Hierarchical Policy Management in SDN

COEN 233 – Term Project – Fall 2014

Kaushik Krishnakumar(kkrishnakumar@scu.edu), Lavanya Pothineni <lpothineni@scu.edu >, Wenqi Zhou (wzhou@scu.edu)

Table of Contents

Table of Figures

Abstract

In this project we propose and implement a multi layered policy management engine in SDN. The enterprise contains various level/hierarchy of network administrators based on different departments. Each level has a set of access, QoS rules. There are different kinds of rules like local rules, mandatory rules. Local rules are the rules which are present in a particular network. Mandatory rules are those which all the networks should obey. The network rules need to be bound by the global policies that are enforced. The network controller is used to enforce the policy restriction when the decision is to be made on the routing of the traffic.

1. Introduction

1.1. Objective

To Improve the policy management in SDN used in an enterprise.

1.2. What is the problem

In an enterprise network, the Input traffic monitoring, classification and triggered actions are a significant part of network operations. When network policies are defined, the number of policies for a given network can be in hundreds.

Hierarchies in network management a natural way to delegate management of network resources. For example, a network administrator may wish to delegate the authority to block external traffic to a security team. The team leader may wish to further delegate this authority to deputies, and if two deputies disagree on whether to block a particular flow, the leader may want a conservative conflict-resolution policy, such as “deny overrides allow.”

In this paper, we propose to hierarchical policy management in SDN.

1.3. Why this is a project related the this class

This project explores SDN and Open Flow concepts, software systems and protocols that are part of Computer Networks course.

1.4. Why other approach is no good

Flat File or single Hierarchy policy management does not allow flexibility and is error prone. When the number of policies maintained is very large, having a non-hierarchical management is error prone and also sharing of policies difficult. In a non-hierarchical policy tree, the common policies need to be replicated to all the devices. It is also very difficult and error prone to maintain the common policies for all the devices in sync.

1.5. Why you think your approach is better

Since we use policy hierarchy, we centralize the policy management to an extent. We can define organization level policies and also have the flexibility of delegation of the device policy management.

1.6. Area or scope of investigation

In this paper, we will be investigating the Hierarchical policy management in SDN using Floodlight Controller and Mininet Network emulator. We will be establishing hierarchy in a Firewall Access Control List and provision it to the SDN Controller thereby updating the SDN network.

2. Theoretical bases and literature review

2.1. Theoretical background of the problem

A policy is a set of rules or parameters that define a particular aspect of network configuration.

Access lists are used to control network access or to specify traffic for many features to act upon. The ordering of the rules is very important, as traffic flows are assigned the first rule whose definition matches the flow (known as first matching)

Rule-based policies contain one or more rules that govern how to handle traffic on a selected device, such as the access rules and inspection rules defined as part of a firewall service. Rule-based policies can contain hundreds or even thousands of rules arranged in a table, each defining different values for the same set of parameters.

Floodlight SDN Controller

- Open Flow – works with physical- and virtual- switches that speak the OpenFlow protocol
- Apache-licensed – lets you use Floodlight for almost any purpose

Open Flow is a open standard managed by Open Networking Foundation. It specifies a protocol through switch a remote controller can modify the behavior of networking devices through a well-defined "forwarding instruction set". Floodlight is designed to work with the growing number of switches, routers, virtual switches, and access points that support the Open Flow Standard.

Mininet

Mininet is a *network emulator*. It runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses lightweight virtualization to make a single system look like a complete network, running the same kernel, system, and user code. A Mininet host behaves just like a real machine; you can ssh into it (if you start up sshd and bridge the network to your host) and run arbitrary programs (including anything that is installed on the underlying Linux system.)

The programs you run can send packets through what seems like a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like a real Ethernet switch, router, or middle box, with a given amount of queuing. When two programs, like an iperf client and server, communicate through Mininet, the measured performance should match that of two (slower) native machines.

2.2. Related research to solve the problem

[1] Discusses the implementation of policy framework management within the SDN Controller.

[2] Proposes a policy layer on top of SDN Controller. This approach discusses a policy descriptive language

[3] Discusses *Hierarchical Flow Tables* (HFT), a frame- work for specifying and realizing hierarchical policies in software defined networks. HFT resolves conflicts with user-defined conflict-resolution operators and also a compiler that realizes HFT policies on a distributed network of Open Flow switches. The above approach implements Hierarchy at a switch level.

2.3. Your solution to solve this problem

Our solution to the problem of large number of policies and duplication among multiple devices is using Hierarchical Policy Management. The policy management is done at the management layer using shared policies that can be provisioned to the SDN Controller.

2.4. Where your solution different from others

The other solutions try to solve the problem in the SDN control plane or the Data plane. We try to solve the problem in the management layer.

2.5. Why your solution is better

Our solution decentralizes the management across multiple organization hierarchies. The approach is less error prone because there is only one copy of the shared policies across the network management plane.

This reduces error during deployment. Global policies can be enforced across the network in a transparent manner.

3. Hypothesis (or goals)

3.1. Positive/negative hypothesis

Our goal is to implement the Hierarchical policy management such that it is entirely on the management layer and thereby it can simplify policy management in SDN network.

We will be implementing Firewall Access Control List (ACL) and demonstrate the Hierarchical policy management through ACL.

The entire network is SDN Compliant. The policies that are provisioned will currently support a specific SDN Controller. In our research, we are using Floodlight Controller.

All the policies that are created will be provisioned into the SDN Controller.

4. Methodology

4.1. How to generate/collect input data

The policies will be defined in a file. New policies can also be added through the Command line interface.

Defining a SDN Controller to manage

controller <controller-ip> <port>

eg: controller 192.168.0.2 9933

Connects to an SDN controller on 192.168.0.2 port 9933

The below section mentions how to create a firewall access list and establish policy hierarchies within the list.

Define an Access Control List

access-list <list-name> [from <parent-list-name>]
eg:
access-list acl1
access-list acl2 from acl2

Add an access entry

access-entry <list-name> <permit|deny> <source IP> <source-port> <dest IP> <dest-port>

Import from a file

#import <file-name>

eg: import access-list.txt

Provision an ACL to SDN Controller

push <controller-name> <access-list-name>
eg: provision c1 acl1

These transfers the access list defined into the SDN Controller that has been configured.

4.2. How to solve the problem

The syntax of the policies is defined. We will also devise a policy engine which can take inputs from a multiple input sources. The policy Hierarchy is also a part of the policy syntax.
Our solution will be implemented as part of the following modules.

- ***Command Parser*** - Understand the command from Web or CLI or File.
- ***Policy Engine*** - Understand the Policy, Establish Hierarchical Relationships (One Plug-in per Policy Type - e.g.: Firewall Policy, Static Flow Entry, and Forwarding).
- ***Communication Layer*** - To support multiple Communication interfaces
  eg: Floodlight's HTTP interface for Firewall.

4.2.1 Policy Hierarchy

The provisioned policies will be flatted from the hierarchical model into the below form.

Mandatory Section

The Mandatory section contains rules that cannot be overridden by the local rules defined in a child policy.

Default Section

The Default section contains rules that *can* be overridden by local rules.

The following figure describes how rules are ordered in the rules table when using inheritance.

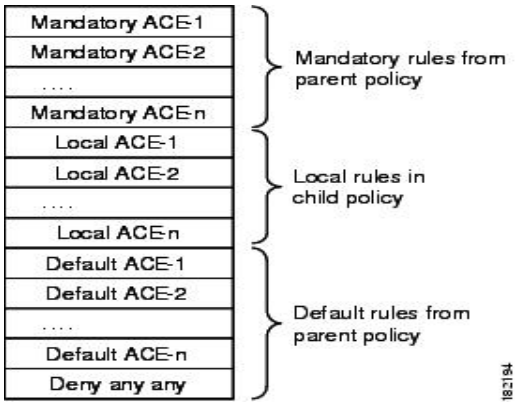


Figure 1: Order of Rules When Using Inheritance

4.3. Algorithm design

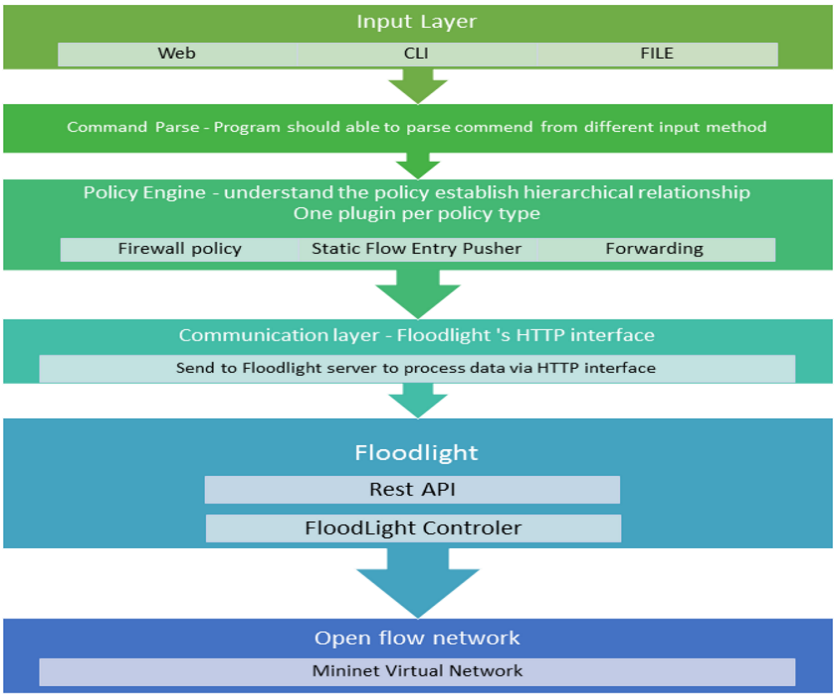


Figure 2: Flow of Data from one layer to the other

4.4. Language used

- JAVA
- Shell Script

4.5. Tools used

- Eclipse
- Mininet
- Floodlight Controller

4.6. A prototype (optional if time permit)

The prototype contains the Firewall manager CLI Application, The Mininet Virtual Machine which is provisioned with the topology mentioned below. The Floodlight Controller is also installed on the Mininet VM. The Steps Required to Provision are as follows:

1. Start the CLI Application
    a. Load the Controller Configuration
    b. Define Hierarchical Access Lists
2. Start Mininet with the required Topology
    a. The Topology is defined in the Python Script(See Section 9.XXX)
3. Provision
    a. Push an ACL to a defined Controller
4. Verify if the ACL is in effect. Switching the Floodlight Controller's Firewall Module between ON and OFF can do this.

4.7. How to generate output

Output is generated via automated scripts, which internally invoke the commands that have been defined in the above sections.

- The created Policies can be saved to a file. They can be invoked from the command

```
# import src/test/resources/acl1.txt
```

- The policies will be provisioned to the SDN Controller.

```
# provision c1 acl1
```

4.8. How to test against hypothesis

Hypothesis can be verified on using ping or similar commands on the hosts configured in mininet.

The devices' flow tables can be checked to see if the policies have been provisioned.

5. Implementation

5.1. Code (refer programming requirements)

Compiling Source Code:

The source code compiles using gradle wrapper. To compile execute the following command.

```
# ./gradlew rtJar
```

This generates a single JAR file with all the required libraries.

The jar file is generated at **build/libs/coen233-p3-0.1-rt.jar**

To Create eclipse project , invoke

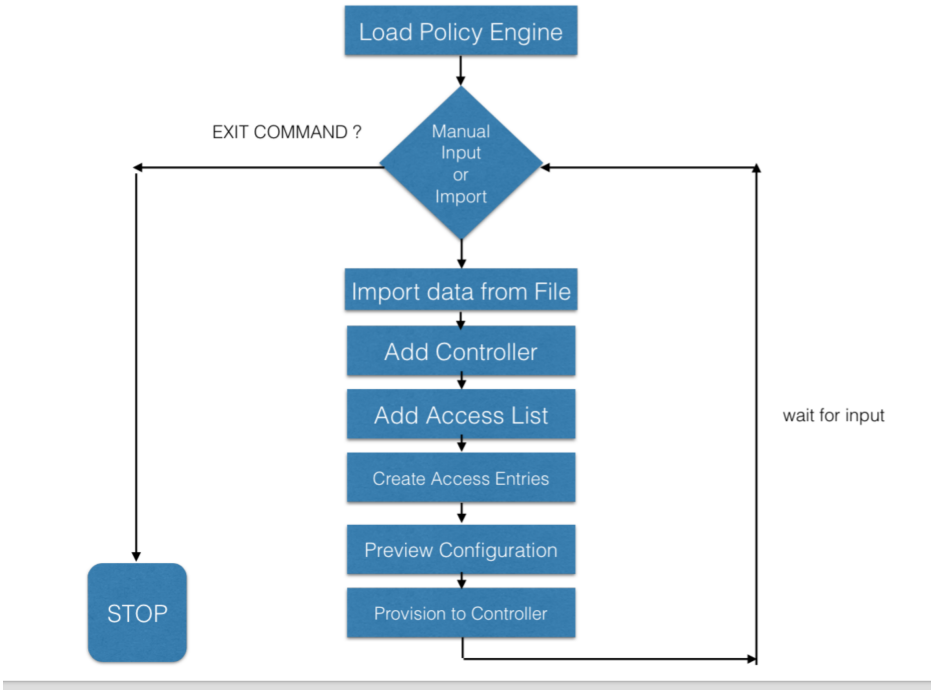
```
#./gradlew eclipse
```

This creates the .eclipse and .classpath files using which the project can be imported into eclipse

Running the Hierarchical Policy Provisioning Manager

```
# java –jar build/libs/coen233-p3-0.1-rt.jar
```

5.2. Flowchart



6. Data analysis and discussion

6.1. Output generation

The ACL entries are present in a text file which is imported into the application through 'import' command. This provides bulk provisioning. The format of the file is present in Section 9.3

Provision/Push – This pushes the data into Floodlight Controller. The sample can be seen in section 9.3

6.2. Output analysis

The firewall rules were verified in the Floodlight Controller. This was done using the Floodlight Firewall REST API. The result shows that the rules were provisioned in hierarchical order. This is denoted by the increasing order of Rule ID's for the rules in the floodlight Firewall Module. The order of rules has hence been established.

Eg: http://10.0.1.15:8080/wm/firewall/rules/json

This URL returns the Firewall Rules present on the Floodlight Controller. The MS Excel CSV extract (sorted on Rule ID) of the JSON is below.

ruleid	dl_type	nw_src_prefix	nw_src maskb	nw_dst_pre	nw_dst_mas kbits	nw_proto	tp_src	tp_d	action
2092466386	2048	10.0.0.0	24	11.0.0.16	32	6	0	80	DENY
2018489456	2048	10.0.0.0	24	11.0.0.17	32	6	0	80	DENY
2007659717	2048	10.0.0.0	24	11.0.0.11	32	6	0	80	DENY
1663640573	2048	10.0.0.0	24	11.0.0.0	24	6	0	0	ALLOW
1641046104	2048	10.0.0.0	24	11.0.0.13	32	6	0	80	DENY
1554771682	2048	10.0.0.0	24	11.0.0.15	32	6	0	80	DENY
683424488	2048	10.0.0.0	24	11.0.0.10	32	6	0	80	DENY
549335567	2048	10.0.0.0	24	11.0.0.12	32	6	0	80	DENY
306310581	2048	10.0.0.0	24	11.0.0.14	32	6	0	80	DENY
228127027	2048	192.168.5.0	24	11.0.0.0	24	6	0	0	ALLOW
76885302	2048	192.168.3.0	24	11.0.0.0	24	6	0	0	ALLOW
52361327	2048	192.168.1.0	24	11.0.0.0	24	6	0	0	ALLOW
-129615012	2048	192.168.0.0	24	11.0.0.0	24	6	0	0	ALLOW
-219166274	2048	192.168.2.0	24	11.0.0.0	24	6	0	0	ALLOW
-1149327958	2048	192.168.4.0	24	11.0.0.0	24	6	0	0	ALLOW

Figure 3: Firewall rules in Jason example

6.2.1 Before firewall rule application

```
mininet> h10_4 wget -O - h11_4
--2014-12-09 02:28:50--  http://11.0.0.4/
Connecting to 11.0.0.4:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 454 [text/html]
Saving to: 'STDOUT'
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"><html>
<title>Directory listing for /</title>
<body>
<h2>Directory listing for /</h2>
<hr>
<ul>
<li><a href="coen233_p3_topology.png">coen233_p3_topology.png</a>
<li><a href="e.py">e.py</a>
<li><a href="example/">example/</a>
<li><a href="p3.py">p3.py</a>
<li><a href="p3t.py">p3t.py</a>
<li><a href="p3t_fixed.py">p3t_fixed.py</a>
<li><a href="Readme.txt">Readme.txt</a>
</ul>
<hr>
</body>
</html>

     0K                                                       100% 9.70M=0s

2014-12-09 02:28:50 (9.70 MB/s) - written to stdout [454/454]
```

Figure 4: Screen before firewall rule application

6.2.2 After Firewall Rule application

```
mininet> h10_4 wget -O - h11_4
--2014-12-09 02:29:28--  http://10.0.0.4/
Connecting to 10.0.0.4:80... failed: Connection refused.
mininet> h10_4 wget -O - h11_4
--2014-12-09 02:29:30--  http://11.0.0.4/
Connecting to 11.0.0.4:80... failed: No route to host.
mininet>
```

Figure 5: Screen after firewall rule application

6.2.3 Check if Firewall is ON

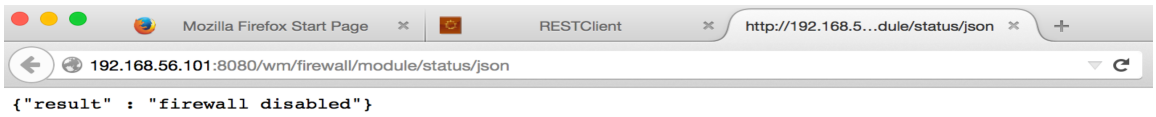


Figure 6: Checking if firewall is enabled

6.2.4 Enable Firewall

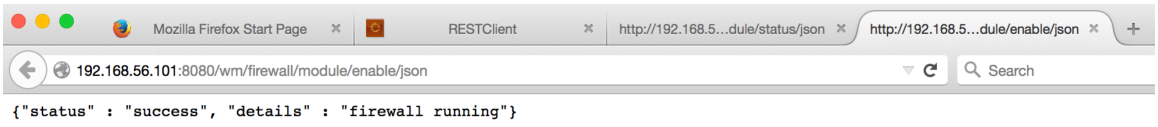


Figure 7: Enable firewall

6.2.5 Before Adding Rules

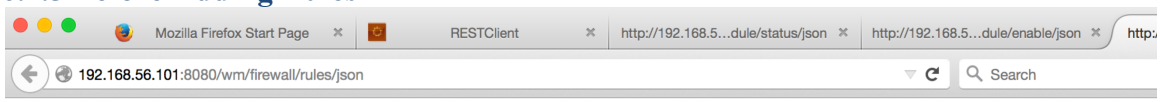


Figure 8: Before adding rules

6.2.6 Import Rules into Application

```
>show acl
AccessList [name=p1, id=0, mandatoryRules=9, localRules=0]
AccessList [name=acl1, , parentId=p1, id=0, mandatoryRules=6, localRules=8]

>controller c2 192.168.56.101 8080
Added Controller to Cache
Controller [name=c2, ipaddress=192.168.56.101, port=8080, type=1]
>show controller
Controller [name=c1, ipaddress=127.0.0.1, port=8080, type=1]
Controller [name=c2, ipaddress=192.168.56.101, port=8080, type=1]

>show acl acl1
AccessList [name=acl1, id=0, parentId=p1, mandatoryRules=[
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.1.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.2.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.4.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.5.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
], localRules=
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.1, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.2, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.3, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.4, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.5, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.6, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.7, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.8, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
]]
>show acl p1
AccessList [name=p1, id=0, parentId=null, mandatoryRules=[
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.10, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.11, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.12, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.13, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.14, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.15, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.16, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.17, destNetmask=32, destPort=80]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
], localRules=
]]
```

Figure 9: Import rules into application

6.2.7 After Adding Rules – Preview

```
>preview c2 acl1
preview
[{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.10/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.11/32","tp
-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.12/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.1
3/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.14/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":
"11.0.0.15/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.16/32","tp-dst":80},{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24",
"dst-ip":"11.0.0.17/32","tp-dst":80},{"action":"allow","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.0.0/24"
,"dst-ip":"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.1.0/24","dst-ip":"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.2.0/24","dst-ip"
:"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.0/24","dst-ip":"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.4.0/24","dst-ip":"11.0.0.
0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.5.0/24","dst-ip":"11.0.0.0/24"},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.1/32","dst-ip":"123.121.43.55/32
","tp-dst":9999},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.2/32","dst-ip":"123.121.43.55/32","tp-dst":9999},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.3/3
2","dst-ip":"123.121.43.55/32","tp-dst":9999},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.4/32","dst-ip":"123.121.43.55/32","tp-dst":9999},{"action":"allow","nw-proto":
"tcp","src-ip":"192.168.3.5/32","dst-ip":"123.121.43.55/32","tp-dst":9999},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.6/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.7/32","dst-ip":"123.121.43.55/32","tp-dst":9999},{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.8/32","dst-ip":"123.
121.43.55/32","tp-dst":9999}]
>push c2 acl1
```

Figure 10: After adding rules

5.2.8 Rules Provisioned on Floodlight Controller

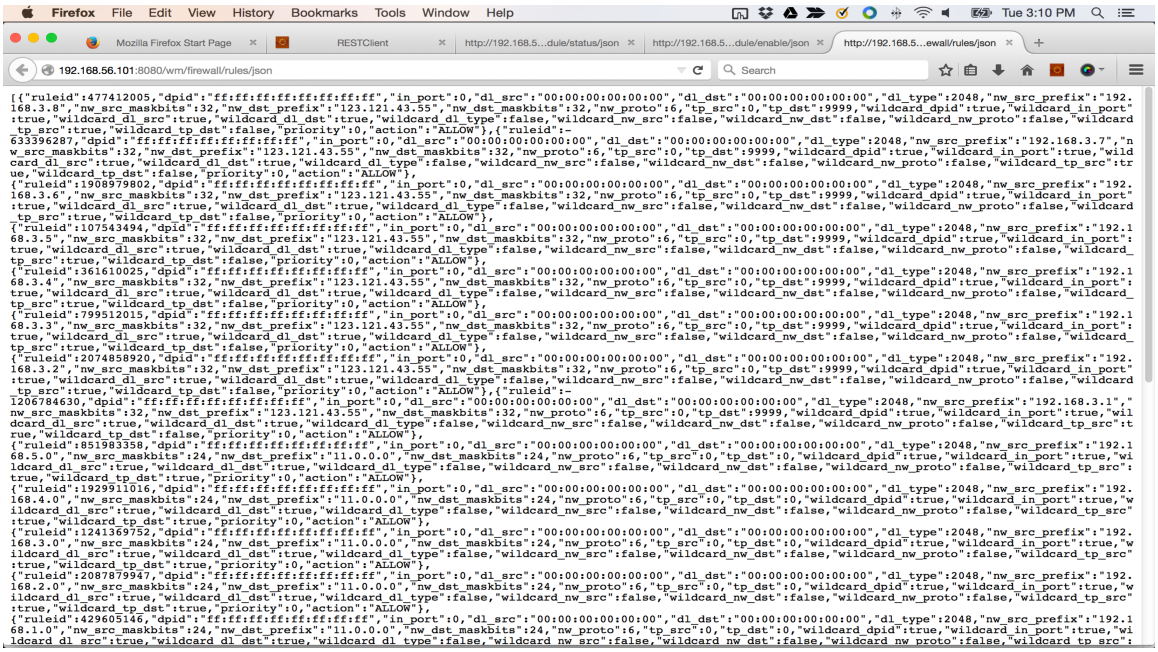


Figure 11: Rules provisioned on floodlight controller

6.3. Compare output against hypothesis

The hypothesis was verified using the method below.

1. Create HTTP Servers on port 80 for hosts 11.0.0.1-11.0.0.10
2. Access the 11.0.0.0/24 host servers from 10.0.0.0/24 hosts. Section 9.3 contains the sample output. The Server on host 11.0.0.4 responds with 200 OK message. Similar requests was done for all the HTTP Servers running on 11.0.0.0/24 network.
3. The firewall rules were provisioned using 'push' command and the HTTP access was again verified. The HTTP Server was not accessible.

This confirmed the hypothesis that the rules were provisioned in order and the hierarchical policies took effect

6.4. Abnormal case explanation (the most important task)

Multiple provisioning of the same rules does not work. When there are duplicate/overlapping or similar rules, the floodlight controller rejects the rule provisioning. Hence some of the rules that were provisioned were lost. If there is proper error handling mechanism to detect server error when provision these can be handled. Another way to handle duplicate rules is to backup the firewall rules table prior to provision and rollback when errors occur. This would ensure that the Rule tables are consistent.

6.5. Discussion

Our project is based on implementing hierarchical policy management in SDN(Software Defined Network). It helps in maintaining the policies or rules without duplication. In other words, the policies are written at several levels. It means whenever we are planning to send the data it should be passed through

the different levels of policies and will be finally sent to the destination if the all the levels are being passed by the sent data. If the data sent has a restricted level of access at a particular level, it will be blocked from sending to the destination by the policy manager.

7. Conclusions and recommendations

7.1. Summary and conclusions

We hereby conclude that Hierarchical Policy Management is an evolutionary way to improve the policy management in SDN used in an enterprise. We have demonstrated the provisioning process and the rule flattening algorithm which can be used for converting the access list to any SDN based controller.

7.2. Recommendations for future studies

The current implementation does not take into account transaction and failures that happen in the provisioning. Failsafe algorithms have to be explored for the same. The architecture can also be extended for any SDN Controller. Though the current paper explores only firewall policy, the same concept has to be explored for other network policies and how they can be included in Hierarchical Policy Management.

8. Bibliography

1. Sharma, Puneet, et al. "Enhancing Network Management Frameworks with SDN-like Control." IFIP/IEEE International Symposium on Integrated Network Management (2013): n. pag. PDF file.
2. Kim, Hyojoon, and Nick Feamster. "Improving Network Management with Software Defined Networking." IEEE Communications Magazine Feb. 2013: n. pag. PDF file.
3. Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. 2012. Hierarchical policies for software defined networks. In Proceedings of the first workshop on Hot topics in software defined networks (HotSDN '12). ACM, New York, NY, USA, 37-42.
4. Nadeau, Thomas D., and Ken Gray. SDN: Software Defined Networks. Sebastopol: O'Reilly Media, Inc., 2013. Print.
5. "SDN architecture." Open Networking Foundation. June 2014. PDF file.
6. Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: enabling innovation in campus networks. SIGCOMM Comput. Commun. Rev. 38, 2 (March 2008), 69-74.
7. Kanui, Bethany. "Floodlight REST API." Ed. Jason Parraga. *Project Floodlight*. Project Floodlight, 20 Jan. 2014. Web. 11 Nov. 2014.
8. Wang, Kuang-Ching. "Firewall REST API." Ed. Jason Parraga. *Project Floodlight*. Project Floodlight, 12 June 2013. Web. 11 Nov. 2014.
9. Wang, Kuang-Ching. "Installation Guide." Ed. Jason Parraga. Project Floodlight. Project Floodlight, 13 Dec. 2013. Web. 11 Nov. 2014.
10. Lantz. "OpenFlow Tutorial." OpenFlow. OpenFlow, 7 May 2013. Web. 11 Nov. 2014.
11. "Adding an Extended Access List." ASA 5500 Series Configuration Guide using the CLI 8.2. Cisco, 2011. Web. 11 Nov. 2014.
12. "Managing Policies." User Guide for Cisco Security Manager 4.0v. Cisco, 2010. Web. 11 Nov. 2014.

9. Appendices

9.1. Program flowchart

9.2. Program source code with documentation

The source code with documentation is available at the location
https://bitbucket.org/coen233p3/coen233p3/get/project-defense.zip

The sources are also submitted using 'Submit' tool.

9.3. Input/output listing

INPUT
The Following are the Access Control List data that was imported.

```
#smaller topology
#1 company 5 department
# define controller
controller c1 127.0.0.1 8080
acl p1
# Define ACL - Parent
# Block certain hosts before allowing network access between 9.0.0.0 and 11.0.0.0
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.10 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.11 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.12 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.13 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.14 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.15 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.16 32 80 mandatory
ace p1 deny tcp 9.0.0.0 24 -1 11.0.0.17 32 80 mandatory

acl p1 allow tcp 9.0.0.0 24 -1 11.0.0.0 24 -1 mandatory

# Block certain hosts before allowing network access between 10.0.0.0 and 11.0.0.0
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.10 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.11 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.12 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.13 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.14 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.15 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.16 32 80 mandatory
ace p1 deny tcp 10.0.0.0 24 -1 11.0.0.17 32 80 mandatory

ace p1 allow tcp 10.0.0.0 24 -1 11.0.0.0 24 -1 mandatory

# Define ACL - acl1
acl acl1 parent p1

# define ace for acl1
## allow access from 192.168.X.X to 11.0.0.0/24 network
ace acl1 allow tcp 192.168.0.0 24 -1 11.0.0.0 24 -1 mandatory
ace acl1 allow tcp 192.168.1.0 24 -1 11.0.0.0 24 -1 mandatory
ace acl1 allow tcp 192.168.2.0 24 -1 11.0.0.0 24 -1 mandatory
ace acl1 allow tcp 192.168.3.0 24 -1 11.0.0.0 24 -1 mandatory
```

```
ace acl1 allow tcp 192.168.4.0 24 -1 11.0.0.0 24 -1 mandatory
ace acl1 allow tcp 192.168.5.0 24 -1 11.0.0.0 24 -1 mandatory
# Local Rules
# Allow Hosts to access Source Code Server - 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.1 32 -1  123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.2 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.3 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.4 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.5 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.6 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.7 32 -1 123.121.43.55 32 9999
ace acl1 allow tcp 192.168.3.8 32 -1 123.121.43.55 32 9999
```

OUTPUT :

The 'show' commands provide a quick view of the data that was added into the Hierachical Policy Manager

```
# show acl
AccessList [name=p1, id=0, mandatoryRules=9, localRules=0]
AccessList [name=acl1, , parentId=p1, id=0, mandatoryRules=6, localRules=8]
```

ACL – P1 – Contains 9 Mandatory Rules

```
>show acl p1

AccessList [name=p1, id=0, parentId=null, mandatoryRules=[
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.10, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.11, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.12, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.13, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.14, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.15, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.16, destNetmask=32, destPort=80]
        AccessEntry [permit=deny, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.17, destNetmask=32, destPort=80]
        AccessEntry [permit=allow, protocol=tcp, sourceIp=10.0.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
], localRules=]]
```

ACL – ACL1 – Contains 6 Mandatory Rules , 8 Local Rules.

```
>show acl acl1

AccessList [name=acl1, id=0, parentId=p1, mandatoryRules=[
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.0.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.1.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.2.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.4.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
   AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.5.0, sourceNetmask=24, sourcePort=-1, desIp=11.0.0.0, destNetmask=24, destPort=-1]
], localRules=
```

```
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.1, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.2, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.3, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.4, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.5, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.6, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.7, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
AccessEntry [permit=allow, protocol=tcp, sourceIp=192.168.3.8, sourceNetmask=32, sourcePort=-1, desIp=123.121.43.55, destNetmask=32, destPort=9999]
]]
```

Controller C1, C2 – Floodlight Controllers running on localhost, Mininet respectively

```
>show controller
Controller [name=c1, ipaddress=127.0.0.1, port=8080, type=1]
Controller [name=c2, ipaddress=10.0.1.15, port=80, type=1]
```

Provisioned Data in Floodlight


```
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.10/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.11/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.12/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.13/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.14/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.15/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.16/32","tp-dst":80},
{"action":"deny","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.17/32","tp-dst":80},
{"action":"allow","nw-proto":"tcp","src-ip":"10.0.0.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.0.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.1.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.2.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.4.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.5.0/24","dst-ip":"11.0.0.0/24"},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.1/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.2/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.3/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.4/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.5/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.6/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.7/32","dst-ip":"123.121.43.55/32","tp-dst":9999},
{"action":"allow","nw-proto":"tcp","src-ip":"192.168.3.8/32","dst-ip":"123.121.43.55/32","tp-dst":9999}
```


Server Response before Rule Provisioning

```
mininet> h10_4 wget -O - h11_4
--2014-12-09 02:28:50--  http://11.0.0.4/ Connecting to 11.0.0.4:80... connected.
HTTP request sent, awaiting response...
200 OK Length: 454 [text/html]
Saving to: ‘STDOUT’
```

Server response after rule provisioning

```
mininet> h10_4 wget -O - h11_4
--2014-12-09 02:29:30--  http://11.0.0.4/
Connecting to 11.0.0.4:80... failed: No route to host.
```