

# Building an SDN firewall application in OpenDaylight

Prateek Sreedhar Bharadwaj  
School of Computing  
Clemson University  
Clemson, United States of America  
pbharad@clemson.edu

Siri Uday Shastry  
School of Computing  
Clemson University  
Clemson, United States of America  
sshastr@clemson.edu

Shreyas Prabhakar  
School of Computing  
Clemson University  
Clemson, United States of America  
sprabha@clemson.edu

Vivek Arakali Nagaraja Rao  
School of Computing  
Clemson University  
Clemson, United States of America  
varakal@clemson.edu

**Abstract**— SDN is a tool for solving network management problems easily. New technologies are always faced with security issues and must be addressed in order for them to be used extensively. This report presents documentation on building an SDN firewall application in OpenDaylight(ODL), which is an open source SDN platform. ODL employs a model driven approach and microservices architecture so that any app or function can be bundled into a service that is loaded into the controller. A stateless firewall is developed to check the flow of malicious packets into the software defined network.

**Keywords**—Software Defined Networking (SDN), OpenDayLight(ODL), Mininet, Maven, Karaf, Stateless firewall, Security.

## I. INTRODUCTION

### A. Software Defined Networking (SDN)

It is an approach to using open protocols, such as OpenFlow, to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware. Software-defined networking (SDN) is an umbrella term encompassing several kinds of network technology aimed at making the network as agile and flexible as the virtualized server and storage infrastructure of the modern data centre. In a software-defined network, a network administrator can shape traffic from a centralized control console without having to touch individual switches, and can deliver services to wherever they are needed in the network, without regard to what specific devices a server or other hardware components are connected to. The control plane is separated from the underlying data plane which makes SDN to be dynamic, scalable and to fulfil the needs of the modern computing environments.

### B. OpenDayLight

The OpenDaylight Project is a collaborative open source project hosted by The Linux Foundation. The goal of the

project is to promote software-defined networking (SDN) and network functions virtualization (NFV). The software is written in the Java programming language. SDN is an industry movement for building programmable networks that are flexible and responsive to organizations' and users' needs. OpenDaylight, the largest open source SDN controller helps to lead this transition.

Our project is set out to provide security for an emerging technology namely the Software Defined Networking. Since we are building a firewall application, only packets which are valid (adherent to rules provided) can enter the network and other packets are dropped. These rules can be based on specific source Internet Protocol(IP) addresses, destination IP addresses, number of packets, size of the packets and so on.

The goal of the project is to build a firewall application for Software Defined Networking (SDN) which blocks malicious packets from entering the network and allow only verified packets. A stateless firewall is developed to ensure security for SDN.

## II. MOTIVATION

Security has been of utmost concern since any technology has emerged. With new technologies come new security challenges that must be addressed in order for the technology to be used in industry or academia. Since SDN is an emerging technology, the security issues must be addressed in order for it to be widely used. In that attempt, we are building a firewall application in OpenDaylight (ODL) SDN controller. Some reasons behind using ODL are:

- Open source platform.
- Supports OpenFlow 1.0 and 1.3
- Highly available, modular, scalable, extensible, scalable and multi-protocol controller infrastructure

### III. PRE-REQUISITES

The following are the software/tools and some terms used during the process of building a firewall application for SDN using OpenDaylight:

#### A. OpenDayLight controller

It is a modular open SDN platform for networks of any size and scale. ODL enables network services across a spectrum of hardware in multivendor environments. Our microservices architecture allows users to control applications, protocols and plugins, as well as to provide connections between external consumers and providers. We use the latest version - Boron.

#### B. Mininet

It is mainly used to create the topology. It creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command.

#### C. Maven

It is an open source - Apache build manager for Java projects. It features a (POM) Project Object Model, extensible process plugin framework. Apache Maven 3.5.0 is used in the project.

#### D. YANG

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF), NETCONF remote procedure calls, and NETCONF notifications.

#### E. Karaf

It is a small OSGi based runtime which provides a lightweight container for loading different modules. The different components and apps can be deployed on this. It provides enterprise ready features such as shell console, remote access, dynamic configuration etc.

#### F. Wireshark

It is a free open source packet analyzer used to analyze what's going inside the network. It tries to capture network packets and to display that packet data.

#### G. REST APIs

It is a way of providing interoperability between computer systems. It allows requesting systems to access and manipulate textual representations of Web resources using a uniform and predefined set of stateless operations. XML and JSON formats are used for data exchange.

REST APIs used in ODL are:

- GET – (/restconf/operational/<identifier>) - Returns the value of the data node from the operational datastore. <identifier> points to the concrete data node and its data will be returned.
- PUT – (/restconf/config/<identifier>) – Updates or creates data in config datastore and returns the state about success. <identifier> points to a data node which will be updated or created.

- POST – (/restconf/config/<identifier>) – Creates the data if it does not exist in the config datastore and returns the state about success. <identifier> points to a data node where data must be stored. The root element of data must have the namespace (data are in XML) or module name (data are in JSON).

### IV. ENVIRONMENT SETUP

#### 1) Setup Ubuntu 14 Virtual Machine on VirtualBox

#### 2) Install JDK.

```
sudo apt-get install openjdk-8-jdk
```

#### 3) We need Maven to build ODL. Install the most recent version of Maven

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

#### 4) Other dependencies

```
sudo apt-get install pkg-config gcc make ant g++ git  
libboost-dev libcurl4-openssl-dev libjson0-dev libssl-dev  
openjdk-7-jdk unixodbc-dev xmlstarlet
```

#### 5) Opendaylight installation

Get the karaf distribution file from opendaylight nexus repository -

```
wget  
https://nexus.opendaylight.org/content/repositories/public/org/opendaylight/integration/distribution-karaf/0.5.3-Boron-SR3/distribution-karaf-0.5.3-Boron-SR3.tar.gz
```

Run the following (Fig 1)

```
tar -xvzf distribution-karaf-0.5.3-Boron-SR3.tar.gz  
cd distribution-karaf-0.5.3-Boron-SR3  
bin/karaf
```

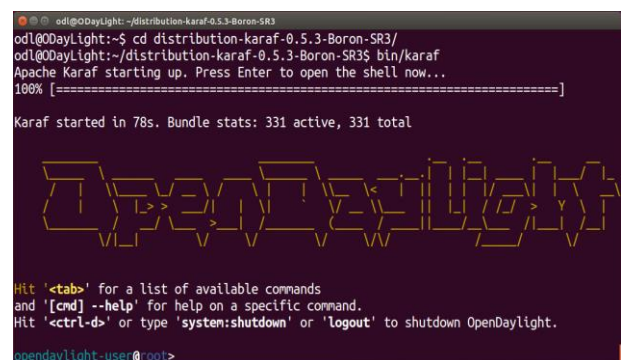


Fig 1: Starting Apache Karaf

#### 6) Install OpenDaylight features: Install the minimum set of features required for OpenDaylight and the OpenDaylight GUI:

```
opendaylight-user@root> feature:install odl-  
restconf odl-l2switch-switch odl-mdsal-apidocs odl-  
dlux-all
```

- odl-restconf: Allows access to RESTCONF API

- odl-l2switch-switch: Provides network functionality similar to an Ethernet switch
- odl-mdsal-apidocs: Allows access to Yang API
- odl-dlux-all: OpenDaylight graphical user interface

To list all available optional features, run the command:

```
opendaylight-user@root> feature:list
```

To list all installed features, run the command:

```
opendaylight-user@root> feature:list --installed
```

#### 7) Installing Mininet

The following are the steps to install mininet natively from source:

#### 8) Get the source code

```
git clone git://github.com/mininet/mininet
cd mininet
git tag # list available versions
git checkout -b 2.2.1 2.2.1
cd ..
```

9) Once you have the source tree, the command to install Mininet is:

```
mininet/util/install.sh -nfv
```

## V. FLOW OF THE PROJECT

### A. Creating a topology using mininet

1) *Interact with Hosts and Switches:* Start a minimal topology and enter the CLI:

```
sudo mn --topo linear,3 --mac --
controller=remote,ip=127.0.0.1,port=6633 --switch
ovs,protocols=OpenFlow13
```

2) *Test connectivity between hosts(Fig 2):* To verify that there can be a ping from host 0 to host 1

```
mininet> pingall
```

```
odl@OpenDayLight:~$ sudo mn --topo linear,3 --mac --controller=remote,ip=127.0.0.1,
port=6633 --switch ovs,protocols=OpenFlow13
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s2) (h3, s3) (s2, s1) (s3, s2)
*** Configuring hosts
h1 h2 h3
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3
h2 -> h1 h3
h3 -> h1 h2
*** Results: 0% dropped (6/6 received)
mininet>
```

Fig 2: Hosts and switches

### B. DLUX web UI

To view the OpenDaylight graphical user interface, open a browser on the host system and enter the URL of the

OpenDaylight User Interface (DLUX UI). It is running on the OpenDaylight VM so the IP address is local host and the port, defined by the application, is 8181:

So the URL is: <http://localhost:8181/index.html>. (Fig 3)

(During sign-in process, the default username and password used is 'admin' in both fields)

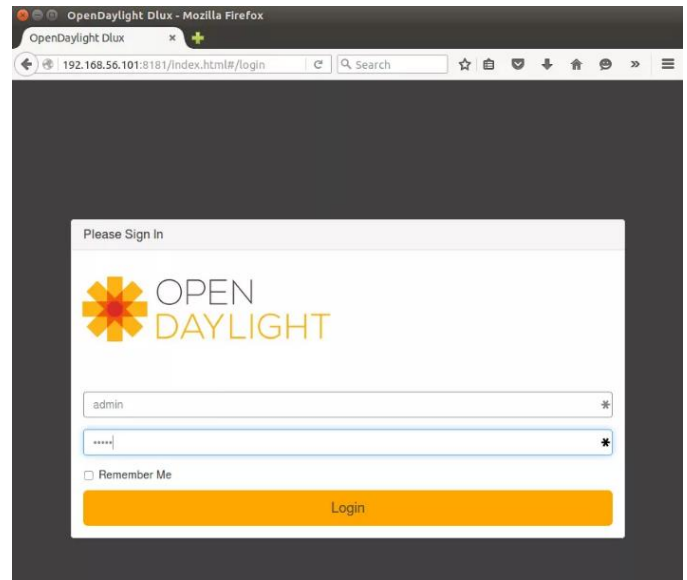


Fig 3: OpenDayLight sign-in page

### C. View the topology created using mininet (Fig 4)

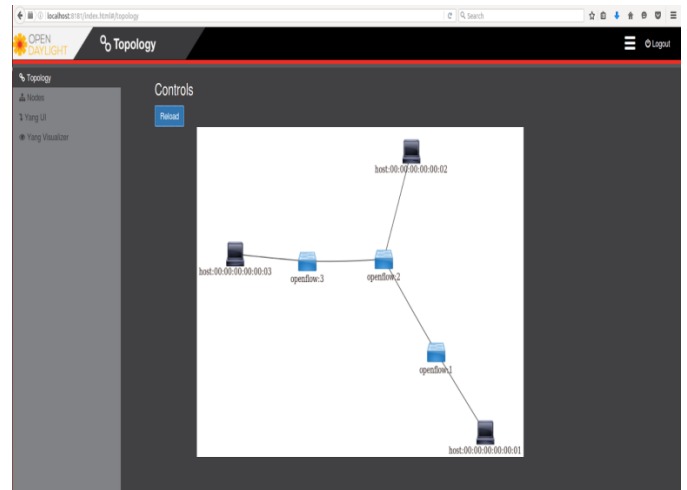


Fig 4: Topology created in the OpenDayLight's controller 'topology' tab, using mininet.

### D. Nodes

By clicking on the Nodes tab the information about each switch in the network can be viewed (Fig 5).

Node Id	Node Name	Node Connectors	Statistics
openflow2	None	4	Flows   Node Connectors
openflow3	None	3	Flows   Node Connectors
openflow1	None	3	Flows   Node Connectors

Fig 5: Nodes' descriptions

### E. REST API

Rest API is used to transfer control messages or ACL rules for the functioning of firewall. The action, source and destination IP are sent as part of the XML which is shown in the Fig 6. Below is the list of REST API's used.

#### a) To get node Inventory

GET request to

`http://localhost:8181/restconf/operational/.opendaylight-inventory:nodes/` using basic auth: 'admin', 'admin' to get the node inventory

#### b) To Allow Packets

POST request to

`http://localhost:8080/restconf/operations/sal-flow:remove-flow` with basic auth: 'admin', 'admin' with body of type XML(application/xml)

#### c) To Deny Packets

POST request to

`http://localhost:8080/restconf/operations/sal-flow:add-flow` with basic auth: 'admin', 'admin' with body of type XML(application/xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<input xmlns="urn:opendaylight:flow:service">
  <barrier>false</barrier>
  <node xmlns:inv="urn:opendaylight:inventory">
    /inv:nodes/inv:node[inv:id="openflow:2"]</node>

  <cookie>55</cookie>
  <flags>SEND_FLOW_REM</flags>
  <hard-timeout>0</hard-timeout>
  <idle-timeout>0</idle-timeout>
  <installHw>false</installHw>
  <match>
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
    </ethernet-match>
    <ipv4-destination>10.0.0.2/32</ipv4-destination>
    <ipv4-source>10.0.0.1/32</ipv4-source>
  </match>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>0</order>
          <drop-action/>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <priority>3</priority>
  <strict>false</strict>
  <table_id>0</table_id>
</input>
```

Fig 6: XML skeleton

### F. Firewall implementation

We integrated the XML and JSON parsing by writing a python script which takes three inputs from the user, the source IP, destination IP and the Action(allow/deny). In fig 7 we can see how the source and destination are set once they are taken as input from the user.

```
headers = {'Content-Type': 'application/xml'}
urlInv = 'http://localhost:8181/restconf/operational/
opendaylight-inventory:nodes/'
response = requests.get(urlInv, auth=('admin', 'admin'))

lst = json.loads(response.text)
control = list()
for item in lst['nodes']['node']:
    control.append(item['id'])

for item in control:
    print 'adding ACL Rule to switch ',item, ' : ',
    action, ' packets flowing from ',
    SourceIP, ' to ', DestinationIP
    xmlNode = xmlN1 + item + xmlN2
    xmlScript = xmlNode + xmlB1 + xmlSrcIP +
    xmlDstIP + xmlB2
    if action == 'deny':
        requests.post('http://localhost:8080/restconf/
operations/sal-flow:add-flow',
data=xmlScript, headers=headers, auth=
('admin', 'admin'))
    else:
        requests.post('http://localhost:8080/
restconf/operations/sal-flow:remove-flow',
data=xmlScript, headers=headers, auth=
('admin', 'admin'))
```

Fig 7: Python Code snippet to set rules

Now that the environment is set up and parameters initialized, we can see the working of the firewall based on the need. We have three switches which are connecting three hosts.

We first keep the OpenDayLight controller running. In parallel we open two different terminals to run the firewall and check it's working. From Fig 8 we can see that a deny ACL rule is added from the source 10.0.0.1 to the destination 10.0.0.2. This rule blocks ICMP traffic from the source to destination. Before the rule is added we ping from the source to destination to ensure connectivity which is shown in the Fig 9. Once the rule is added the switch starts to drop packets which is depicted by the empty space in the Fig 9. The icmp\_seq stops ping after 16. This ACL rule ensures that the stateless firewall is stopping the packets.

To allow the packets to flow successfully we again run the python code which asks the user for input. In Fig 9 we can see that an Allow ACL rule is added between the source 10.0.0.1 and destination 10.0.0.2. The packets resume to flow once the allow rule is added. This is depicted in the Fig 8 where icmp\_seq resumes from 44.

```
"Node: h1"
root@ODayLight:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.280 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.151 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.170 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.175 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.201 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.199 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.157 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.185 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.151 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.156 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.174 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.152 ms
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=0.157 ms
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=0.171 ms
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=0.173 ms

64 bytes from 10.0.0.2: icmp_seq=44 ttl=64 time=0.274 ms
64 bytes from 10.0.0.2: icmp_seq=45 ttl=64 time=0.181 ms
64 bytes from 10.0.0.2: icmp_seq=46 ttl=64 time=0.169 ms
64 bytes from 10.0.0.2: icmp_seq=47 ttl=64 time=0.135 ms
64 bytes from 10.0.0.2: icmp_seq=48 ttl=64 time=0.141 ms
64 bytes from 10.0.0.2: icmp_seq=49 ttl=64 time=0.192 ms
64 bytes from 10.0.0.2: icmp_seq=50 ttl=64 time=0.321 ms
64 bytes from 10.0.0.2: icmp_seq=51 ttl=64 time=0.178 ms
64 bytes from 10.0.0.2: icmp_seq=52 ttl=64 time=0.164 ms
64 bytes from 10.0.0.2: icmp_seq=53 ttl=64 time=0.174 ms
64 bytes from 10.0.0.2: icmp_seq=54 ttl=64 time=0.144 ms
64 bytes from 10.0.0.2: icmp_seq=55 ttl=64 time=0.167 ms
64 bytes from 10.0.0.2: icmp_seq=56 ttl=64 time=0.185 ms
64 bytes from 10.0.0.2: icmp_seq=57 ttl=64 time=0.182 ms
^C
--- 10.0.0.2 ping statistics ---
57 packets transmitted, 30 received, 47% packet loss, time 56245ms
rtt min/avg/max/mdev = 0.135/0.181/0.321/0.041 ms
root@ODayLight:~#
```

Fig 8: Firewall blocking and allowing ICMP packets

To ensure that these ACL rules do not alter the data flow in the other switches, we open two other terminals and ping to the third host from the source and the destination. The data flow is not altered ensuring that the other switch is unaltered. At the end of the data transfer we can see there is a 47% packet loss because of the firewall application.

```
odl@ODayLight:~$ python fw.py
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Action (allow/deny): deny
adding ACL Rule to switch openflow:3 : deny packets flowing from 10.0.0.1
to 10.0.0.2
adding ACL Rule to switch openflow:2 : deny packets flowing from 10.0.0.1
to 10.0.0.2
adding ACL Rule to switch openflow:1 : deny packets flowing from 10.0.0.1
to 10.0.0.2
odl@ODayLight:~$ python fw.py
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
Action (allow/deny): allow
adding ACL Rule to switch openflow:3 : allow packets flowing from 10.0.0.1
to 10.0.0.2
adding ACL Rule to switch openflow:2 : allow packets flowing from 10.0.0.1
to 10.0.0.2
adding ACL Rule to switch openflow:1 : allow packets flowing from 10.0.0.1
to 10.0.0.2
odl@ODayLight:~$
```

Fig 9: Source IP, Destination IP, Action user input

## VI. CONCLUSION AND FUTURE WORK

OpenDayLight controller serves as a powerful firewall application tool to block or allow data packets between two hosts. The stateless firewall created was tested across different source and destination IP's to ensure efficiency. We will be

extending this concept to serve as an efficient firewall for different types of data packets and a larger network topology with parallel execution of the firewall. SDN technologies are improving at a great pace and the learning curve in the SDN firewall domain is immense. Including port numbers to make sure that the packets from a particular port should be blocked is also on the list.

The SNMP4SDN also supports ACL rules which has also been tested. In the recent future, we will be building a custom REST API which supports the ACL rules. Later we will be extending the project to support some advanced features in the firewall application which include:

- Examining dynamic network policy updates
- Checking indirect security violation
- Stateful monitoring

## REFERENCES

- [1] Hu, Hongxin, Wonkyu Han, Gail-Joon Ahn, and Ziming Zhao. "Flowguard." *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14* (2014): n. pag. Web.
- [2] Chang, Xiaolin, Bin Wang, and Jiqiang Liu. "Network state aware virtual network parallel embedding." *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)* (2012): n. pag. Web.
- [3] Mantur, Bhimshankar, Abhijeet Desai, and K. S. Nagegowda. "Centralized Control Signature-Based Firewall and Statistical-Based Network Intrusion Detection System (NIDS) in Software Defined Networks (SDN)." *Emerging Research in Computing, Information, Communication and Applications* (2015): 497-506. Web.
- [4] Ashraf, Javed, and Seemab Latif. "Handling intrusion and DDoS attacks in Software Defined Networks using machine learning techniques." *2014 National Software Engineering Conference* (2014): n. pag. Web.
- [5] Zhang, Fengwei, Kevin Leach, Angelos Stavrou, and Haining Wang. "Towards Transparent Debugging." *IEEE Transactions on Dependable and Secure Computing* (2016): 1. Web.
- [6] "SNMP4SDN:Main." *SNMP4SDN:Main - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [7] "Main Page." *OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [8] "Controller Projects' Modules/Bundles and Interfaces." *Controller Projects' Modules/Bundles and Interfaces - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [9] Mininet. "Mininet/openflow-tutorial." *GitHub*. N.p., n.d. Web. 03 May 2017.
- [10] Mininet. "Mininet/mininet." *GitHub*. N.p., n.d. Web. 03 May 2017.
- [11] "GettingStarted:Development Environment Setup." *GettingStarted:Development Environment Setup - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [12] "GettingStarted: Eclipse." *GettingStarted: Eclipse - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [13] "OpenDaylight Controller:Gerrit Setup." *OpenDaylight Controller:Gerrit Setup - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [14] "Service Function Chaining:Main." *Service Function Chaining:Main - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [15] "OpenDaylight OpenFlow Plugin:End to End Flows." *OpenDaylight OpenFlow Plugin:End to End Flows - OpenDaylight Project*. N.p., n.d. Web. 03 May 2017.
- [16] "Using the OpenDaylight SDN Controller with the Mininet Network Emulator." *Open-Source Routing and Network Simulation*. N.p., 27 Oct. 2016. Web. 03 May 2017.
- [17] "OpenFlow - Stanford University." N.p., n.d. Web. 3 May 2017.
- [18] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys and Tutorials*, vol. 17, pp. 27-51, 2015.
- [19] D. Rawat, and S. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communications Surveys and Tutorials*, vol. pp, issue. 99, pp.1-22, 2016.
- [20] Kreutz et al, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14-76, 2015.
- [21] Astuto et al, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 1617-1634, 2014.
- [22] S. Scott-Hayward, S. Natarajan and S. Sezer, "A Survey of Security in Software Defined Networks", *IEEE Communications Surveys and Tutorials*, vol. 18, pp. 623-654, 2016.
- [23] D. Kreutz, F. Ramos, P. Verissimo, "Towards Secure and Dependable Software-Defined Networks", *HotSDN*, 2013.
- [24] S. Ali, V. Sivaraman, A. Radford, S. Jha, "A Survey of Securing Networks Using Software Defined Networking", *IEEE Transactions on Reliability*, vol. 64, pp. 1086-1097, 2015.
- [25] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys and Tutorials*, vol. 17, pp. 27-51, 2015.
- [26] D. Rawat, and S. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communications Surveys and Tutorials*, vol. pp, issue. 99, pp.1-22, 2016.
- [27] Kreutz et al, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, pp. 14-76, 2015.
- [28] Astuto et al, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys and Tutorials*, vol. 16, pp. 1617-1634, 2014.
- [29] Nishtha, M. Sood, "A Survey on Issues of Concern in Software Defined Networks," *Third International Conference on Image Information Processing*, pp. 295-300, 2015.
- [30] N. Feamster, J. Rexford, E. Zegura, "The road to SDN: an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87-98, 2014.

## APPENDIX A

### CONTRIBUTIONS

The work distribution of the project is shown below:

Prateek Sreedhar Bharadwaj	Environment Setup
Shreyas Prabhakar	Adding ACL rules and building the firewall application.
Siri Uday Shastry	Testing the application
Vivek Arakali Nagaraja Rao	Implementation of the firewall

## APPENDIX B

### CODE

```
import requests
import json
#import xml.etree.ElementTree as ET

xmlN1 = '<?xml version="1.0" encoding="UTF-8" standalone="no"?><input
xmlns="urn:opendaylight:flow:service"><barrier>false</barrier><node
xmlns:inv="urn:opendaylight:inventory">/inv:nodes/inv:node[inv:id=""

xmlN2 = ""</node>'

xmlDstIP1 = '<ipv4-destination>'
xmlDstIP2 = '/32</ipv4-destination>'

xmlSrcIP1 = '<ipv4-source>'
xmlSrcIP2 = '/32</ipv4-source>'

xmlB1 = '<cookie>55</cookie><flags>SEND_FLOW_REM</flags><hard-timeout>0</hard-timeout><idle-
timeout>0</idle-timeout><installHw>false</installHw><match><ethernet-match><ethernet-
type><type>2048</type></ethernet-type></ethernet-match>'

xmlB2 = '</match><instructions><instruction><order>0</order><apply-actions><action><order>0</order><drop-
action/></action></apply-
actions></instruction></instructions><priority>3</priority><strict>false</strict><table_id>0</table_id></input>'

headers = {'Content-Type': 'application/xml'}

SourceIP = raw_input('Source IP: ')

if len(SourceIP) == 0:

    SourceIP = '10.0.0.3'

DestinationIP = raw_input('Destination IP: ')
```



```

if len(DestinationIP) == 0:
    DestinationIP = '10.0.0.2'
action = raw_input('Action (allow/deny): ')
if len(action) == 0:
    action = 'deny'
xmlSrcIP = xmlSrcIP1 + SourceIP + xmlSrcIP2
xmlDstIP = xmlDstIP1 + DestinationIP + xmlDstIP2
#print xmlSrcIP
#print xmlDstIP

urlInv = 'http://localhost:8181/restconf/operational/opendaylight-inventory:nodes/'
response = requests.get(urlInv, auth=('admin', 'admin'))
lst = json.loads(response.text)
control = list()
for item in lst['nodes']['node']:
    control.append(item['id'])
for item in control:
    print 'adding ACL Rule to switch ', item, ': ', action, ' packets flowing from ', SourceIP, ' to ', DestinationIP
    xmlNode = xmlN1 + item + xmlN2
    xmlScript = xmlNode + xmlB1 + xmlSrcIP + xmlDstIP + xmlB2
    if action == 'deny':
        requests.post('http://localhost:8080/restconf/operations/sal-flow:add-flow', data=xmlScript,
headers=headers, auth=('admin', 'admin'))
    else:
        requests.post('http://localhost:8080/restconf/operations/sal-flow:remove-flow', data=xmlScript,
headers=headers, auth=('admin', 'admin'))

```