boot, upgrade and uninstall, through OSGI bundles can be flexible to load code and function, to achieve functional isolation, which solves the scalability problem of function module and facilitates the loading and cooperation of functional modules.

The structure of opendaylight controller is shown in Figure 1.

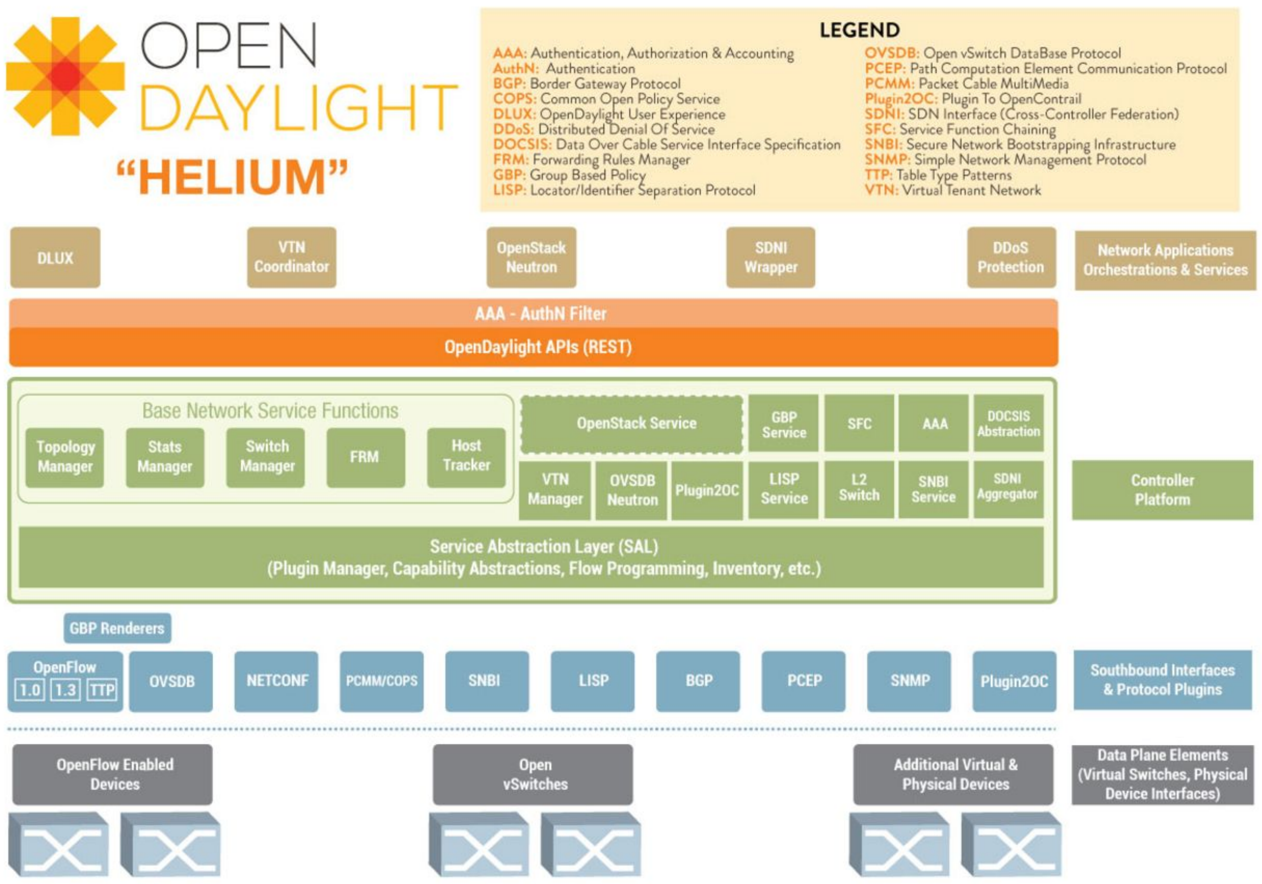

OPEN DAYLIGHT
"HELIUM"
LEGEND
AAA: Authentication, Authorization & Accounting
AuthN: Authentication
BGP: Border Gateway Protocol
COPS: Common Open Policy Service
DLUX: OpenDaylight User Experience
DDoS: Distributed Denial Of Service
DOCSIS: Data Over Cable Service Interface Specification
FRM: Forwarding Rules Manager
GBP: Group Based Policy
LISP: Locator/Identifier Separation Protocol
OVSDB: Open vSwitch DataBase Protocol
PCEP: Path Computation Element Communication Protocol
PCMM: Packet Cable MultiMedia
Plugin2OC: Plugin To OpenContrail
SDNI: SDN Interface (Cross-Controller Federation)
SFC: Service Function Chaining
SNBI: Secure Network Bootstrapping Infrastructure
SNMP: Simple Network Management Protocol
TTP: Table Type Patterns
VTN: Virtual Tenant Network
DLUX
VTN Coordinator
OpenStack Neutron
SDNI Wrapper
DDoS Protection
Network Applications Orchestrations & Services
AAA - AuthN Filter
OpenDaylight APIs (REST)
Base Network Service Functions
Topology Manager
Stats Manager
Switch Manager
FRM
Host Tracker
OpenStack Service
GBP Service
SFC
AAA
DOCSIS Abstraction
VTN Manager
OVSDB Neutron
Plugin2OC
LISP Service
L2 Switch
SNBI Service
SDNI Aggregator
Controller Platform
Service Abstraction Layer (SAL)
(Plugin Manager, Capability Abstractions, Flow Programming, Inventory, etc.)
GBP Renderers
OpenFlow 1.0 1.3 TTP
OVSDB
NETCONF
PCMM/COPS
SNBI
LISP
BGP
PCEP
SNMP
Plugin2OC
Southbound Interfaces & Protocol Plugins
OpenFlow Enabled Devices
Open vSwitches
Additional Virtual & Physical Devices
Data Plane Elements (Virtual Switches, Physical Device Interfaces)

Figure 1 Structure of Opendaylight

4. OPENFLOW

OpenFlow is a protocol that uses an API process to configure a network switch. The network device maintains a FlowTable and only forwards it according to FlowTable. The generation, maintenance, delivery of FlowTable itself are achieved entirely by the external controller. The OpenFlow network controller can determine the path of the network packet in the network switch. The controller is very different from the switch. This separation allows for more complex traffic management than the use of access control lists (ACLs) and the feasibility of routing protocols. The Network structure based on OpenFlow protocol is shown in Figure 2.

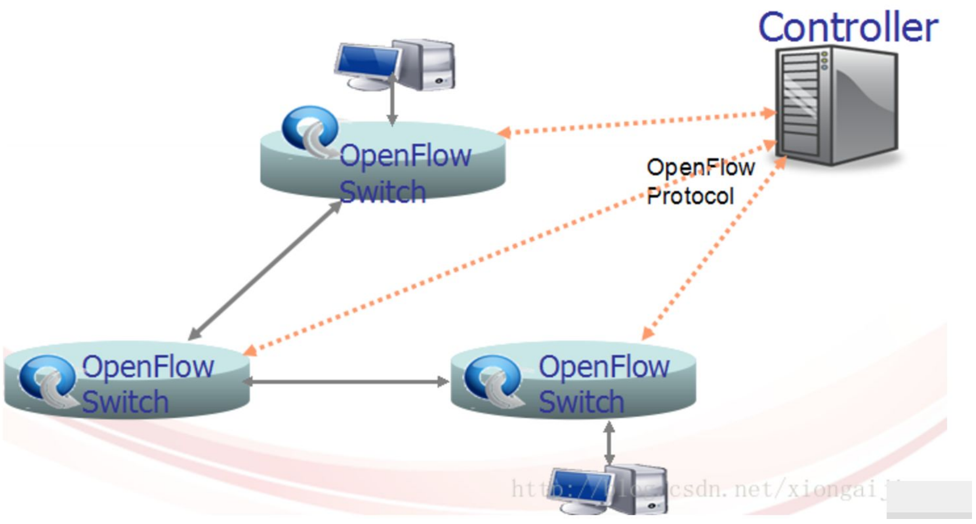


Figure 2 Network structure based on OpenFlow protocol

5. ACL

Access Control List (ACL) is a list of commands for the router and switch interfaces to control incoming and outgoing packets from the port. ACLs are applicable to all routing protocols such as IP, IPX, and AppleTalk. After configuring the ACL, you can restrict network traffic, allow access to specific devices, specify forwarding specific port packets, etc.. For example, you can configure the ACL to prohibit the LAN device to access the external public network, or can only use the FTP service. The ACL can be configured on the router or on the service software with ACL function. ACL is an important technology to ensure the security of the system of the Internet of Things. Based on the security of the hardware layer of the device, it is necessary to use the programmable method to specify the access rules to prevent the illegal device from destroying the system and getting system data by controlling the access control between the devices at the software level.

DESIGN

In this section, we are going to talk about the design of our application SDN Firewall, which includes the function design, architecture design, and the whole operation design.

1. Function Design

So far, there are three functions included in our application, which consist authorization, ACL rules setting, and ACL rules removing.

The foundational function of our application is authorization. In this function, users should give the username, password, Ip address, and port of the remote SDN controller. Our application will check if the SDN controller is existed and the username with the password is corrected.

In our application, the main function of the firewall is to set ACL rules, so that the openflow switches can block or unblock some specific flows. The rules that can be configured are:

- Block / Unblock Ip Address
- Block / Unblock Mac Address
- Block / Unblock Protocol
- Block / Unblock Vlan
- Block / Ublock the mixed rules

Users can block or unblock specific Ip address or a range of Ip address, meanwhile, users can choose which kind of Ip address to block or unblock, source Ip address, destination Ip address, or mixed Ip address. Similarly, Users can block or unblock source mac address, destination mac address, or mixed mac address. For the protocol blocking or unblocking, users can choose TCP, UDP, IGMP, or ICMP to block or unblock. Users also can block or unblock flows based on the vlan ID. In addition, Users can set mixed rule which consists all the above rules, for example, they block a flow which has the specific source Ip address and specific protocol. In this way, only the flow matched all the rules, it can be blocked.

Of cause, Users can remove the rules which setted by themselves before. In our application, we will show all the rules that the users configured before and also provide an interface for users to delete the rules.

Figure 3 presents the functions of our application:

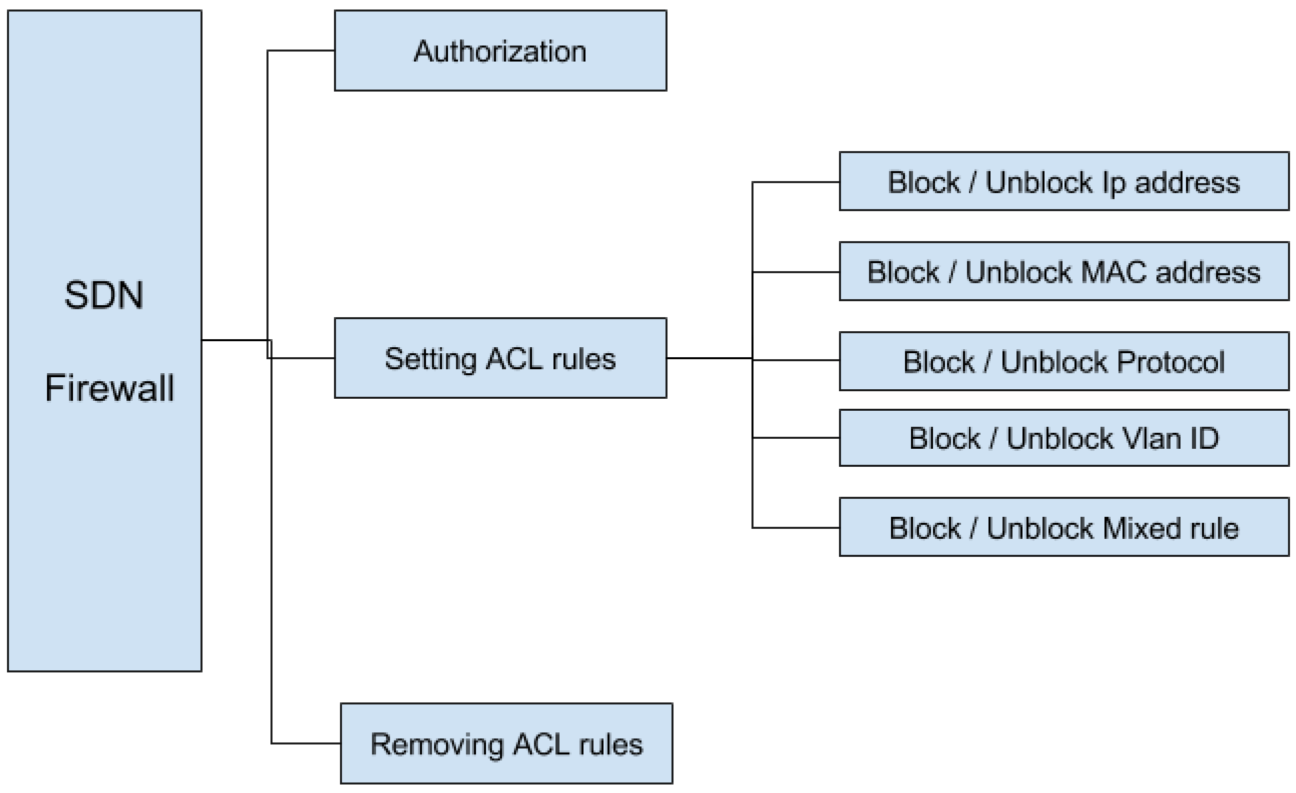


Figure 3. Function Design

2. Architecture Design

Our application is a web-based SDN firewall, which means we have a client end and a server end. Users access our client through the web browse and configure the ACL rules. Our server end receive the requests from client end and handle the requests. Figure 4 is the overview of our architecture:

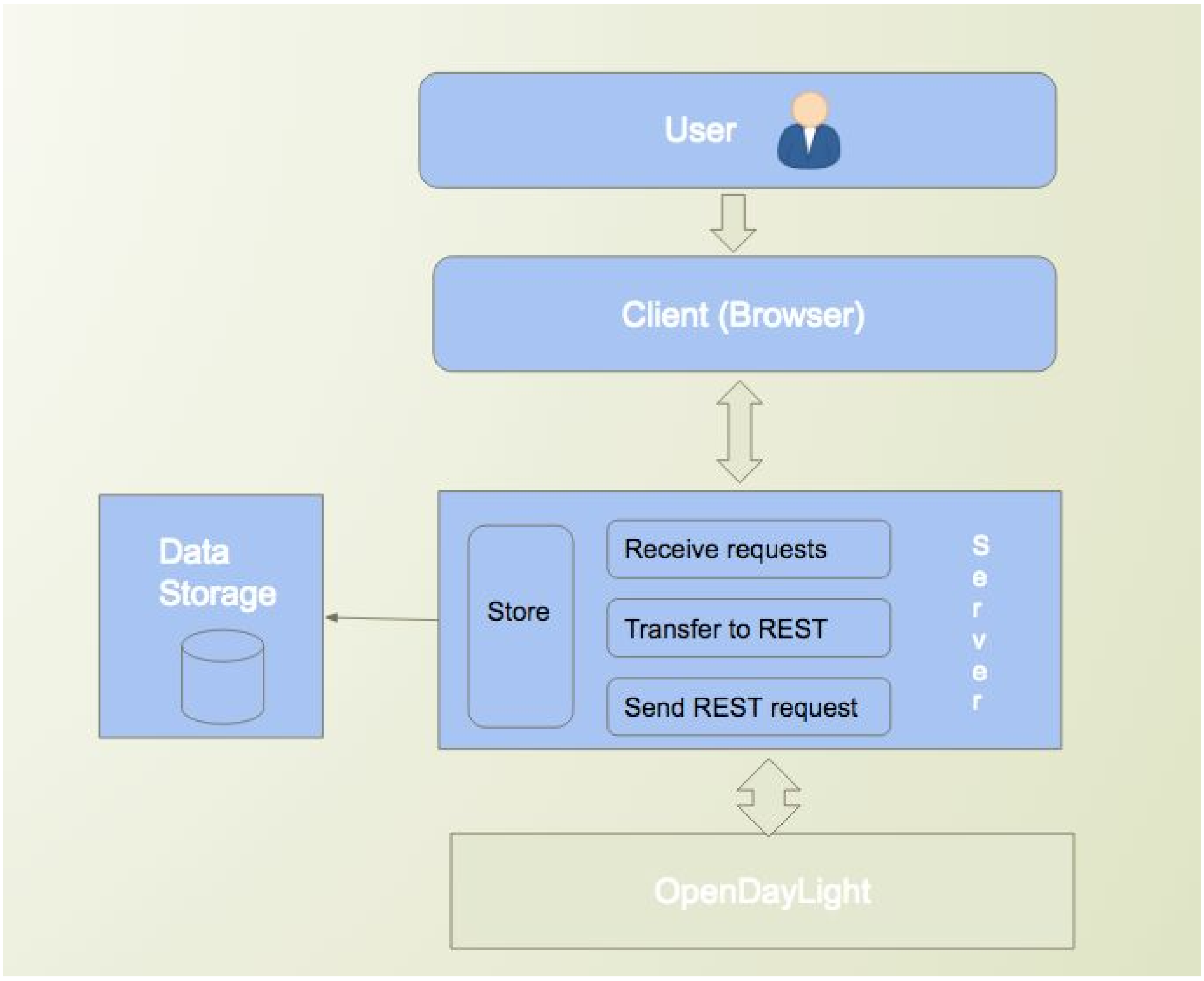


Figure 4. The Overview of Firewall Architecture

Our client part provide a user interface to receive the configurations from users and then send related requests to the server end. In function of the server end include receiving requests, transferring related requests to REST API, storing configurations and sending REST request.

3. Whole Operation Design

The following shows the whole operation design of our application. From figure 5, you can see that our firewall is operated on the Network Application layer of OpenDaylight. The server end of our application connect with the OpenDaylight API by REST API. Meanwhile, we choose OpenFlow protocol to connect OpenFlow switches, which simulated by mininet, and OpenDaylight.

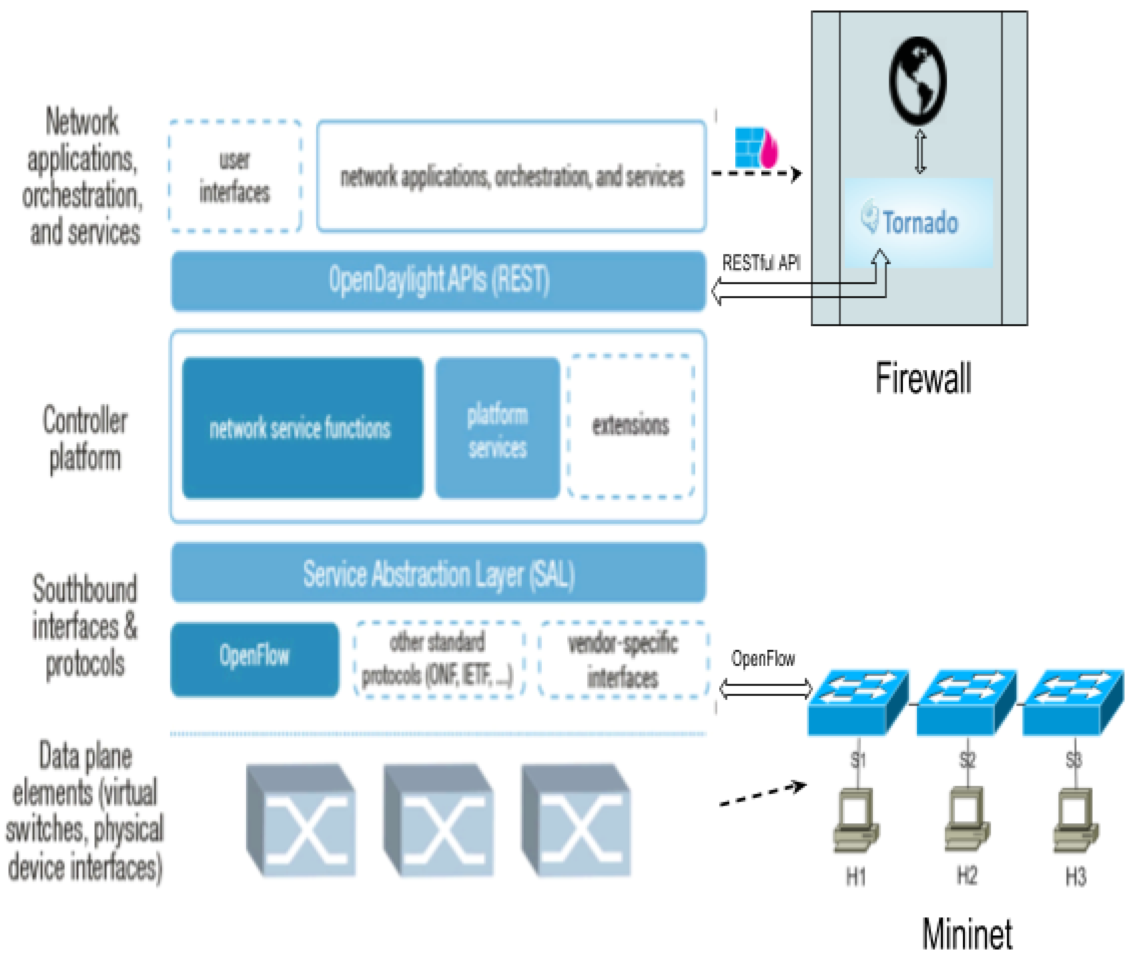

Network applications, orchestration, and services
user interfaces
network applications, orchestration, and services
OpenDaylight APIs (REST)
RESTful API
Tornado
Firewall
Controller platform
network service functions
platform services
extensions
Southbound interfaces & protocols
Service Abstraction Layer (SAL)
OpenFlow
other standard protocols (ONF, IETF, ...)
vendor-specific interfaces
OpenFlow
S1
S2
S3
Data plane elements (virtual switches, physical device interfaces)
H1
H2
H3
Mininet

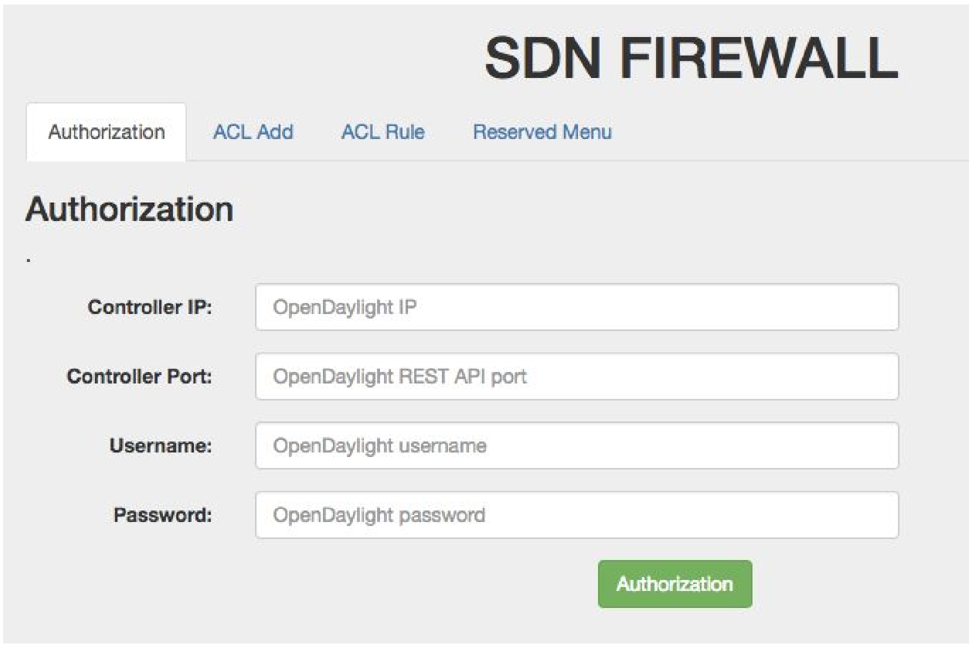
Figure 5. The Overview of Whole Operation Framework

IMPLEMENTATION

1. Front-end Implementation

Our front-end is a series of web pages, we use HTML, CSS, JavaScript as our development programming language. In the front-end, we provide a user interface with three main functions: authorization, setting ACL rule, and removing ACL rules.

The following shows the authorization interface. Users should provide controller IP, controller port, username, and password.



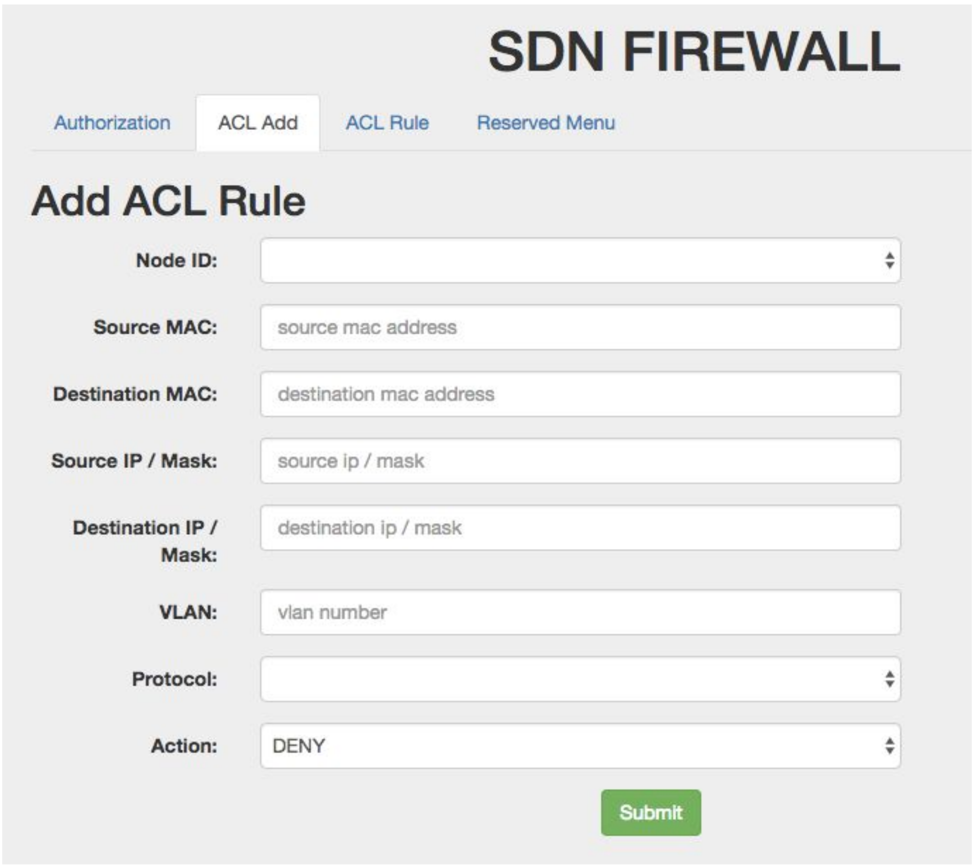
SDN FIREWALL

Authorization

Controller IP:	OpenDaylight IP
Controller Port:	OpenDaylight REST API port
Username:	OpenDaylight username
Password:	OpenDaylight password

Figure 6. Authorization function

Figure 7 is the ACL rules setting interface, which users can configure specific rule in this page.



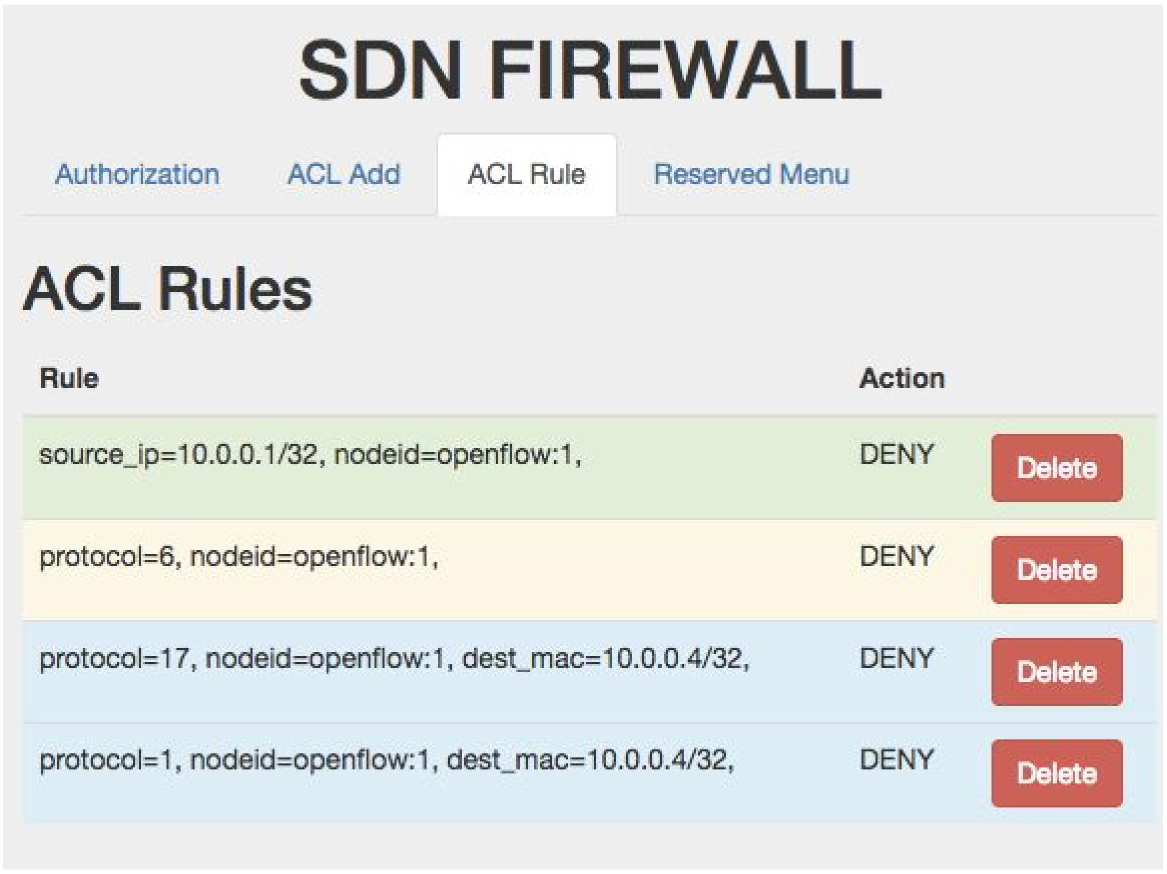
SDN FIREWALL

Add ACL Rule

Node ID:	
Source MAC:	source mac address
Destination MAC:	destination mac address
Source IP / Mask:	source ip / mask
Destination IP / Mask:	destination ip / mask
VLAN:	vlan number
Protocol:	
Action:	DENY

Figure 7. Setting ACL Rule Function

Figure 8 is the ACL rules removing interface, which user can browse all the ACL rules they configured and delete any of them.



ACL Rules

Rule	Action	
source_ip=10.0.0.1/32, nodeid=openflow:1,	DENY	Delete
protocol=6, nodeid=openflow:1,	DENY	Delete
protocol=17, nodeid=openflow:1, dest_mac=10.0.0.4/32,	DENY	Delete
protocol=1, nodeid=openflow:1, dest_mac=10.0.0.4/32,	DENY	Delete

Figure 8. Removing ACL Rule Function

2. Back-end Implementation

For the back-end, we use Tornado as our web server framework and Python2.7 as our programming language. The main function of server end is to transfer the requests, which received from the client, to REST API and send to the OpenDaylight controller.

Tornado is an asynchronous networking library and Python web framework, we can easily use it to launch a web server. In the server end, we use many handlers to handle the related requests, for example, *SetACLRuleHandler* to handle the setting ACL rule requests, *GetACLRuleHandler* to handle the getting ACL rule requests, etc.

Corresponding with the front end functions, we have the related functions to handle them. I will give the detail code of those functions next.

Authorization function

```
def authentication(self, body):
       try:
           controller_ip = body['controller_ip']
           controller_port = body['controller_port']
           base_auth = body['authentication']
       except:
          self.write("Authentication failed!!")

       topo_api = '/restconf/operational/network-topology:network-topology'
       topo_url = 'http://' + str(controller_ip) + ':' + \
               str(controller_port) + topo_api

       headers = {'authorization': base_auth}
       input_data = {}
       node_list = []
       ret = requests.get(topo_url, headers=headers)
       if ret:
           network_topology = ret.json()['network-topology']
           try:
               nodes = network_topology['topology'][0]['node']
               for node in nodes:
                   if 'host' not in node['node-id']:
                       node_list.append(node['node-id'])
           except:
              pass

       if ret.status_code == 200:
           self.set_secure_cookie("controller_ip", controller_ip)
           self.set_secure_cookie("controller_port", controller_port)
           self.set_secure_cookie("base_auth", base_auth)
           self.write(json.dumps(node_list))
       else:
           self.write_error(500)
```

Setting ACL rule

```
def set_acl_rule(self, body):
       if not self.get_secure_cookie("base_auth"):
           self.write("Please Authentication")
           return

       body_cp = copy.deepcopy(body)
       action = body_cp['action']
       body_cp.pop('action')
       body_cp.pop('post_type')
       rule = {
           "id"      : str(uuid.uuid1()),
           "condition"    : body_cp,
           "action" : action
       }

       rules_list = []
       with open('rules.json') as data_file:
           try:
               rules_list = json.load(data_file)
           except:
                pass

        add_rule = True
        for temp in rules_list:
            if(temp['condition'] == rule['condition']):
                if(temp['action'] != rule['action']):
                    rules_list.append(rule)
                    rules_list.remove(temp)
                    add_rule = False
                else:
                    return

        if add_rule:
            rules_list.append(rule)
        with open('rules.json', 'w') as outfile:
            json.dump(rules_list, outfile)

        controller_ip = self.get_secure_cookie("controller_ip")
        controller_port = self.get_secure_cookie("controller_port")
        base_auth = self.get_secure_cookie("base_auth")

        base_url = 'http://' + str(controller_ip) + ':' \
                + str(controller_port)
        add_acl_api = '/restconf/operations/sal-flow:add-flow'
        url = base_url + add_acl_api
        payload = self.generate_xml(body)
        headers = {
                'authorization': base_auth,
                'content-type': "application/xml"
            }
        ret = requests.post(
                url,
                data=payload,
                headers=headers
            )

        if ret.status_code == 200:
            self.write(" success ")
        else:
            self.write_error(500)
        return
```

Removing ACL rule

```
    def del_acl_rule(self, body):
        if not self.get_secure_cookie("base_auth"):
            self.write("Please Authentication")
            return

        acl_id = body['id']
        rules_list = []
        with open('rules.json') as data_file:
            try:
                rules_list = json.load(data_file)
            except:
                self.write(500)

        body_tmp = {}
        is_exist = False
        for temp in rules_list:
            if(temp['id'] == acl_id):
                body_tmp = temp['condition']
                body_tmp['action'] = str(temp['action'])
                rules_list.remove(temp)
                is_exist = True
                break

        if not is_exist:
            return

        with open('rules.json', 'w') as outfile:
            json.dump(rules_list, outfile)

        controller_ip = self.get_secure_cookie("controller_ip")
        controller_port = self.get_secure_cookie("controller_port")
        base_auth = self.get_secure_cookie("base_auth")

        base_url = 'http://' + str(controller_ip) + ':' \
                + str(controller_port)
        remove_acl_api = '/restconf/operations/sal-flow:remove-flow'
        url = base_url + remove_acl_api

        payload = self.generate_xml(body_tmp)
        #import pdb;pdb.set_trace()
        headers = {
                'authorization': base_auth,
                'content-type': "application/xml"
            }
        ret = requests.post(
                url,
                data=payload,
                headers=headers
            )

        if ret.status_code == 200:
            for item in rules_list:
                rule = ''
                condition = item['condition']
                for one in condition:
                    if(condition[one]):
                        rule += one + '=' + condition[one] + ', '
                item['condition'] = rule

            self.write( json.dumps(rules_list))
        else:
            self.write_error(500)
        return
```

3. The REST API We use

Totally, we use two REST API which provided by OpenDaylight.

Add an ACL rule

http://controller_ip:8181/restconf/operations/sal-flow:add-flow

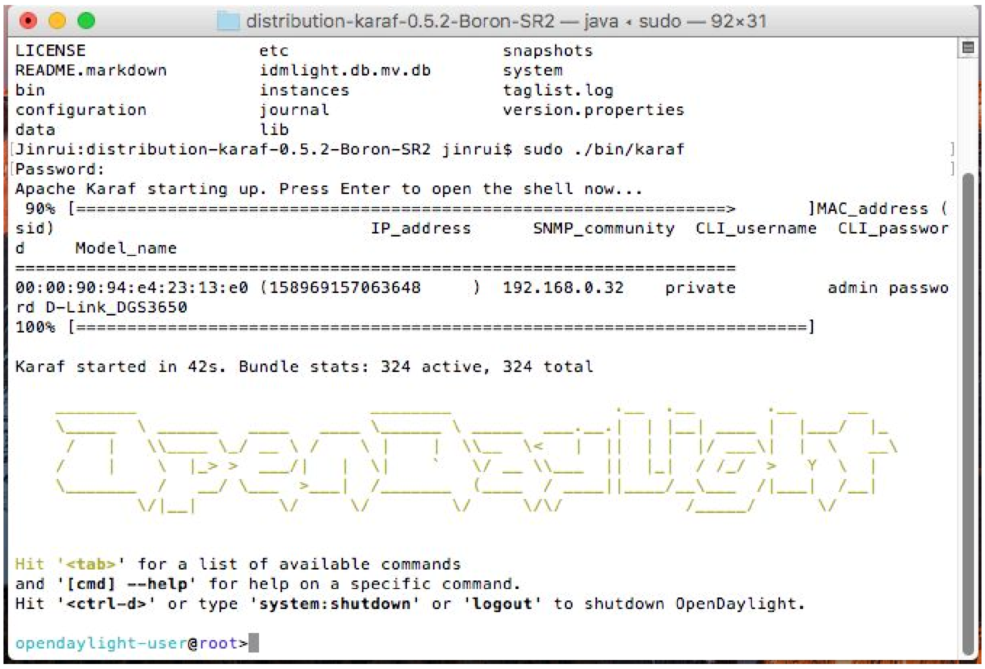
Remove an ACL rule

http://controller_ip:8181/restconf/operations/sal-flow:remove-flow

MEASUREMENT

1. Building Testbed

In this project, we created a virtual machine in Virtualbox using Ubuntu 14.04.2 LTS as our operation system for our experiment. We chose to use OpenDaylight Boron since this version not only provides developers with a lot of network management capabilities but also enhance the flexibility and performance. We used karaf to start OpenDaylight, at the same time we installed related features to configure OpenDaylight which is shown in Figure 9. Since OpenvSwitch 2.3.9 can support existing standard management interfaces and protocols, we selected it as our virtual switch which controlled by OpenDaylight based on OpenFlow 1.3 protocol. Our topology was created with mininet because mininet not only support Openflow, but also provide python API making it easy for us to simulate the network environment. We chose to use mininet 2.2.11 because only Mininet 2.1.0 [6] and above can support OpenFlow 1.3.



```
distribution-karaf-0.5.2-Boron-SR2 — java ◂ sudo — 92×31
LICENSE                 etc                     snapshots
README.markdown         idmlight.db.mv.db       system
bin                     instances               taglist.log
configuration           journal                 version.properties
data                    lib
Jinrui:distribution-karaf-0.5.2-Boron-SR2 jinrui$ sudo ./bin/karaf
Password:
Apache Karaf starting up. Press Enter to open the shell now...
 90% [================================================================>      ]MAC_address (
sid)                              IP_address      SNMP_community  CLI_username  CLI_passwor
d     Model_name
======================================================================
00:00:90:94:e4:23:13:e0 (158969157063648     )  192.168.0.32    private         admin passwo
rd D-Link_DGS3650
100% [========================================================================]

Karaf started in 42s. Bundle stats: 324 active, 324 total

    ________                       ________                .__  .__       .__     __
    \_____  \ ______   ____   ____ \______ \ _____  ___.__.|  | |__| ____ |  |___/  |_
     /   |   \\____ \_/ __ \ /    \ |    |  \\__  \<   |  ||  | |  |/ ___\|  |  \   __\
    /    |    \  |_> >  ___/|   |  \|    `   \/ __ \\___  ||  |_|  / /_/  >   Y  \  |
    \_______  /   __/ \___  >___|  /_______  (____  / ____||____/__\___  /|___|  /__|
            \/|__|        \/     \/        \/     \/\/            /_____/      \/


Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.

opendaylight-user@root>
```

Figure 9. Starting and Configuring OpenDaylight

2. Test Topology

We use mininet to create our topology, the following is the topology and our configuration mininet code.

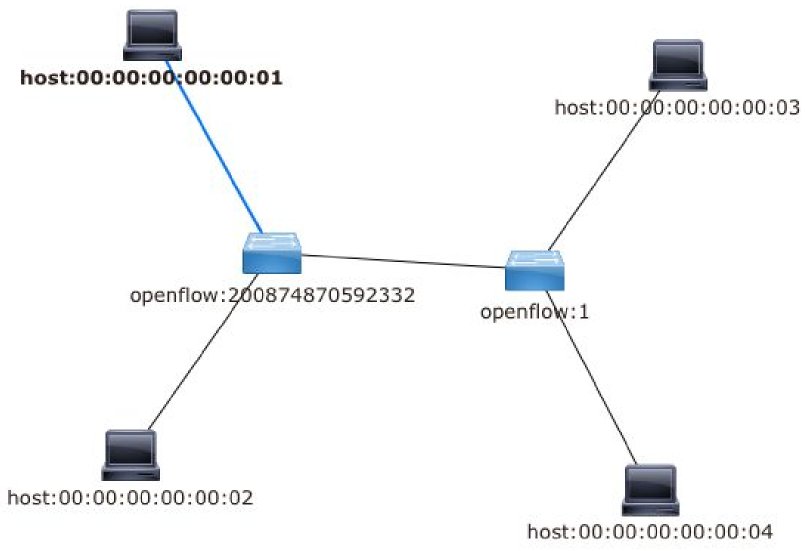


Figure 10. The Topology of Test

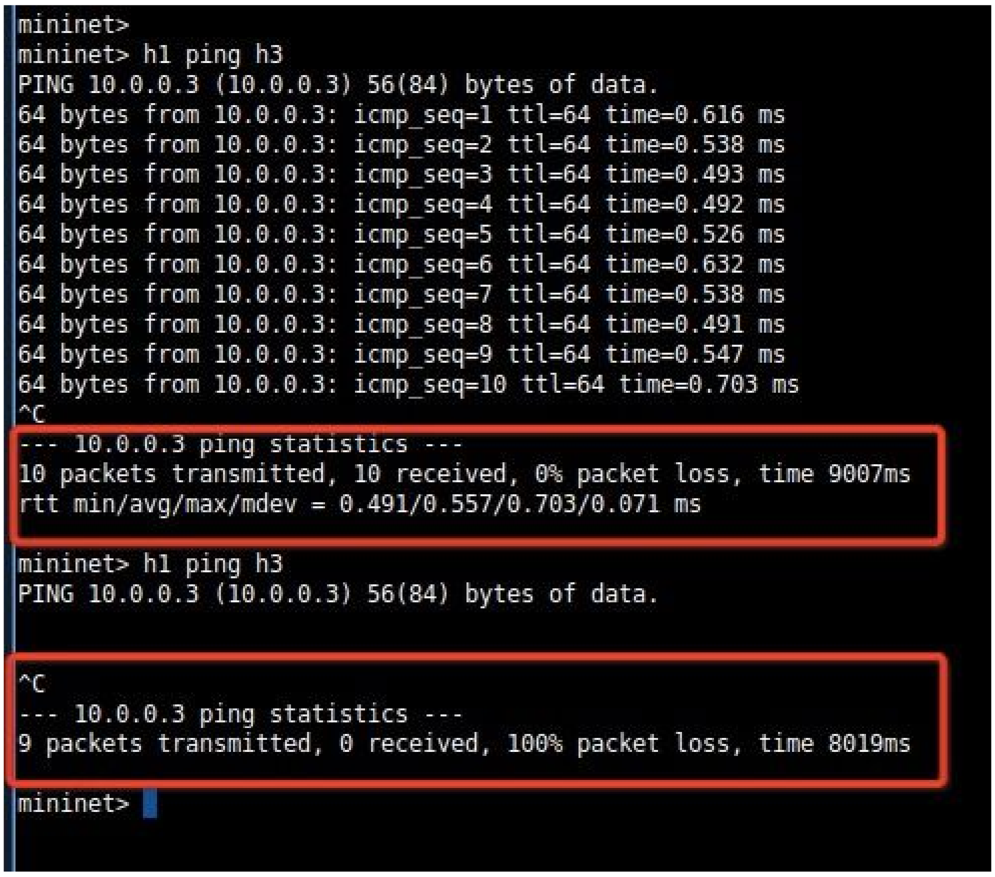
mn_config.py

```
class TestTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
        # Add hosts and switches
        h1 = self.addHost('h1', ip='10.0.0.1', mac='00:00:00:00:00:01')
        h2 = self.addHost('h2', ip='10.0.0.2', mac='00:00:00:00:00:02')
        s0  = self.addSwitch('s0', ip='10.0.0.10', mac='00:00:00:00:00:10')

        h3 = self.addHost('h3', ip='10.0.0.3', mac='00:00:00:00:00:03')
        h4 = self.addHost('h4', ip='10.0.0.4', mac='00:00:00:00:00:04')
        s1  = self.addSwitch('s1', ip='10.0.0.11',mac='00:00:00:00:00:11')

        # Add links
        self.addLink(h1, s0)
        self.addLink(h2, s0)
        self.addLink(h3, s1)
        self.addLink(h4, s1)
        self.addLink(s0, s1)

def basicTest():
    c0 = RemoteController('c0', ip='127.0.0.1', port=6633)
    #net = Mininet(topo=TestTopo(),controller=None)
    net = Mininet(topo=TestTopo(),controller=None, switch=OVSSwitch)
    net.addController(c0)
    net.start()
    net.pingAll()
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    basicTest()
```

3. Test Case

a) Blocking source ip address

We use h1, which has the ip address 10.0.0.1, to ping h3, which has the ip address 10.0.0.3. Before blocking, we can see that the h1 can ping h3 very well. However, when we set the ACL rule to block the source ip address 10.0.0.1, we can see h1 cannot ping h2 any more. The figure 11 shows the result:

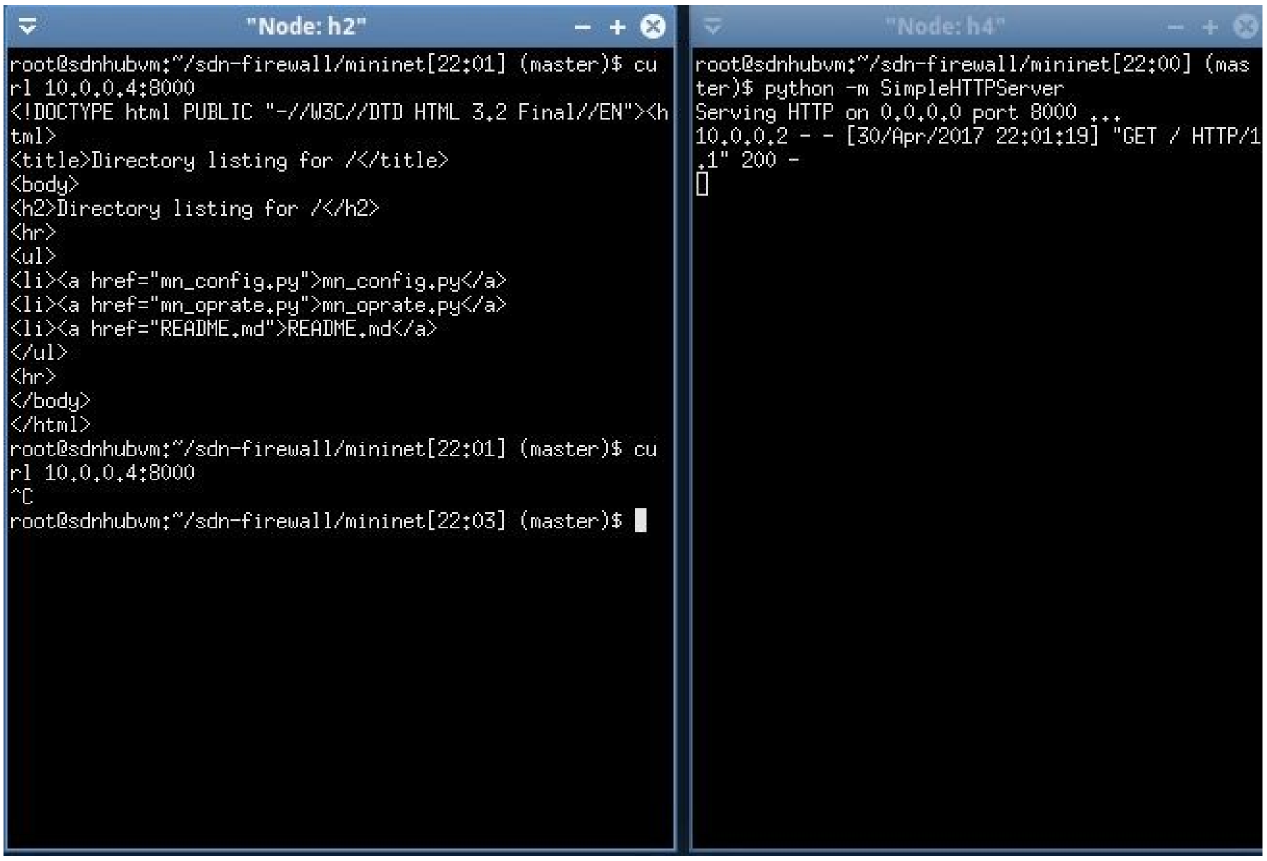


```
mininet>
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.616 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.538 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.493 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.492 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.526 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.632 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.538 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.491 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.547 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.703 ms
^C
--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9007ms
rtt min/avg/max/mdev = 0.491/0.557/0.703/0.071 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.


^C
--- 10.0.0.3 ping statistics ---
9 packets transmitted, 0 received, 100% packet loss, time 8019ms
mininet>
```

Figure 11. The Result of Blocking Source Ip Address.

b) Blocking protocol

In this test case, we test TCP protocol. At first, we launch a simple HTTP server use python in h4, and then we use curl to access the HTTP server in h2. Before blocking, we can easily get the related information from server h4, but after blocking, curl cannot get information. The result can be found in figure 12.



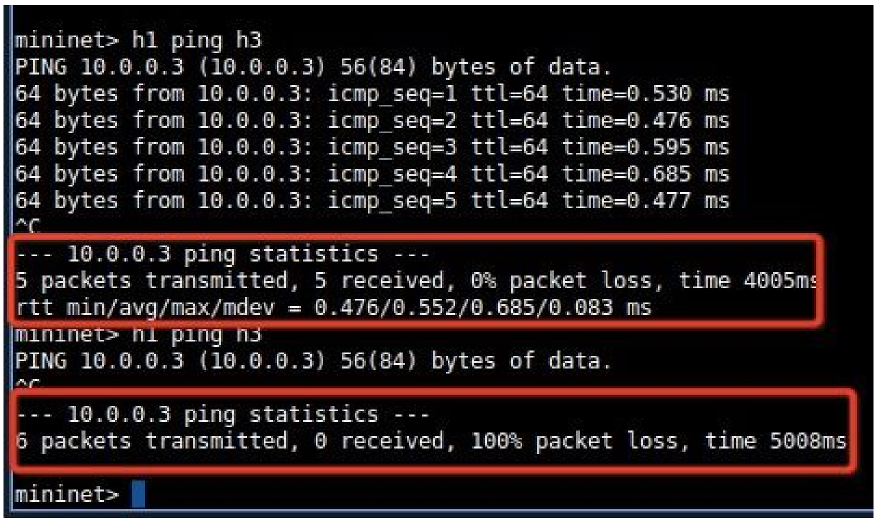
```
root@sdnhubvm:~/sdn-firewall/mininet[22:01] (master)$ cu
rl 10.0.0.4:8000
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><h
tml>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="mn_config.py">mn_config.py</a>
<li><a href="mn_oprate.py">mn_oprate.py</a>
<li><a href="README.md">README.md</a>
</ul>
<hr>
</body>
</html>
root@sdnhubvm:~/sdn-firewall/mininet[22:01] (master)$ cu
rl 10.0.0.4:8000
^C
root@sdnhubvm:~/sdn-firewall/mininet[22:03] (master)$
```

```
root@sdnhubvm:~/sdn-firewall/mininet[22:00] (mas
ter)$ python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
10.0.0.2 - - [30/Apr/2017 22:01:19] "GET / HTTP/1
.1" 200 -
```

Figure 12. Result of Blocking Protocol

c) Blocking mixed rules

In this test, we use h1 to ping h3 again. At first, we blocked source ip address 10.0.0.1 and TCP, we find that h1 can ping h3 with lost packages. Then, we block source ip address 10.0.0.1 and IGMP, we found that h1 cannot ping h3 any more. Figure 13 gives the result.



```
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.530 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.476 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.595 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.685 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.477 ms
^C
--- 10.0.0.3 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 0.476/0.552/0.685/0.083 ms
mininet> h1 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5008ms

mininet>
```

Figure 13. The Result of Block Mixed Rules

FUTURE WORK

For the future work, we are going to add some advanced features, like dynamic network policy updates examining, indirect security violation checking, and stateful monitoring. Meanwhile, we will still working on those basic features, like adding more blocking or unblocking conditions, making our application more reliability, etc.

CONTRIBUTION

In this project, all our memberships are working very hard. At first, we discussed and selected our topics and written our project proposal. Then we researched the paper FLOWGUARD: Building Robust Firewalls for Software-Defined Networks [1] and prepared the paper presentation. At the end, we built our development environment and accomplished the basic features.

CONCLUSION

Finally, we finished our project with accomplishing an SDN firewall in OpenDaylight. Our application has some advantages compared with traditional firewall. a) Usability, Our application is web-based and easy to use, users needn't to remember the complex REST APIs. b) Expandability, our firewall are not only used for OpenDaylight, but also can be used for other SDN controllers after simple change. c) In addition, it's cost efficiency, setup time, maintenance, etc are better than traditional firewall. Also through this project, we learned quite many new knowledge about SDN and Security of SDN.

REFERENCES

[1] H. Hu, W. Han, G.-J. Ahn, Z. Zhao, "FLOWGUARD: Building robust firewalls for software-defined networks", Proc. 3rd Workshop Hot Topics Softw. Defined Netw., pp. 97-102, 2014.

[2] OpenDaylight: The OpenDaylight Platform. https://www.opendaylight.org/

[3] N. McKeown et al., "OpenFlow: Enabling innovation in campus networks", SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69-74, Mar. 2008.

[4] Tornado: Tornado Web Server: http://www.tornadoweb.org/en/stable/

[5] Python: https://www.python.org/

[6] Mininet: An Instant Virtual Network on your Laptop (or other PC): https://mininet.org